Ingeniería Electrónica

Proyecto Final de Carrera

Automatización de máquina de ensayos de abrasión

Delmonte, Lucas Adrián

Nieto, Maximiliano

Director: Uicich, Gustavo

Año: 2017

Índice

1

Máquina, Ensayos y Requerimientos

1.1 Ensayos

El proyecto consistía en desarrollar un sistema electrónico para automatizar una máquina utilizada para realizar ensayos de abrasión, por pedido de la División Tribología de INTEMA. Primero describiremos el alcance del ensayo y las características de la máquina para comprender las funcionalidades requeridas para la automatización.

Alcance

La norma ASTM-G65 cubre este método de ensayo y los procedimientos para determinar la resistencia de materiales metálicos al desgaste por abrasión mediante un ensayo de arena seca / rueda de goma. El propósito es poder ordenar, de manera reproducible, los materiales de acuerdo a su resistencia al desgaste por abrasión bajo condiciones específicas. El desgaste por abrasión se define como el desgaste debido a partículas o protuberancias duras forzadas contra una superficie sólida en movimiento relativo.

Los resultados del ensayo de abrasión son reportados como pérdida de volumen en milímetros cúbicos para el procedimiento de ensayo en particular. Aquellos materiales con mayor resistencia a la abrasión, tendrán una pérdida de volumen menor. Este método de ensayo cubre cinco procedimientos recomendados que son apropiados para diferentes grados de resistencia a la abrasión o espesores de la muestra a ensayar.

Procedimiento

El ensayo de abrasión por arena seca / rueda de goma involucra la abrasión de una muestra estándar con granos de arena de composición y tamaño controlados. El abrasivo es introducido entre la muestra a ensayar y una rueda giratoria que debe tener una goma de clorobutilo con una dureza específica. Mediante un brazo de palanca, se presiona la muestra contra la rueda giratoria con una fuerza específica, al mismo tiempo que un flujo controlado de granos de arena abrade la superficie de la muestra. La rotación de la rueda es tal, que la cara de contacto se mueve en la dirección del flujo de arena. Un esquema básico del método de ensayo se muestra en la figura 1.1.

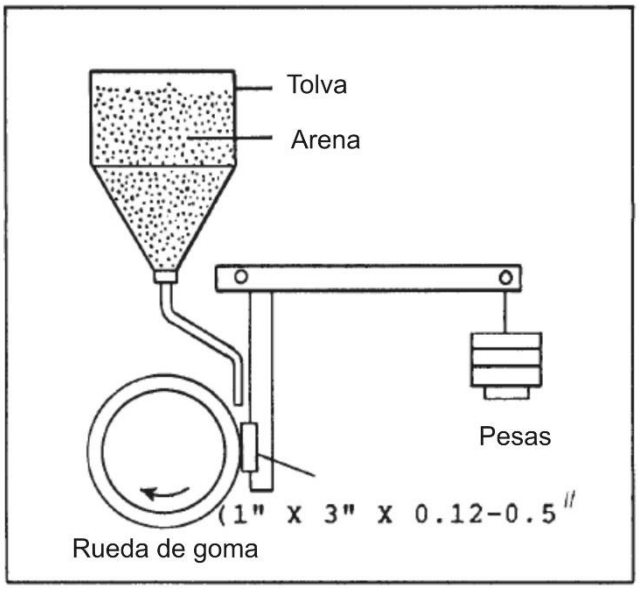


Figura 1.1: Diagrama del método de ensayo.

Al comenzar la rotación de la rueda, inmediatamente se baja el brazo de palanca cuidadosamente para que la muestra entre en contacto con la rueda, momento en el cual empieza el ensayo. La duración del ensayo y la fuerza aplicada por la palanca varían de acuerdo a lo expresado en los Procedimientos A hasta E. Las muestras deben ser pesadas antes y después del ensayo, y se debe determinar la pérdida de masa. Es necesario convertir la cantidad de masa perdida en pérdida de volumen en milímetro cúbicos, debido a la amplia diferencia entre los

valores de densidad de los materiales. La abrasión se reporta como pérdida de volumen en un procedimiento específico.

La severidad del desgaste por abrasión en cualquier sistema dependerá del tamaño, forma y dureza de la partícula abrasiva; la magnitud del esfuerzo aplicado por la partícula y la frecuencia de contacto con el abrasivo. En la figura 1.2 se muestra la marca de desgaste sobre el material ensayado.

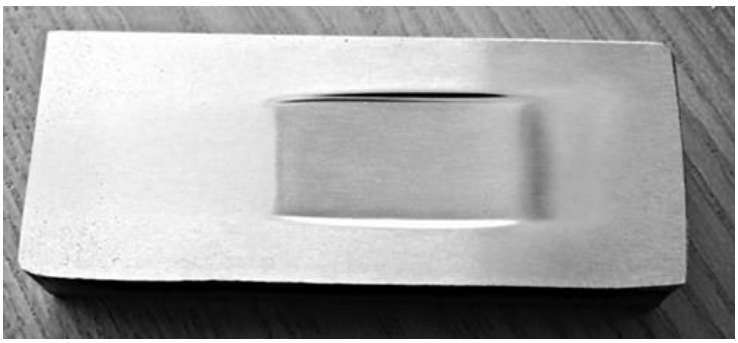


Figura 1.2: Marca de desgaste sobre el material ensayado.

Parámetros del ensayo

Los parámetros para los cinco procedimientos estándar son los siguientes:

- La fuerza a aplicar contra la muestra de ensayo y el número de revoluciones para cada procedimiento de ensayo desde el A al E.
- La tasa de flujo de arena debe ser 300 a 400 g/min. El tiempo de ensayo será aproximadamente 30 min para los Procedimientos A y D, 10 min para el Procedimiento B, 5 min para el Procedimiento E, y 30 s para el Procedimiento C, dependiendo de la velocidad real de la rueda. En todos los casos, el parámetro controlador es la cantidad de revoluciones de la rueda y no el tiempo.
- La distancia lineal de la abrasión por rayado desarrollada utilizando una rueda de 228.6 mm de diámetro y el número específico de revoluciones. A medida que se reduce el diámetro de la rueda de goma, se debe ajustar el número de revoluciones para igualar la distancia de deslizamiento de una rueda nueva, o la tasa de abrasión reducida debe ser considerada ajustando el volumen perdido producido por la rueda gastada con el volumen perdido normalizado de una nueva rueda.

La tabla 1 resume los parámetros más importantes.

Procedimiento Específico	Fuerza aplicada a la muestra[A], N	Revoluciones de la Rueda	Abrasión Lineal, m
A	130	6000	4309
B	130	2000	1436
C	130	100	71.8
D	45	6000	4309
E	130	1000	718

Tabla 1: Parámetros de ensayo para los cinco procedimientos estándar.

Rectificación y materiales de ensayo

Para la preparación y cuidado de la rueda de goma se requiere rectificar la periferia de la rueda de goma. El objetivo es producir una superficie uniforme que correrá tangente a la muestra sin causar vibraciones o saltos en el brazo palanca. La rueda puede ser utilizada hasta que el diámetro se desgaste hasta 215.9 mm. Para la rectificación se recomiendo una velocidad de giro en la rueda de 200 RPM.

Por último, se destacan tres materiales de referencia con los procedimientos a realizar sobre los mismos:

- Acero para Herramientas AISI D-2 (Tipo No libre Corte) — Este es el Material de Referencia No. 1 Para el Procedimiento A.
- Acero para Herramientas AISI H-13— Este es el Material de Referencia No. 2 para el Procedimiento B.
- Acero AISI 4340 — Este es el Material de Referencia No.3 para los Procedimientos B o E.

1.2 Máquina

La construcción de la máquina sigue el diagrama básico de la figura 1.1. Imágenes desde el frente y de atrás de la misma se observan en las figuras 1.3 y 1.4 respectivamente.

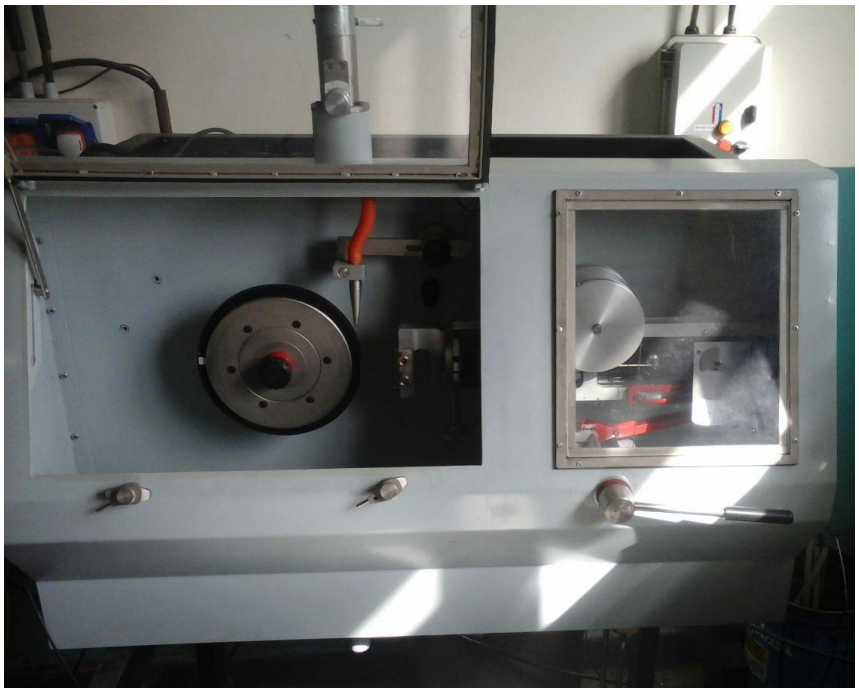


Figura 1.3: Máquina vista de frente.

Figura 1.4: Máquina vista desde atrás.

De frente se pueden apreciar la rueda de goma, el soporte para la muestra, la boquilla por la cual pasa el flujo de arena, una cubierta de protección y un sistema mecánico accionado por una palanca para aplicar una carga sobre el material de ensayo. En la parte trasera aparece un gabinete plástico para la colocación de la placa que controlara el proceso, el motor para hacer girar la rueda de goma y un sensor inductivo.

Las revoluciones por minutos (200 ± 10 RPM) deben permanecer constantes bajo carga. La norma exige la instalación de cualquier motor que produzca 200 RPM bajo carga. Para esta máquina se instaló un motor asincrónico trifásico con reductor, que más adelante será caracterizado.

Por otro lado, la máquina está equipada con un contador de revoluciones que monitoreará el número de revoluciones de la rueda como se especifica en el procedimiento. Se recomienda que el contador incremental pueda detener la máquina cuando se alcance una cantidad preseleccionada de revoluciones o hasta 12 000 revoluciones. Esta función es realizada con un sensor inductivo ya instalado en la máquina, cuyo funcionamiento será analizado.

1.3 Requerimientos

Para la automatización de esta máquina se requiere de un sistema que realice distintas funciones, que se detallan a continuación:

- El control del motor trifásico, mediante un variador de velocidad comercial. Con el uso de este equipo es posible controlar el momento de arranque y detención del motor, además de definir la velocidad y sentido de giro del eje.
- Definir la duración del ensayo en función del recorrido tangencial del disco. Esta función se realiza procesando la información otorgada por un sensor inductivo, ya instalado en la máquina.
- La instrumentación de una celda de carga, para medir el esfuerzo realizado sobre la carga. Esto consiste en la instalación de una celda de carga, y el diseño de un circuito para amplificar y acondicionar la señal entregada por la misma.
- La instalación de un sensor, que permita determinar si la carga esta efectivamente colocada.
- La implementación de un sistema que le brinde la posibilidad al usuario de guardar todos los datos del ensayo.
- La alimentación de una tira de diodos LED, que ya se encontraba instalada y sirve para iluminar el interior de máquina.
- El desarrollo de una interfaz con el usuario, que le permita al mismo definir todos los parámetros del ensayo.
- Hacer la instalación eléctrica correspondiente.

Ya con la descripción del ensayo, la máquina a automatizar y los requerimientos del usuario, en los siguientes capítulos se detallará el diseño electrónico para cumplir con los puntos desarrollados. Para comprender mejor el diseño, en la figura 1.5 se muestra un diagrama en bloques de la solución propuesta.

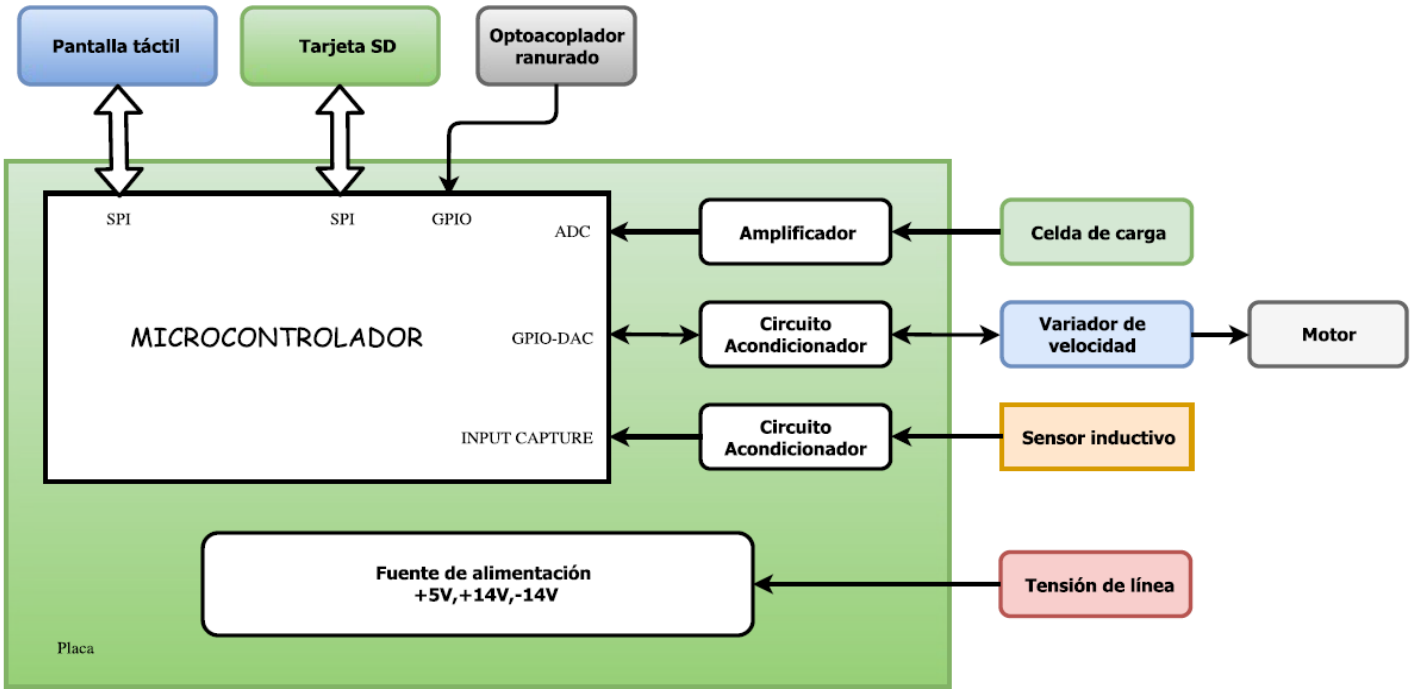


Figura 1.5: Diagrama en bloques del sistema.

2

Fuente de alimentación

2.1 Fuentes conmutadas

Para la alimentación del circuito se decidió implementar una SMPS (Switching Mode Power Supply), que tiene como gran ventaja su elevada eficiencia. La baja significativa de la potencia de pérdida de la fuente conmutada, en comparación con una fuente lineal, tiene que ver con que el transistor deja de trabajar en su zona lineal (como lo hace en un regulador lineal) y ahora trabaja en corte y saturación a alta frecuencia.

En la figura 2.1 se puede observar el diagrama básico de cómo trabaja una SMPS. Se puede ver como la tensión de línea, u, es rectificada y filtrada. Esta tensión es reducida y pulsada a alta frecuencia por el transformador y transistor respectivamente, y finalmente filtrada para producir la tensión de salida regulada, U_0. La tensión de salida es realimentada, comparada con la tensión de referencia, para luego ser amplificada por el amplificador de error que produce la señal que será comparada con un oscilador (de forma de onda triangular) para producir la señal de PWM que comanda el transistor. La variable de regulación es el ancho del pulso de la onda cuadrada de alta frecuencia que comanda el transistor, cuanto más grande sea el ciclo de trabajo de esta señal, mayor será la potencia entregada en la salida.

Las SMPS tienen como desventajas, frente a las fuentes lineales, una tensión menos estable, con mayor ripple y ruido en la salida, las interferencias de alta frecuencia que genera y que pueden afectar a otros equipos o a la línea, además de ser un circuito mucho más complejo para diseñar. Aunque la elevada eficiencia y reducido volumen, producto de que el transformador opera en alta frecuencia, terminaron definiendo el uso de una fuente conmutada.

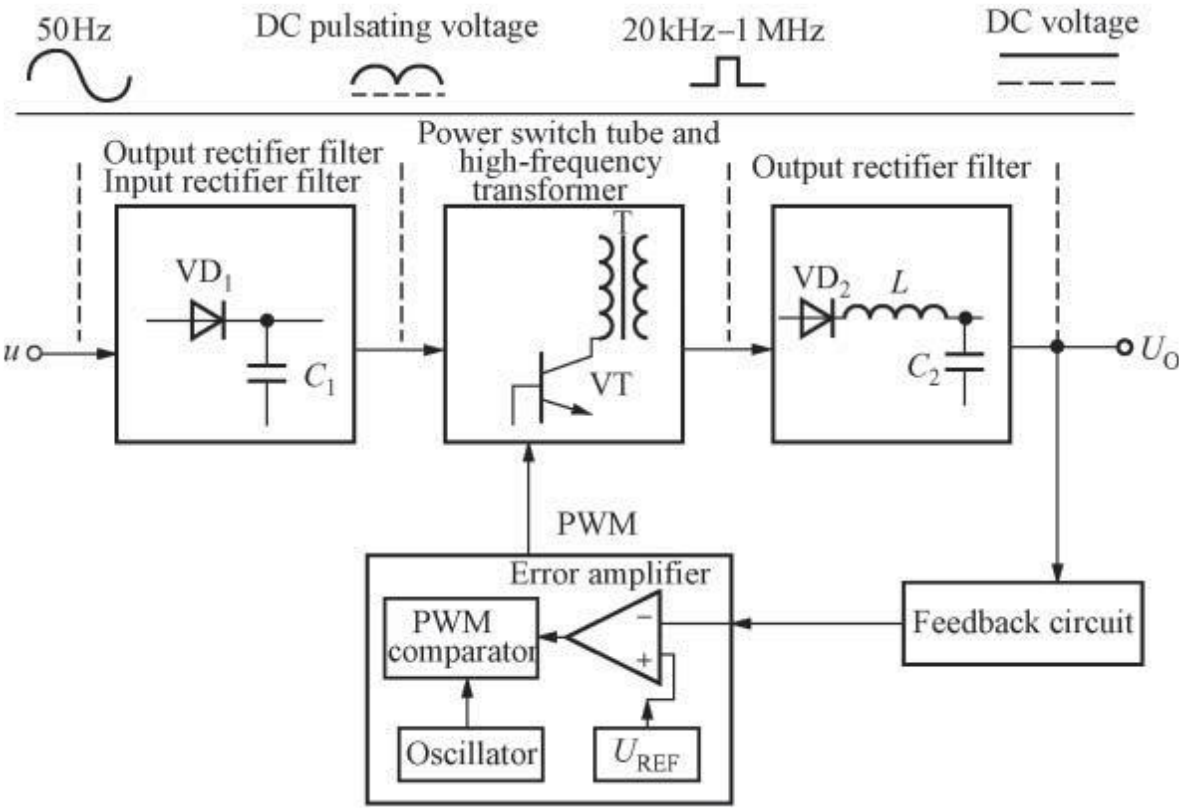


Figura 2.1: Diagrama de la fuente de alimentación conmutada.

2.2 Convertidor Flyback

Las fuentes conmutadas se basan en un convertidor de continua a continua, cuya función es transformar un nivel de tensión en la entrada a otro nivel de tensión en la salida. Hay una gran cantidad de convertidores, de los cuales se descartaron aquellas topologías que no ofrezcan aislación. Dentro de las topologías restantes se adoptó el convertidor Flyback, debido a su simplicidad y bajo costo. En la figura 2.2 se puede observar el circuito del convertidor Flyback, que deriva del convertidor Buck-Boost.

Cuando la señal que comanda la llave está en alto, la llave se cierra y se carga la inductancia del primario, mientras que el capacitor es el que se encarga de suministrar energía en la salida. En cambio cuando la señal PWM está en bajo, la llave se abre y se transfiere la energía almacenada por el bobinado primario al secundario y se vuelve a cargar C.

Este proceso, que se puede observar en la figura 2.2, ocurre a alta frecuencia y la cantidad de energía que se transfiere en cada ciclo lo determina el ciclo de trabajo de la señal de PWM.

Las formas de onda del convertidor Flyback se pueden observar en la figura 2.3. La forma de onda verde es la señal de PWM, la azul es la corriente del bobinado primario, la roja es la corriente del bobinado secundario, la rosa es la tensión de drain del MOSFET utilizado como llave y la gris es la tensión de salida.

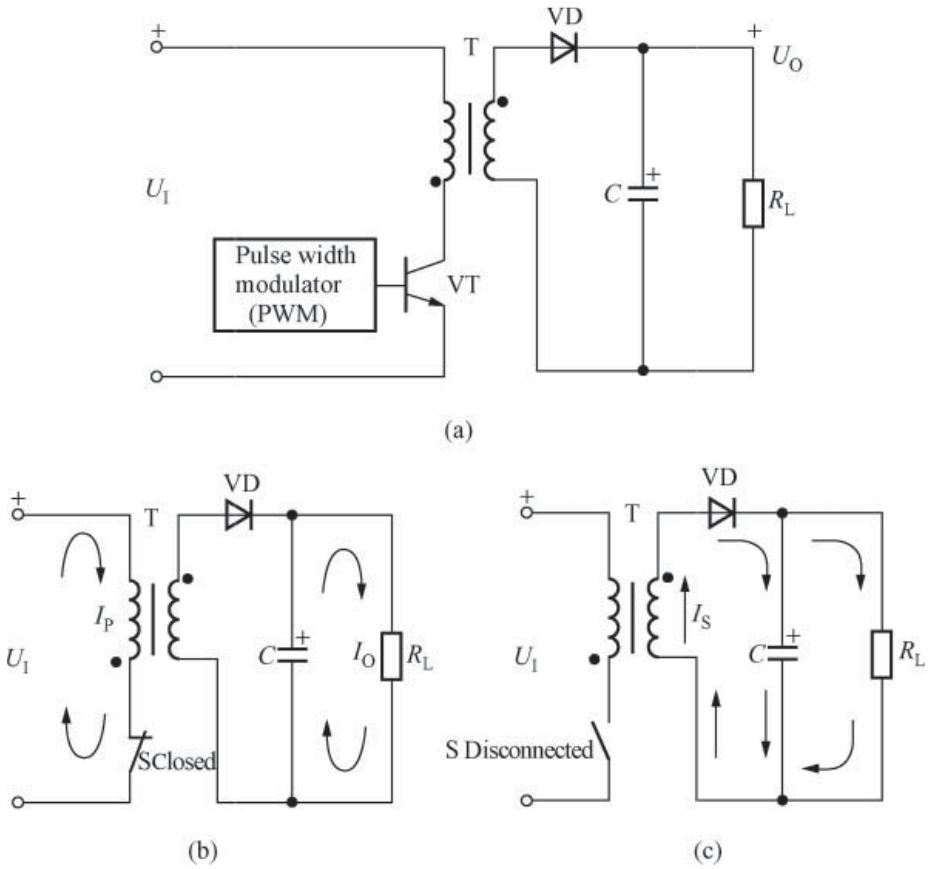


Figura 2.2: Convertidor Flyback.

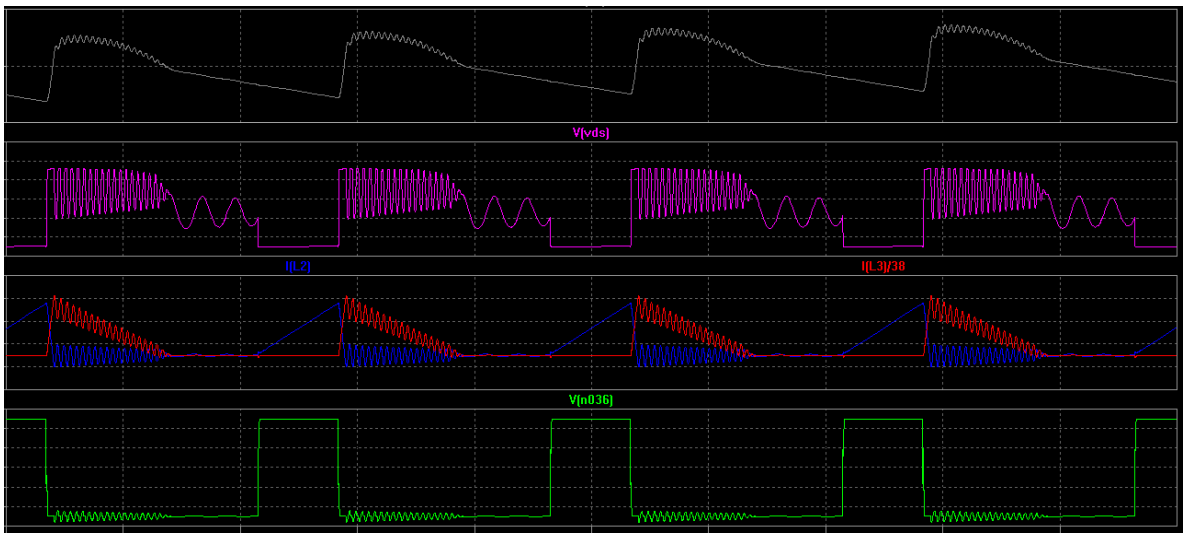


Figura 2.3: Formas de onda del convertidor Flyback.

En el tiempo que la señal de PWM está en alto, la llave permanece cerrada y se puede observar cómo se carga la inductancia del primario. Una vez finalizado este tiempo, T_{ON}, la corriente

del primario llega a su amplitud pico. Este valor depende del modo de operación en el que se encuentre el convertidor, los distintos modos de operación serán analizados en la siguiente sección. La ecuación 2.1 expresa la corriente pico en la inductancia del primario para el convertidor operando en conducción discontinua o en modo de transición:

$$I_{PP} = \frac{T_{ON} * V_{IN}}{L_P} = \frac{D * T_S * V_{IN}}{L_P} \qquad (2.1)$$

Donde V_{IN} es la tensión de entrada, D es el ciclo de trabajo de la señal de PWM, L_P es la inductancia del primario y T_S es el periodo de conmutación. Si el convertidor opera en conducción continua la expresión anterior es simplemente el aumento de corriente, para obtener el valor pico hay que sumar la corriente promedio en la inductancia, resultando la ecuación 2.2:

$$I_{PP} = I_{L_{PROM}} + \Delta I_L \qquad (2.2)$$

La corriente promedio en la bobina primaria está relacionada con la corriente promedio en la entrada, como se indica en la ecuación 2.3:

$$I_{IN_{PROM}} = D * I_{L_{PROM}} \qquad (2.3)$$

A su vez la corriente promedio en la entrada está relacionada con la potencia de entrada, como lo expresa la ecuación 2.4:

$$P_{IN} = I_{IN_{PROM}} * V_{IN} \qquad (2.4)$$

Por otro lado, cuando el convertidor opera en conducción discontinua o en modo de transición, el ciclo de trabajo define I_{PP}, y a su vez esta define la energía que se almacena en la bobina del primario, que luego será transferida al secundario cuando la llave se abra. La energía por ciclo para el convertidor operando en conducción discontinua se expresa en la ecuación 2.5:

$$E = \frac{1}{2} * L_P * {I_{PP}}^2 \qquad (2.5)$$

Si a la ecuación 5 se la multiplica por la frecuencia de conmutación, f_S, se obtiene la potencia en el primario como se muestra en la ecuación 2.6:

$$P_{IN} = f_S * E = \frac{1}{2} * L_P * {I_{PP}}^2 * f_S \qquad (2.6)$$

En conclusión, el ciclo de trabajo de la señal de PWM es lo que determina la potencia que se entrega en la salida.

La corriente en el secundario es N_{PS} veces más grande que la corriente en el primario, donde N_{PS} es la relación de vueltas entre el primario y el secundario. Cuando la llave se abre esta corriente vuelve a cargar el capacitor de salida.

En la tensión de salida se puede ver el ripple producto de la carga y descarga del capacitor de salida. Cuando la corriente en el secundario está en su valor pico, el capacitor se carga a su máximo valor de tensión, y en cambio al final de T_{ON} está en su mínimo.

En todas las señales la frecuencia fundamental es la de conmutación de la llave, sin embargo aparecen oscilaciones de muy alta frecuencia de circuitos resonantes LC, producto de las

inductancias de pérdida en cada bobinado y las capacidades parásitas de los componentes de la fuente.

En cuanto a las desventajas de esta topología, se pueden mencionar dos principales, ambas asociadas a las grandes corrientes RMS en los bobinados. Una es que debido a estas corrientes se producen pérdidas RMS que limitan la potencia máxima de la SMPS, es por esto que para grandes potencias se recomienda el uso de otra topología. La otra desventaja es común a todas a las topologías y es que estas corrientes, a través de capacidades parásitas en los componentes, crean un camino de retorno a línea de 220V que producen una interferencia. Esto último crea la necesidad de agregar un filtro de interferencias electromagnética, EMI.

Por último, en esta topología cabe mencionar que en realidad más que un transformador son bobinas acopladas que almacenan y liberan energía. Además al conmutar a alta frecuencia es de volumen reducido, a diferencia de los transformadores que trabajan a frecuencia de línea en las fuentes lineales. Esta topología también facilita el agregado de otras salidas, simplemente añadiendo bobinados en el secundario. La característica anteriormente mencionada es importante debido a la necesidad de salidas de +5V y ±15V.

2.3 Modos de operación

Una vez adoptada la topología hay que elegir uno de los tres modos de operación que tiene el convertidor, estos modos son el modo de conducción continua (CCM), el modo de conducción discontinua (DCM) y el modo de transición (TM). Lo que define en qué modo de operación se encuentra el convertidor es si en T_{ON} la corriente empieza a crecer desde cero o no.

En CCM cuando la llave se cierra, la corriente empieza a crecer desde un valor distinto de cero, como se puede ver en la figura 2.4. En este modo de operación se puede ver que no toda la energía es transferida al secundario y además se puede saber cuál es el valor promedio del ciclo de trabajo en régimen permanente mediante las tensiones de entrada y salida y la relación de vueltas entre primario y secundario del transformador.

En DCM la inductancia del primario empieza a cargarse desde cero, como se ilustra en la figura 2.5. A diferencia de CCM, aquí toda la energía es transferida al secundario y se pueden destacar tres tiempos diferentes. Durante el tiempo de encendido de la llave, la energía se empieza a almacenar en la inductancia del primario. Luego, cuando termina este tiempo y la llave se abre, empieza el tiempo de desmagnetización, T_{DM}, que es el tiempo que se toma en transferir la energía de la bobina secundaria al capacitor de salida. Cuando la transferencia de energía termina, termina T_{DM}, y empieza el tiempo de conducción discontinua, T_{DIS}, en el cual la corriente en el secundario es cero y termina cuando empieza el próximo ciclo.

Finalmente TM es un caso especial, límite entre DCM y CCM, en el que la corriente del secundario llega a cero y la energía es transferida completamente pero con T_{DIS} igual a cero, es decir, que cuando se completa la transferencia de energía inmediatamente empieza el próximo ciclo.

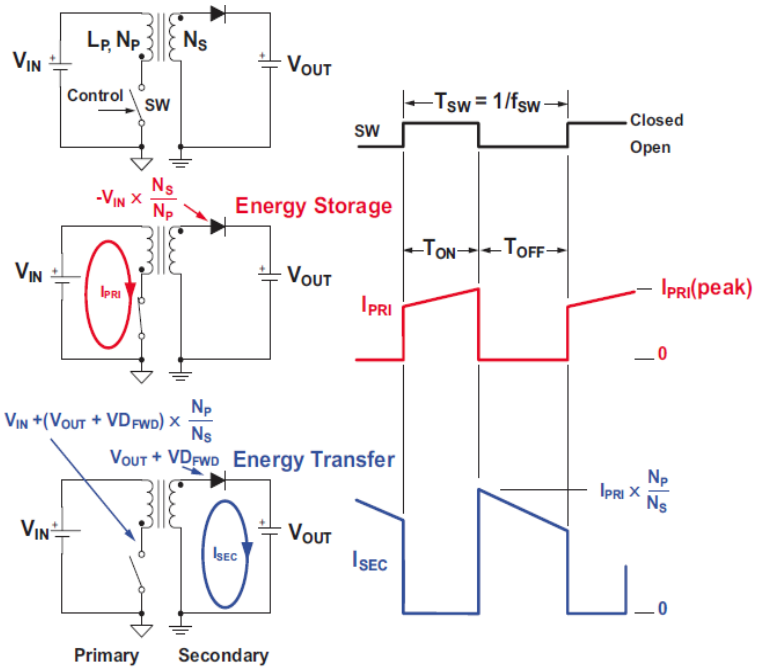


Figura 2.4: El convertido Flyback en CCM.

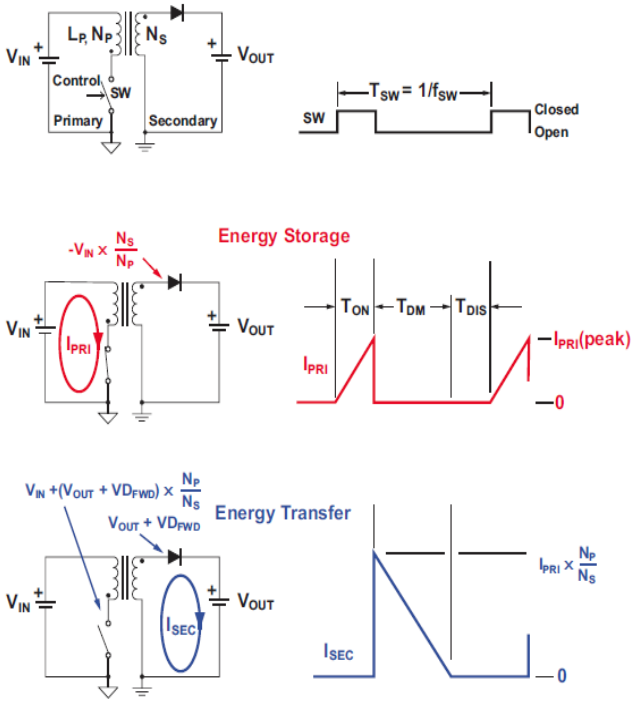


Figura 2.5: El convertidor Flyback en DCM.

Mirando las formas de onda de los distintos modos de operación se puede ver como la forma de onda triangular de los modos DCM y TM tiene mayores pérdidas RMS, frente a la forma de onda trapezoidal del modo CCM. Estas pérdidas RMS se disipan en la llave de potencia, el núcleo del transformador y los del devanados del transformador. Aunque en CCM, las pérdidas por recuperación inversa del diodo, asociadas con el encendido de la llave de potencia mientras el diodo de salida aún está conduciendo, y las pérdidas por conmutación asociadas al punto de inicio distinto de cero en la corriente del primario, suelen ser mayores que las perdidas RMS en DCM. Por lo tanto, ante la menor pérdida de potencia y la mejor dinámica que ofrece DCM frente a CCM, se adoptó el modo de operación de conducción discontinua.

2.4 Control en modo tensión vs Control en modo corriente

Existen dos modos de control en una SMPS, el más común es el control en modo de tensión y el otro es el control en modo de corriente.

En la figura 2.6 se puede observar el esquema de una SMPS trabajando en modo de tensión. En este modo de control la tensión de salida, U_0, es continuamente muestreada resultando U_Q, y luego comparada con una tensión de referencia, U_{REF}. La diferencia entre estas dos tensiones es amplificada por el amplificador de error, dando como resultado a la tensión de error U_R. Por otro lado, un oscilador genera la señal de SET de un Flip-Flop RS a la frecuencia de conmutación y a la vez genera una forma de onda de diente de sierra de amplitud y frecuencia (frecuencia de conmutación) constantes, U_J. La comparación entre U_R y U_J define la señal de RESET del Flip-Flop. Finalmente la señal que comanda la llave se conforma de la siguiente manera: al inicio de cada ciclo el oscilador setea la salida del FF y se resetea cuando la señal U_R supera la señal U_J, esto se puede ver en las formas de onda de la figura 2.6. Queda claro que la señal de error es lo que define el ciclo de trabajo de la señal de PWM, y por lo tanto la potencia que se transfiere a la salida.

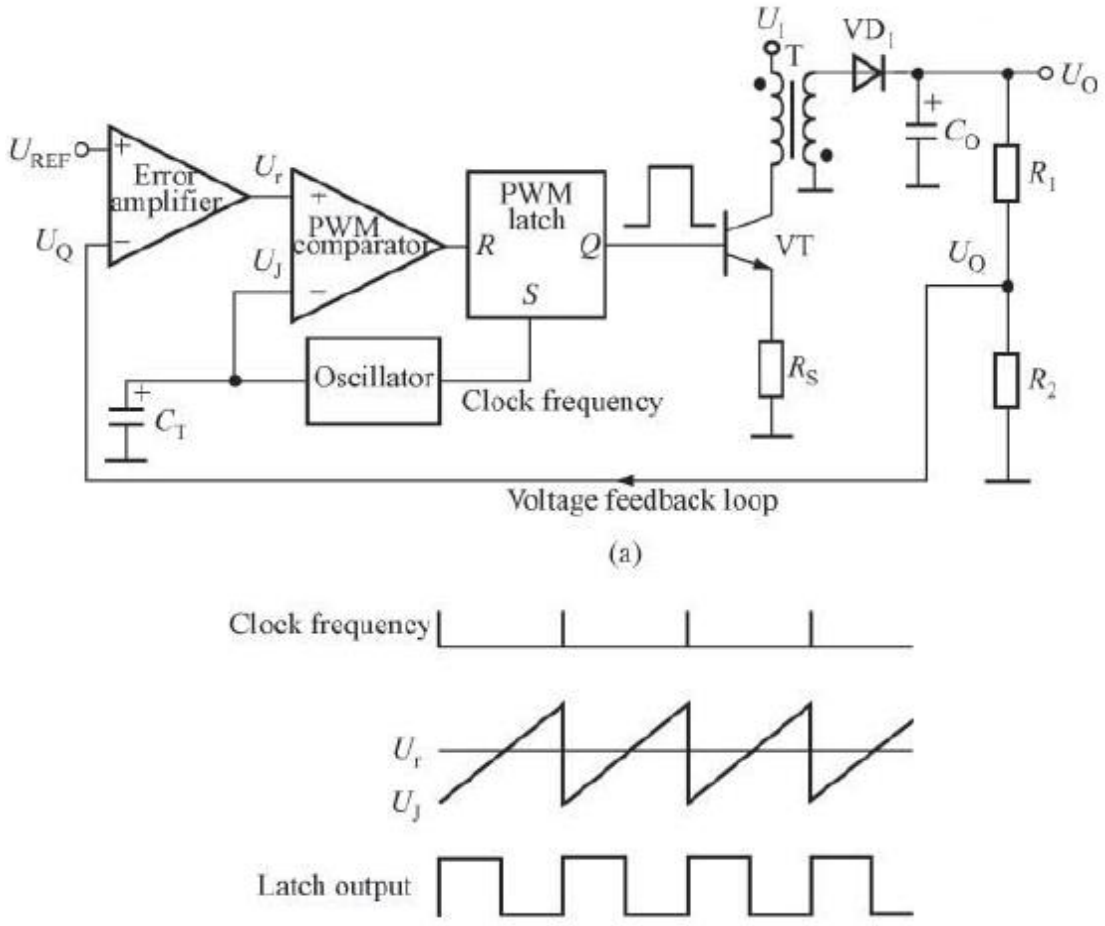


Figura 2.6: El convertidor Flyback con control en modo de tensión.

Este modo de control presenta algunas desventajas:

- Respuesta retrasada: Si bien la resistencia R_S sensa la corriente en el primario, esta información no es incluida en el lazo de control. Por lo tanto ante un cambio en la tensión de entrada, la corriente pico en el primario y en consecuencia la potencia puesta en juego

se modifican. Pero el lazo de control no se entera hasta que la tensión de salida varia, y debido a la constante de tiempo del filtro de salida, pasan varios ciclos hasta que el lazo de control puede actuar en consecuencia. Este retardo que presenta la respuesta del lazo puede afectar la estabilidad de la tensión en la salida.

- Se necesita diseñar una protección adicional por sobrecorriente en el primario.
- La compensación del lazo de control resulta en un menor ancho de banda, debido a la respuesta dinámica que presenta este modo de control, y la ganancia a lazo cerrado cambia con la tensión de entrada.

En cambio, en el control en modo de corriente, además del lazo de tensión se agrega un lazo de corriente. La figura 2.7 muestra el funcionamiento de este modo de control. Esta configuración mantiene el lazo de tensión, pero añadiendo la caída de tensión en R_S se agrega un lazo de corriente interno al de tensión. La caída de tensión en R_S, U_S, es producida por la corriente en el bobinado primario. De esta manera aparece un pulso en la señal de RESET del FF cuando la corriente del primario supera el valor impuesto por la señal de error, tal y como muestran las formas de onda en la figura 2.7. Es decir, que ahora se controla de manera indirecta el ciclo de trabajo, y lo que se está controlando es la corriente pico en el primario.

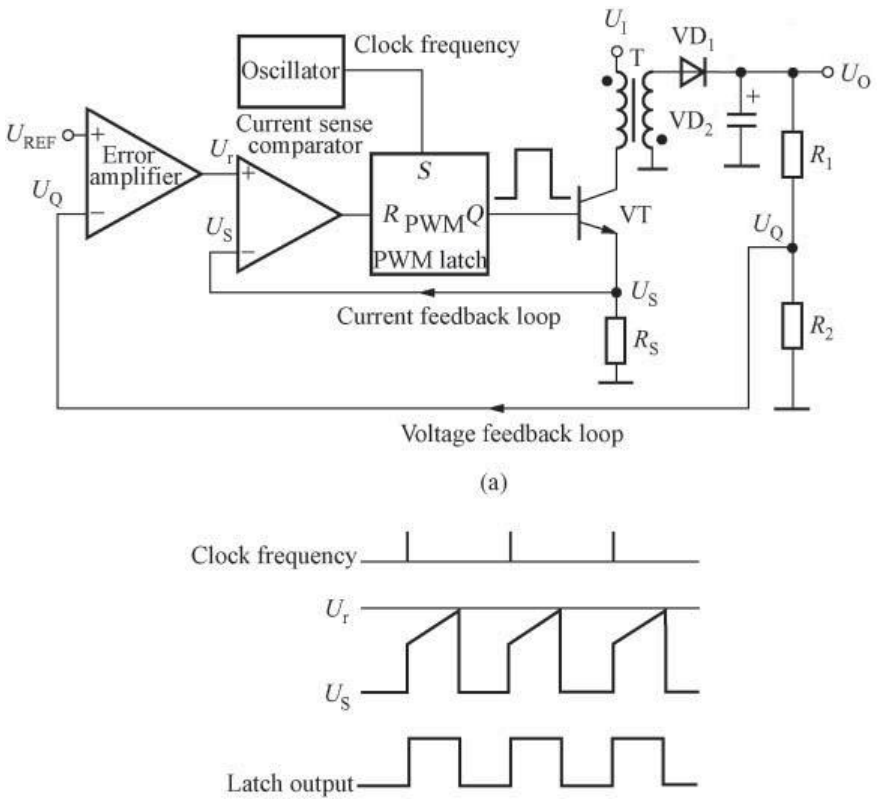


Figura 2.7: El convertidor Flyback control en modo de corriente.

Con respecto al modo de control anterior, tiene las siguientes ventajas:

- El ancho de banda del lazo de corriente es mayor al que presenta el lazo de tensión externo.

- Puede responder rápidamente a variaciones en la tensión de entrada. Esto es posible ya que adapta el ciclo de trabajo a esta nueva situación mediante el comparador de sensado de corriente, más que con el amplificador de error.
- Simplifica el diseño de la red de compensación para el amplificador de error.
- La resistencia de sensado permite un circuito de protección que limita la corriente en el primario.

Aunque esta configuración presenta algunas desventajas:

- Para ciclos de trabajo mayores al 50% se presentan inestabilidades, también llamadas oscilaciones subarmónicas, para todas las topologías en CCM y para algunas en DCM.
- Debido al ruido superpuesto a U_S y la pendiente relativamente pequeña de la corriente en el primario, es probable que ocurran falsos pulsos de RESET.
- El encendido y apagado de la llave generan picos en la corriente que circula por R_S y por lo tanto en U_S, que generan falsos pulsos de RESET. Con lo cual se requiere generar un tiempo muy pequeño (mayor o igual al tiempo de duración de estos picos) de blanking, tiempo durante el cual la señal U_S no podrá generar pulsos de RESET.

En conclusión, las ventajas que ofrece el modo de corriente determinaron la elección del mismo. Además las inestabilidades por oscilaciones subarmónicas no aplican a la topología Flyback trabajando en DCM, y por otro lado los falsos pulsos de RESET generados por ruido y picos en U_S deben ser manejados por el controlador, y no deben ser tenidos en cuenta en el diseño.

2.5 Primary Side Regulation vs Secondary Side Regulation

Antes de definir el controlador a utilizar, faltaba definir si se utilizaría regulación desde el primario o secundario. Haremos un análisis de ambas opciones para definir cuál utilizar, comenzando con la regulación desde el lado secundario.

Cuando se realimenta desde el secundario y se quiere mantener la aislación entre primario y secundario, se suele usar usa un opto-acoplador. La única conexión entre el circuito primario y secundario es en forma óptica, que implica una resistencia de aislación del orden de los MΩ. El esquema eléctrico del opto-acoplador es mostrado en la figura 2.8 y, está compuesto por un diodo LED que emite luz a un fototransistor. La corriente que circula en el colector del fototransistor es proporcional a la que pasa a través del diodo LED, a esta relación se la denomina Current Transfer Ratio, CTR. El CTR a su vez depende del valor de la corriente que circula por el diodo LED, la temperatura y la tensión colector emisor del fototransistor, esto debe ser tenido en cuenta en el diseño.

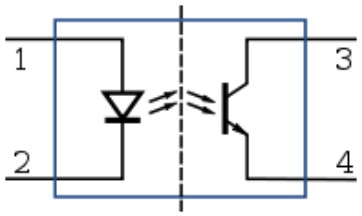


Figura 2.8: Esquema eléctrico del opto-acoplador.

En las SMPS con regulación desde el secundario, suele usarse el regulador shunt ajustable TL431. Este dispositivo, cuyo símbolo, encapsulado TO-92 y diagrama de funcionamiento se pueden ver en la figura 2.9, se usa como referencia de precisión y amplificador de error con salida open-collector al mismo tiempo. El TL431 es capaz de ajustar su tensión de cátodo, V_{KA}, desde su tensión de referencia, V_{REF}, de 2.495V hasta 36V. Para alimentar el dispositivo se necesita de una corriente de cátodo de al menos 1mA.

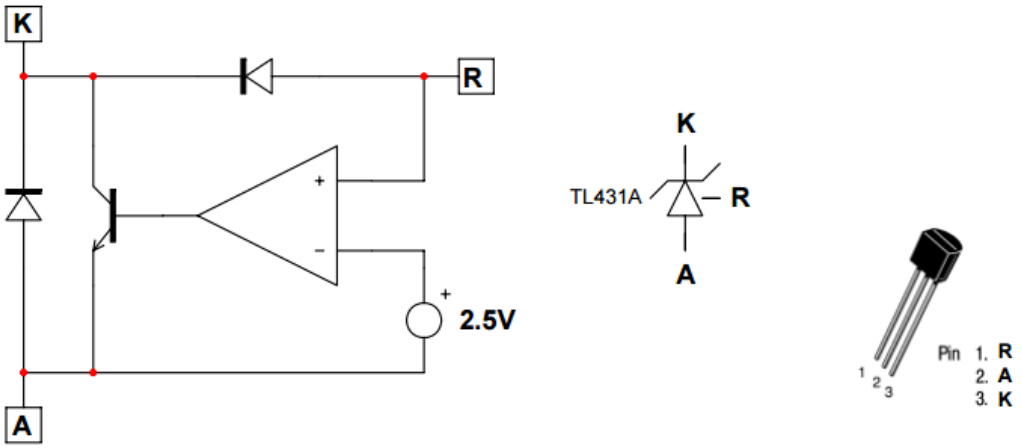


Figura 2.9: El TL431.

Un típico circuito de realimentación usando un opto-acoplador y TL431 se muestra en la figura 2.10. Las resistencias R_1 y R_{LOWER} sensan la tensión de salida, además ajustan el nivel de tensión que se desea junto con la tensión de referencia interna del TL431 y alimentan el pin de referencia. Para regular la tensión de salida el lazo ajusta la tensión de cátodo. Esta tensión produce una corriente en el diodo LED del opto-acoplador, que mediante el CTR produce una corriente en el colector del fototransistor. Por último, la corriente de colector produce una caída de tensión en la resistencia de pull-up, obteniendo finalmente la tensión de realimentación. Esta tensión de realimentación es la que le indica al controlador cuanta potencia debe entregar a la carga para regular la tensión de salida.

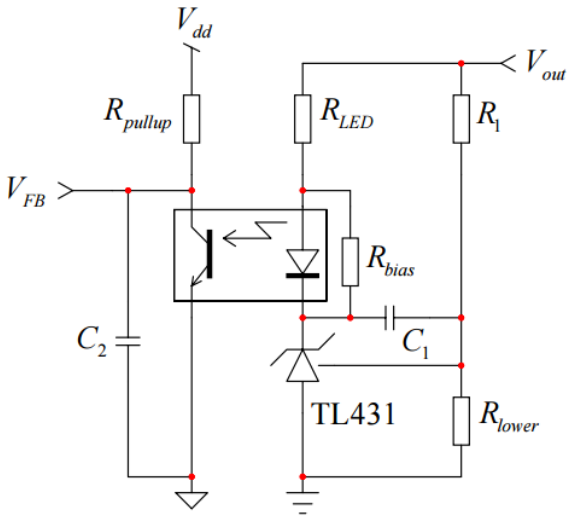


Figura 2.10: Típico circuito de realimentación con el TL431.

Por otro lado, cuando se utiliza regulación desde el primario no es necesario el agregado de un circuito de regulación. En este modo de regulación se aprovecha la señal de un bobinado auxiliar en fase con la salida pero en el lado primario. Dependiendo de las características del controlador, el bobinado auxiliar puede cumplir algunas de las funciones que se detallan a continuación:

- Alimentación: El uso más común es rectificar la forma de onda en este bobinado para entregar la tensión de alimentación al controlador.
- Sensado de desmagnetización: El controlador usa las transiciones de esta forma de onda para saber cuándo, si es que ocurre, termina el tiempo de desmagentización. Esta característica es útil para garantizar DCM o TM, si es que el controlador opera en alguno de estos modos, o para usar la técnica de valley switching (que posteriormente será explicada).
- Sensado de la tensión de entrada: Durante el tiempo de encendido de la llave, la tensión en este bobinado es proporcional, mediante la relación de vueltas entre bobinado auxiliar y primario, a la tensión de entrada.
- Sensado de la tensión de salida: Durante el tiempo de desmagnetización, la tensión en este bobinado es proporcional, mediante la relación de vueltas entre bobinado auxiliar y secundario, a la tensión de salida más la caída de tensión en el diodo de salida.

Esta última característica en la forma de onda del bobinado auxiliar, es la que permite cerrar el lazo de tensión, ahora con la referencia dentro del controlador, para regular la tensión de salida desde el primario de la fuente de alimentación.

En la figura 2.11 se puede ver esta forma de onda, a la izquierda en su versión ideal y a la derecha como es en realidad.

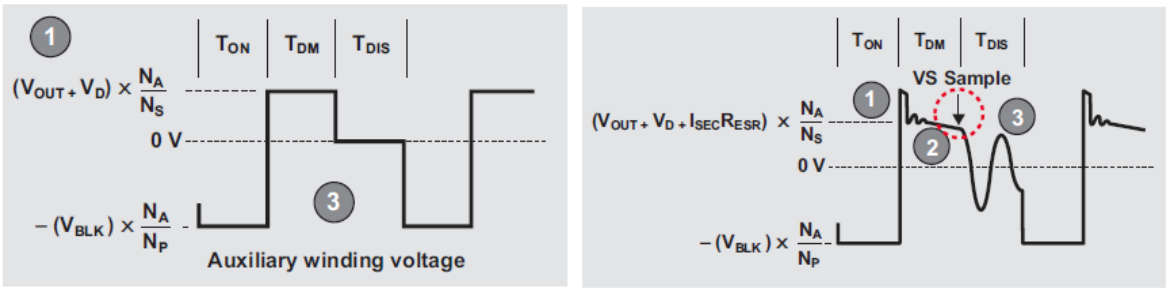


Figura 2.11: Forma de onda del bobinado auxiliar.

La inductancia de pérdida del bobinado que genera el sobrepico en el comienzo de T_{OFF}, las capacidades parásitas que junto con la inductancia de pérdida generan las oscilaciones de alta frecuencia y la ESR (que es producida por el bobinado secundario, el capacitor de salida y diodo de salida), que genera una pendiente durante el tiempo de desmagnetización, son la causas de deformación de la forma de onda. Se ve de forma clara la dificultad de poder tomar una muestra confiable de la tensión de salida. En la imagen de la derecha de la figura 2.11, se señala el punto ideal para muestrear V_0, ya que en este punto la corriente en el diodo de salida es cero y por lo tanto también lo es su caída de tensión, quedando solo la tensión de salida. Sin embargo esto es difícil de conseguir en la práctica e impone un límite a la tolerancia en la tensión de salida.

Ventajas de la regulación desde el primario:

- Reducción de la cantidad de componentes, que ayuda a reducir costo, tamaño y complejidad. La figura 2.12 muestra una comparación entre un típico circuito con regulación desde el primario y otro desde el secundario, donde claramente se ve la reducción de la cantidad de componentes.
- Compensación del lazo de control muy simplificada, ya que no es necesario el agregado de ninguna red de compensación.
- Se puede añadir también regulación de la corriente de salida.

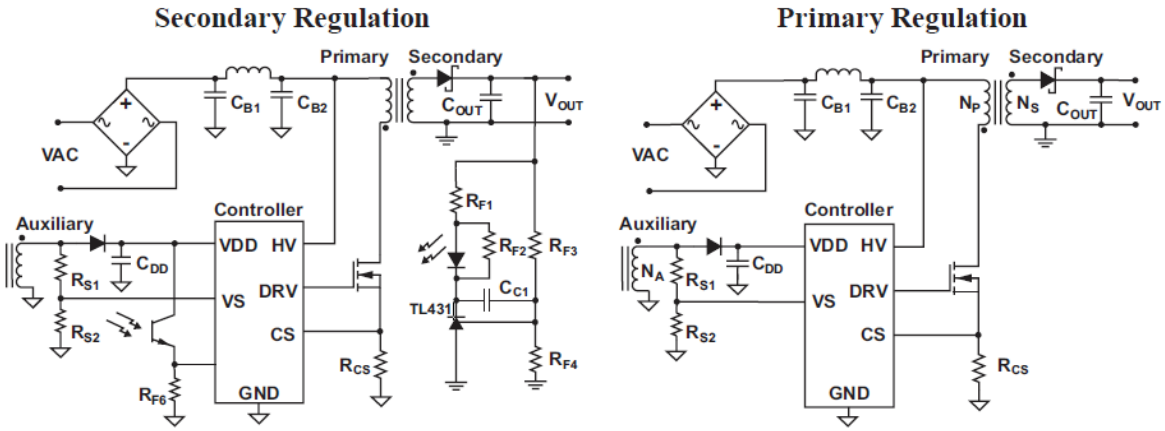


Figura 2.12: Comparación de circuitos para regulación desde el primario y secundario.

Desventajas de la regulación desde el primario:

- Tiene una tolerancia del 3% al 5%.
- La respuesta de los transitorios es dependiente de la frecuencia de operación, pasar de una condición de poca carga a una de carga completa puede causar una gran caída en la tensión de salida.

En conclusión, debido a la mejor regulación y mejor respuesta a los transitorios que ofrece la regulación desde el secundario, se optó por esta última para el controlador Flyback.

2.6 El controlador UCC28740

Teniendo en cuenta las secciones anteriores, se buscó un regulador conmutado para la topología Flyback, que trabaje en modo de conducción discontinua, utilice control en modo de corriente y regule desde el secundario. Se eligió el controlador UCC28740 de Texas Instruments, que cumple con todas las características requeridas. Además de las mencionadas anteriormente, este controlador presenta otras características que mejoran el rendimiento de la fuente, y que se analizarán a continuación.

Valley switching

Este controlador usa la técnica de valley switching para reducir las pérdidas por conmutación y facilitar el filtrado EMI. Para mostrar en que consiste y como logra reducir perdidas esta técnica, nos apoyaremos en la figura 2.13. En esta imagen se muestran las formas de onda idealizadas (sin las oscilaciones producto de las inductancias de pérdida del bobinado primario y las capacidades parásitas) en CCM y DCM de la tensión en la llave, V_{SWN}, y el circuito del convertidor Flyback con el agregado de la capacidad C_{SWN}, que representa la suma de las capacidades parásitas que ve la llave.

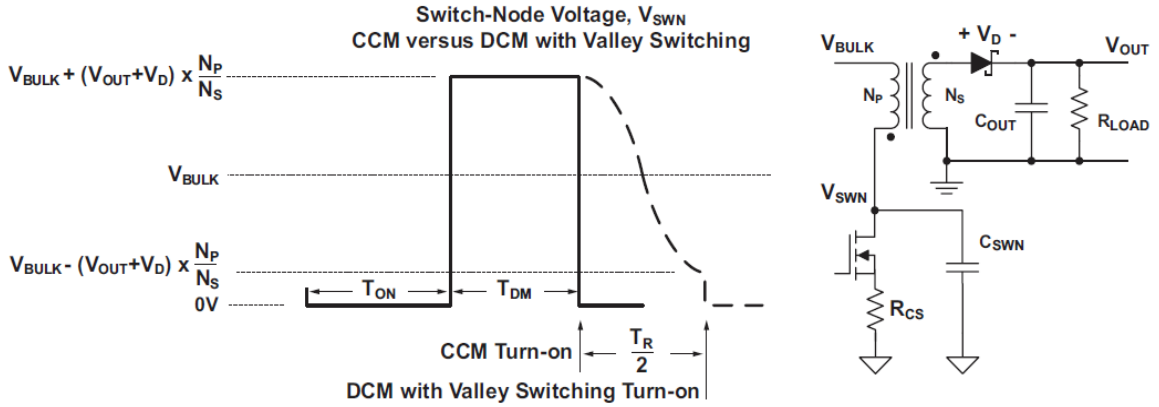


Figura 2.13: Valley switching en DCM vs CCM.

En el momento de encendido de la llave, una energía proporcional a la capacidad C_{SWN} y al cuadrado de la tensión en la llave es disipada. Esto nos dice, que si queremos disminuir estas pérdidas, la tensión en la llave en el momento de la conmutación debe ser lo menor posible. Esto es posible en DCM aprovechando las oscilaciones de alta frecuencia que existe en V_{SWN} una vez terminado T_{DM}.

En la figura 2.13 se aprecia la comparación entre CCM y DCM, donde se puede ver que al final del tiempo de desmagnetización en DCM y el momento de encendido de la llave en CCM, la tensión en la llave es como se expresa en la ecuación 2.7:

$$V_{SWN} = V_{BULK} + N_{PS} * (V_0 + V_D) \qquad (2.7)$$

Pero si en DCM, retrasamos el momento de encendido de la llave en un tiempo igual a la mitad del periodo de las oscilaciones, T_R, la tensión en la llave ahora será como se expresa en la ecuación 2.8:

$$V_{SWN} = V_{BULK} - N_{PS} * (V_0 + V_D) \qquad (2.8)$$

Se ve que la V_{SWN} es inferior en la ecuación 2.8 con respecto a la 2.7. En la ecuación 2.9 se expresa la máxima potencia de pérdida que puede ser ahorrada, simplemente restando la potencia de pérdida producida con la expresión 2.8 a la potencia de pérdida producida por la 2.7.

$$2 * C_{SWN} * V_{BULK} * \left(N_{PS} * (V_0 + V_D)\right) * f_{SW} \qquad (2.9)$$

Queda claro que esta técnica no puede ser aplicada a CCM, debido a que la tensión en la llave en el momento de encendido es siempre la misma. En cambio en DCM, aun sin aplicar esta técnica hay algo de potencia que se podría ahorrar, ya que el momento del encendido puede suceder en cualquier momento de la oscilación. Aunque la potencia que se disipa aun aplicando esta técnica depende del punto de operación de la SMPS. Observando las formas de onda de la figura 2.14, se puede ver que si el tiempo de conducción discontinua es muy grande, como es el caso de la forma de onda de la derecha, la potencia de las oscilaciones es disipada por los elementos que amortiguan el tanque LC y se pierde la posibilidad de ahorrar más potencia. Aunque aún en este caso, es mejor que CCM, ya que ahora la tensión en la llave será igual a V_{BULK}, valor que es menor a la expresada por la ecuación 2.8 en el caso de CCM.

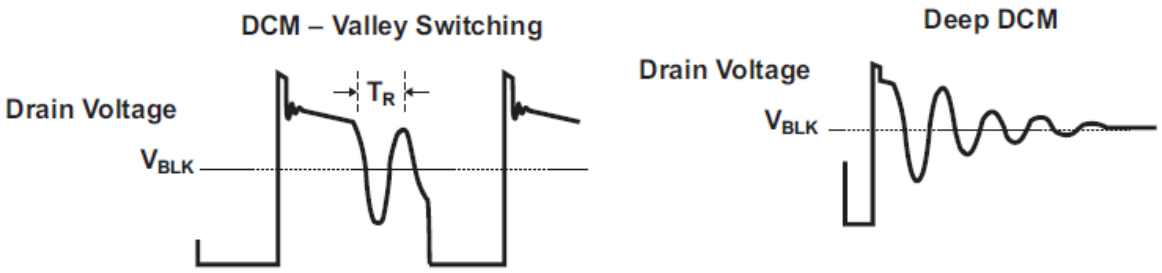


Figura 2.14: Valley switching para distintos T_{DIS}.

Modulación de frecuencia

Además de controlar la corriente pico del primario (modo de control al que llama modulación de amplitud), como se explicó en la sección 1.4, el UCC2870 agrega un control de frecuencia variable. Esta forma de control reduce las pérdidas por conmutación cuando se opera a potencias menores de la máxima y es especialmente importante para mantener bajo el consumo sin carga.

Podemos expresar la potencia de salida, P_0, en función de la potencia de entrada y de la eficiencia, η, como se expresa en la ecuación 2.10:

$$P_0 = P_{IN} * \eta \qquad (2.10)$$

Si además combinamos esta ecuación con la 2.7 obtenemos la 2.11:

$$P_0 = P_{IN} * \eta = E * f_S * \eta = \frac{1}{2} * L_P * {I_{PP}}^2 * f_s * \eta \qquad (2.11)$$

Esta ecuación deja en claro que hay dos formas de controlar la potencia de salida, una es modificando la energía entregada por ciclo mediante la corriente pico del primario (Modulación de amplitud) y la otra es mediante la frecuencia de conmutación (Modulación de frecuencia). Este controlador usa ambos tipos de control dependiendo la potencia que se está entregando a la carga. El gráfico de la figura 2.15 muestra como la corriente pico del primario y la frecuencia de conmutación van cambiando de acuerdo a la corriente de realimentación, I_{FB}, que es la que impone cuanta potencia se le entrega a la carga.

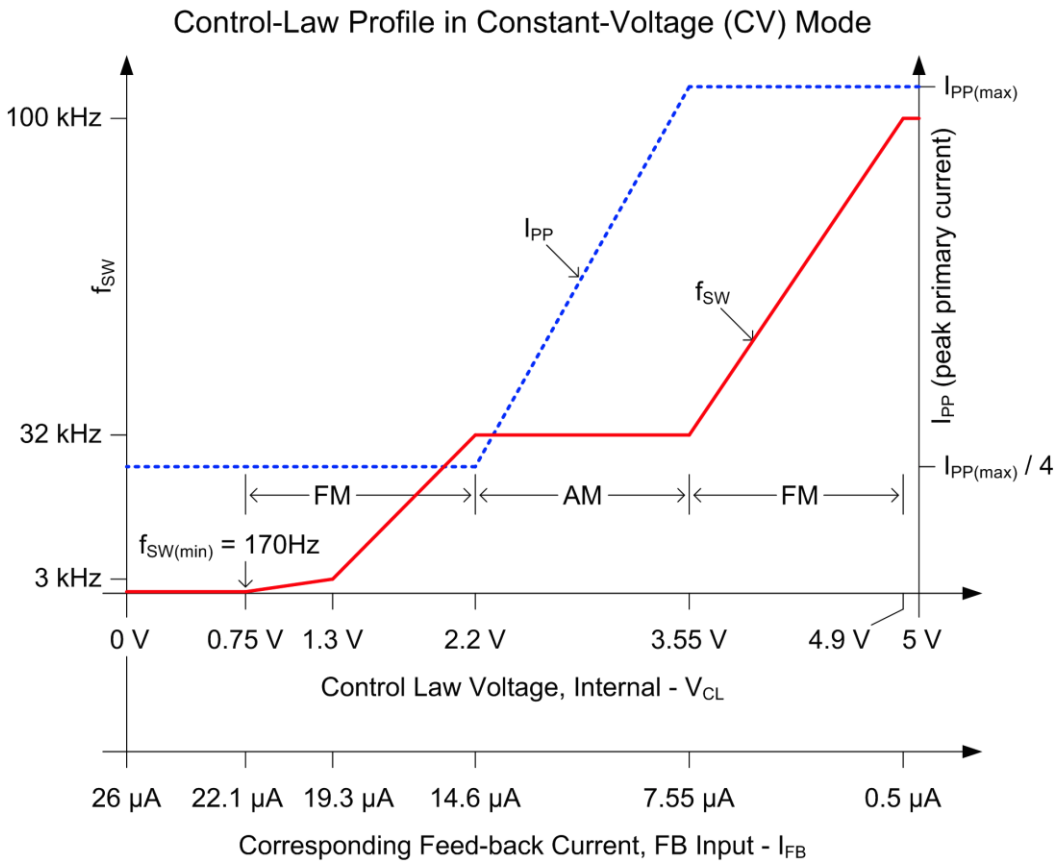


Figura 2.15: Ley de control del UCC28740.

En el grafico se pueden distinguir tres zonas, una de AM y dos de FM. En la zona de más a la derecha se usa FM y termina con corriente pico del primario y frecuencia de conmutación (100KHZ) máximas, por ende esta es la zona de máxima potencia de la fuente. A medida que nos movemos hacia la izquierda la frecuencia de conmutación va disminuyendo, y por ende también la potencia entregada a la salida, mientras que I_{PP} se mantiene fija, al máximo. Esto se mantiene así hasta que se llega a la zona de AM, punto en el que f_{SW} queda fija en 32KHZ y la corriente es ahora la que empieza a disminuir si se necesita entregar menos potencia. Nuevamente se mantiene esta tendencia hasta llegar a la otra zona de FM, donde I_{PP} queda fija en la máxima sobre cuatro y es nuevamente variable la frecuencia. Esta zona tiene dos pendientes diferentes que se unen en los 3KHZ, y tiene como frecuencia mínima los 170HZ. Esta última zona es la que permite potencias sin carga del orden de los mW.

También cabe resaltar que a medida que aumenta la potencia demandada, la corriente de realimentación disminuye (realimentación negativa).

Control de corriente

Cuando la corriente promedio en la carga supera el límite establecido por el diseño, el controlador cambia de modo de tensión constante (CV) a modo de corriente constante (CC). En modo CC, se pierde la regulación de la tensión de salida y el lazo impone I_{FB} al mínimo (es decir máxima potencia). Con la información de los tiempos del bobinado auxiliar y la corriente en la llave, se obtiene regulación de la corriente promedio en el bobinado secundario. En la sección 1.5 ya se explicó cómo se obtiene información de la salida con ayuda del bobinado auxiliar. En la figura 2.16 se puede observar la forma de onda en el bobinado secundario.

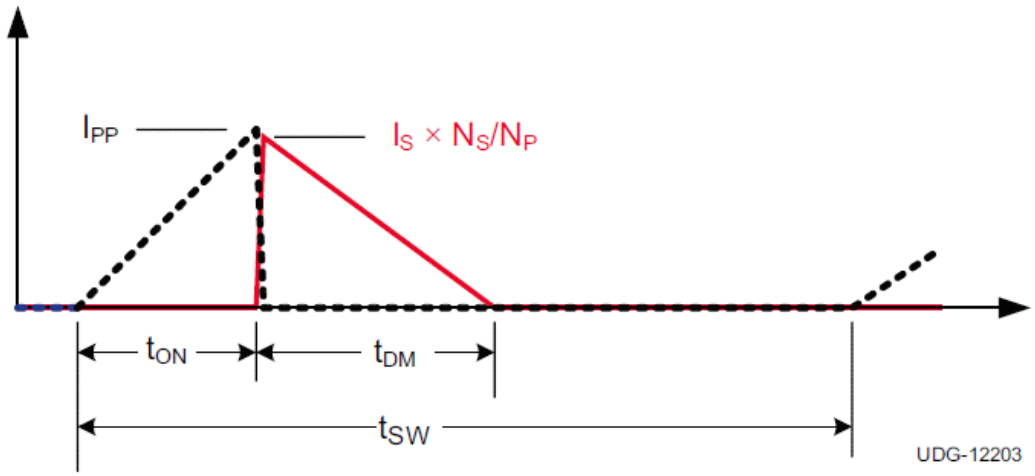


Figura 2.16: Forma de onda idealizada en el bobinado secundario.

Si se hace un promedio de la señal roja, la corriente en el secundario, se obtiene la expresión de la corriente promedio en la salida, como lo indica la ecuación 2.12:

$$I_O = \frac{I_{PP}}{2} * N_{PS} * \frac{T_{DM}}{T_{SW}} \qquad (2.12)$$

Cuando el ciclo de trabajo de desmagnetización llega a su valor máximo, D_{MAGCC}, el controlador empieza a actuar en FM y controla la corriente promedio de salida para cualquier tensión de salida, igual o por debajo a la de regulación. Se mantiene la regulación de la corriente mientras que la tensión de salida sea lo suficientemente alta como para producir una tensión de alimentación en el auxiliar que le permita al controlador seguir funcionando. Cuando la tensión de salida disminuye, T_{DM} aumenta. El controlador actúa aumentando T_{SW} de manera de mantener el ciclo de trabajo de desmagentización constante y de esta forma mantener la corriente promedio en la salida.

En conclusión el UCC28740 controla la tensión de salida hasta que la corriente alcanza un límite, establecido por diseño, y comienza a mantener la corriente constante. La figura 2.17 muestra la relación entre tensión y corriente de salida resultante.

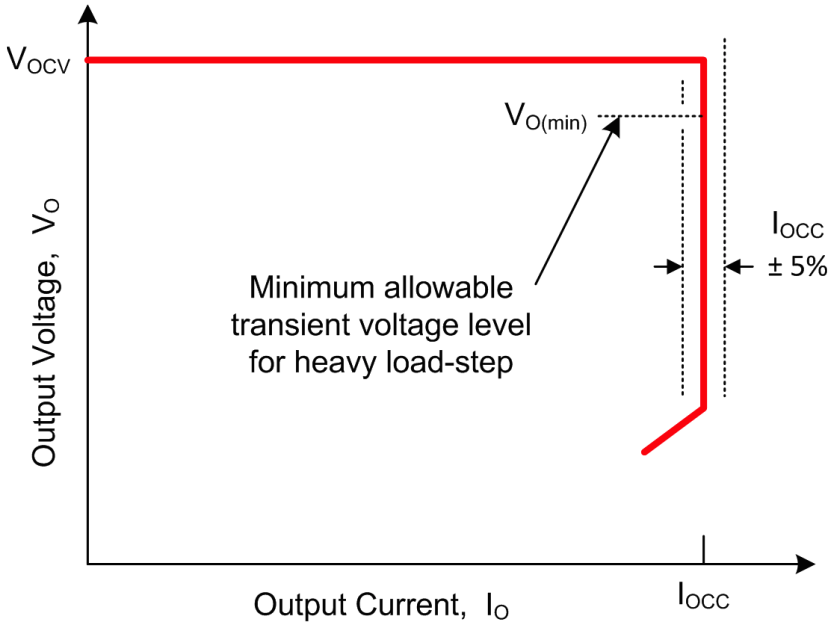


Figura 2.17: Tensión de salida vs Corriente de salida

Operación en el arranque

Una llave de arranque de alta tensión, conectada a la tensión de entrada a través del pin HV del UCC28740, carga el capacitor de alimentación. Esta llave de arranque funciona como una fuente de corriente que carga el capacitor de alimentación con una corriente de aproximadamente 250µA. Una vez que la tensión de alimentación alcanza el límite de 21V la llave se apaga y el controlador esta encendido, se comienza a conmutar la llave.

Generalmente en el encendido inicial, el capacitor de salida se encuentra descargado. El controlador arranca con tres pulsos con I_{PP} al mínimo y verifica cualquier falla en el circuito de entrada y salida. Luego de estos tres pulsos, si la tensión muestreada en el auxiliar no supera el valor de 1.33V, el controlador entra en un modo especial de operación. En este modo se ajusta I_{PP} al 63%

de su valor máximo y se aumenta D_{MAGCC} de 0.425 a 0.735, estas modificaciones permiten una carga de alta frecuencia del capacitor de salida para evitar el ruido audible mientras la tensión de desmagnetización es aún baja. Una vez superados los 1.38V en la tensión del auxiliar, I_{PP} se ajusta al máximo y D_{MAGCC} vuelve a su valor nominal. Mientras el capacitor de salida se carga, el convertidor opera en CC para mantener la corriente de salida constante hasta que se comience la regulación de la tensión de salida. Finalmente, el controlador empieza a actuar por lo que impone la corriente de realimentación. El tiempo que tarda el controlador en empezar a regular es la suma del tiempo que tarda en cargarse el capacitor de alimentación y luego el capacitor de salida.

Protección contra fallas

El UCC28740 provee protección contra diversas fallas que pueden ocurrir durante la operación del mismo:

- Sobretensión en la salida: Mediante la tensión muestreada en el auxiliar, el controlador conoce el valor de la tensión de salida y actúa en caso de que esta tensión supere el límite establecido por diseño. Una vez detectada esta condición, el UCC28740 deja que el capacitor de alimentación se descargue hasta que el valor de esta tensión quede por debajo del nivel de 7.75V establecido por el controlador. Luego de esto se vuelve a comenzar la secuencia de inicio explicada anteriormente.
- Baja tensión en la entrada: Nuevamente usando la tensión en el auxiliar, el UCC28740 conoce el nivel de tensión en la entrada y verifica que este dentro de los limites inferior y superior impuestos para esta tensión en el diseño. Al igual que en el caso de la falla anterior, el controlador se reinicia siguiendo la misma secuencia.
- Exceso de temperatura: El UCC28740 monitorea la temperatura de juntura del encapsulado y no lo permite superar los 165°C. El controlador se resetea de forma continua mientras esta situación persista.
- Sobrecorriente en el primario: El pin CS, encargado de sensar la corriente del primario, además ser usado para el control en modo corriente, se usa como protección.
- Protección interna para los pines VS y CS.

Una vez explicadas las características del controlador, se presenta el diagrama en bloques del mismo en la figura 2.18.

Como se explicó anteriormente, el pin HV del UCC28740 va conectado a la tensión de entrada, V_{BULK}, para alimentar la fuente de corriente IHV que carga el capacitor de alimentación en el momento de encendido.

El pin VDD se conecta a la tensión de alimentación, V_{DD} es generada por la rectificación de la forma de onda en el bobinado auxiliar. Este pin también está conectado a un bloque que detecta los 21V a los que se debe cargar el capacitor de alimentación al momento de encendido. Una vez que el controlador está habilitado verifica que V_{DD} nunca caiga por debajo de 7.75V, tensión de alimentación mínima requerida por el controlador.

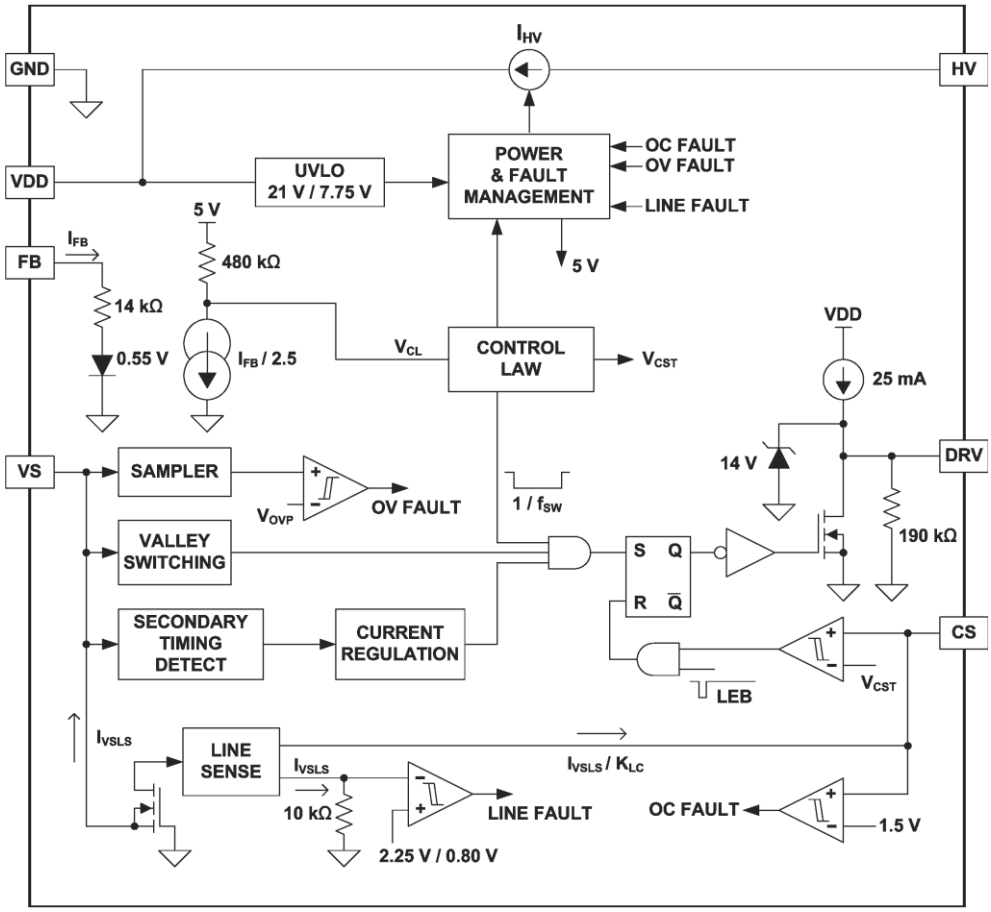


Figura 2.18: Diagrama en bloques del controlador UCC28740.

El diagrama muestra como la corriente de realimentación circula por una resistencia interna de 14KΩ y un diodo. I_{FB} es dividida por 2.5 y luego mediante una resistencia de pull-up de 480KΩ conectada a 5V, genera la tensión de control, V_{CL}. Esta tensión ingresa en el bloque que impone la frecuencia de conmutación y la tensión V_{CST}, que es la determina la corriente pico del primario.

El pin VS va conectado a un divisor de tensión en el bobinado del auxiliar. Esta tensión ingresa a un bloque que muestrea la tensión de salida, que luego es comparada con 4.6V para detectar una sobretensión en la salida. Por otro lado, también desde esta tensión se detectan los valles de las oscilaciones para utilizar la técnica de Valley Switching, explicada anteriormente, que retrasa el inicio de un nuevo ciclo para reducir la potencia disipada en la llave. Además, también mide el tiempo de desmagnetización, que permite la regulación de la corriente promedio de salida. Por último, se genera la corriente I_{VLS} que permite medir V_{BULK} para verificar condiciones de falla y para compensar el nivel en el pin CS para todo el rango permitido en V_{BULK}.

Mediante una resistencia que sensa la corriente en la llave, se genera la tensión en el pin CS. Esta tensión es comparada con V_{CST}, cuyo nivel máximo es 0.773V correspondiente a I_{PP} máxima y el nivel mínimo es de 0.194V que corresponde con I_{PP} máxima dividida cuatro. A la salida de este comparador aparece una compuerta AND, con un pulso de blanking de 235nS para mitigar el problema de los picos de corriente que aparecen por el encendido de la llave. En serie a este pin de

coloca una resistencia, que provee compensación de línea feed-forward para eliminar las variaciones en I_{PP} con la tensión de entrada, debido al delay del comparador interno y el tiempo de apagado del MOSFET.

Por último, el pin DRV va conectado al gate de un MOSFET. Luego de la salida del Flip-Flop, donde se genera la forma de onda cuadrada que el lazo impone para mantener la regulación en la tensión de salida, sigue un circuito que maneja el gate del MOSFET. La señal de este circuito de drive está limitada internamente a 14V mediante un diodo zener. Para el encendido se genera una corriente de 25mA, que limita la dv/dt de encendido del drain del MOSFET y reduce los picos de corriente. A la vez provee una corriente de drive que permite superar la tensión de Miller Plateu. La corriente de apagado está dada por la resistencia de encendido del MOSFET interno, que aparece en el circuito de drive observado en la figura 2.18.

2.7 Diseño de la fuente de alimentación

Con el controlador ya elegido y analizado, se procedió con el diseño de la SMPS. El esquema básico que se siguió, es uno típico de una fuente con topología Flyback con realimentación con opto-acoplador usando el UCC28740, que es mostrado en la figura 2.19.

Potencia de entrada y salida

El primer paso es definir la potencia y las tensiones de salida necesarias de la SMPS. La salida principal, por ser la de mayor consumo, es de 5V y tiene los siguientes circuitos como cargas:

- Pantalla táctil VM800B, cuyo consumo máximo según el fabricante es de 1A. Aunque el consumo depende de la aplicación y en nuestro caso nunca paso de los 0.5A.
- Placa de desarrollo NUCLEO-F410RB, con una corriente limitada por un regulador de tensión interno en la placa, máxima de 0.5A. Sin embargo, los consumos más grandes se dan durante la programación y depuración, durante la operación normal el consumo será mucho menor.
- Opto-acoplador ranurado HOA7720, que tiene un consumo máximo de 40mA.
- Un regulador shunt utilizado para generar una referencia de 3.3V, con un consumo de 6mA por diseño.
- Tarjeta SD, cuyo máximo consumo se da durante la escritura y depende de la tarjeta utilizada, pudiendo llegar a valores no mayores a 200mA.

Teniendo esto en cuenta, se adoptó un consumo de 1.5A para esta salida. Por otro lado, hay dos salidas de ±15V (aproximadamente) que presentan las siguientes cargas:

- Un amplificador con dos operacionales, un limitador con otro operacional y un comparador. Estos circuitos presentan un consumo muy bajo, del orden de los mA por circuito.
- Un regulador shunt que provee una tensión de 10V muy precisos, al que se diseñó con un consumo de aproximadamente 35mA.

- Un sensor de proximidad inductivo, cuyo máximo consumo es de 10mA.

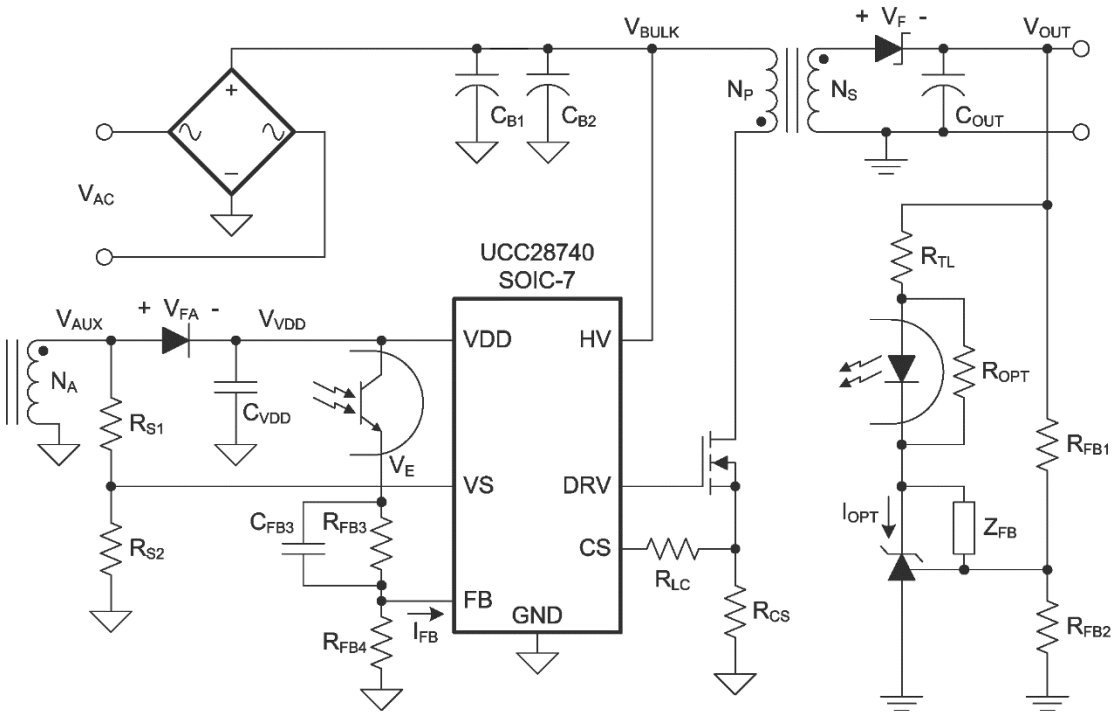


Figura 2.19: Circuito básico de la fuente de alimentación.

Considerando esto, un consumo de corriente de 0.25A se adoptó para las salidas de ±15V. Con las corrientes y tensiones definidas se calcula la potencia de salida requerida mediante la ecuación 2.13:

$$P_0 = 5V * 1.5A + 15V * 0.25A = 11.25W \qquad (2.13)$$

Se adopta P_0 = 11W. Ahora mediante la ecuación 2.10 y adoptando una eficiencia del 85% se obtiene:

$$P_{IN} = \frac{P_0}{\eta} = \frac{11W}{0.85} \approx 13W$$

Tensión de entrada

El siguiente paso fue determinar la mínima tensión de entrada, $V_{BULK(MIN)}$. La tensión de entrada varía debido a las variaciones en la tensión de línea y al ripple producto de la rectificación y filtrado. La fuente de alimentación tiene que ser capaz de entregar la máxima potencia con la mínima tensión de entrada. Por lo tanto es necesario partir de este valor para el diseño, ya que es el peor caso desde el punto de vista de la potencia.

La SMPS fue diseñada para permitir que la tensión de línea de 220V RMS tenga una variación del ±15%, resultando que 187V RMS < V_{LINEA} < 253V RMS. Se tomó como criterio que $V_{BULK(MIN)}$ sea aproximadamente un 20% menor a la tensión de línea pico mínima, es decir:

$$V_{BULK(MIN)} = 0.8 * 187VRMS * \sqrt{2} \approx 211V$$

Para obtener este valor de tensión hay que seleccionar el capacitor de entrada, C_{BULK}, adecuado. En una figura 2.20 se puede observar en azul a la forma de onda de la tensión de línea mínima, que luego de ser filtrada y rectificada resulta la forma de onda verde, que es V_{BULK}. Por lo tanto hay que colocar un capacitor que se descargue de la tensión de pico línea mínima, 264.45V, a 211V en medio periodo de línea y con una carga que represente a P_{IN}. La ecuación 2.14 presenta una solución precisa para esta situación:

$$C_{BULK(MIN)} = \frac{2*P_{IN}*(0.25+\frac{1}{2\pi}*\arcsin(\frac{V_{BULK(MIN)}}{\sqrt{2}*V_{LINEA(MIN)}}))}{(2*{V_{LINEA(MIN)}}^2-{V_{BULK(MIN)}}^2)*f_{LINEA}} \approx 9\mu F \qquad (2.14)$$

Se diseñó para aceptar una variación en la frecuencia de línea del 10%, por eso es que para el cálculo se utilizó 45HZ, ya que es el peor caso para la descarga del capacitor. Finalmente se adoptó una capacidad de 10µF y para el resto del diseño de uso 210V como tensión de entrada mínima, para dejar un margen de seguridad.

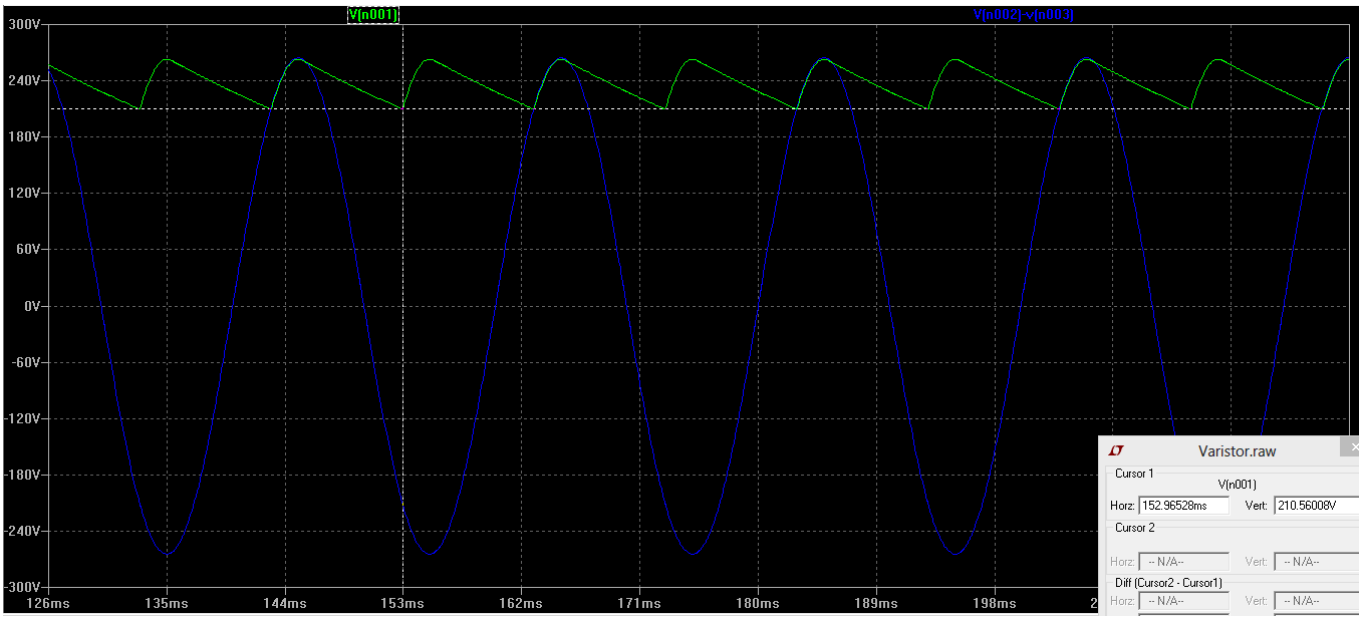


Figura 2.20: Formas de onda para la tensión de línea y la tensión de entrada al convertidor.

Ciclo de trabajo

Luego se definió el tiempo de encendido de la llave máximo, situación que se dará con la SMPS trabajando con carga completa. El controlador ya tiene fijo el máximo ciclo de desmagnetización en 0.425, con lo cual solo resta asignar cuanto será T_{DIS}. Teniendo en cuenta que el controlador trabaja en DCM con valley-switching, se adoptó dejar un margen del 10% para el

tiempo de conducción discontinua, tiempo que le da oportunidad al controlador de detectar oscilaciones de al menos 500KHZ. Esto resulta en un ciclo de trabajo máximo de 0.475, $D_{ON(MAX)}$ = 0.475.

Relación de vueltas entre primario y secundario

A continuación se diseñó la relación de vueltas entre primario y secundario, un parámetro muy importante, ya que condiciona el punto de operación, la eficiencia, el stress en las llaves y el diseño del transformador. Para mejorar la eficiencia, se llevó al convertidor cerca de TM mediante la ecuación 2.15:

$$N_{PS(MAX)} = \frac{D_{ON(MAX)} * V_{BULK(MIN)}}{D_{MAGCC} + (V_0 + V_F)} = \frac{0.475 * 210V}{0.425 * (5V + 0.6V)} \approx 42 \qquad (2.15)$$

Si se utiliza un valor mayor a este, el convertidor entrara en TM para una tensión de entrada superior a la mínima. En cambio, con un valor muy por debajo de este, el convertidor trabajará en DCM con un T_{DIS} muy grande, provocando grandes corrientes RMS que generan muchas pérdidas. Por ende la mejor opción es adoptar un valor ligeramente inferior a este, para que el convertidor trabaje cerca de TM mejorando la eficiencia, pero dejando un margen para DCM con valley switching. En conclusión, se adoptó un valor aproximadamente 10% menor al indicado en la ecuación 2.16, resultando N_{PS} = 38.

Corriente pico del primario e inductancia del primario

Para determinar tanto la corriente pico del primario como la inductancia del primario, se recurre a las ecuaciones 2.1 y 2.6. Se plantea el peor caso, máxima potencia con $V_{BULK(MIN)}$. Resulta un sistema de dos ecuaciones con dos incógnitas:

$$P_{IN} = \frac{1}{2} * L_P * {I_{PP(MAX)}}^2 * f_{S(MAX)} \;\rightarrow\; 13W = \frac{1}{2} * L_P * {I_{PP(MAX)}}^2 * 100KHZ$$

$$I_{PP(MAX)} = \frac{D_{ON(MAX)} * T_{S(MIN)} * V_{BULK(MIN)}}{L_P} \;\rightarrow\; I_{PP(MAX)} = \frac{0.475 * 10\mu S * 210V}{L_P}$$

Resolviendo el sistema se obtiene, L_P = 3.8269mH e $I_{PP(MAX)}$ = 0.26A. Con el nivel máximo para la tensión en el pin CS, 0.773V, se obtiene una resistencia de sensado, R_{CS}, de aproximadamente 2.97Ω. Pero se toma un valor de 3Ω para esta resistencia, ya que con este valor comercial se tiene más opciones para esta resistencia, que debe tener características especiales debido a su función. Con este valor adoptado se recalculan $I_{PP(MAX)}$ y L_P, resultando $I_{PP(MAX)}$ = 0.2576A y L_P = 3.92mH. Para recalcular la inductancia se partió de la ecuación que expresa la potencia, de forma tal que la potencia no se modifique. Como consecuencia ahora se ve modificado el ciclo de trabajo, resultando $D_{ON(MAX)}$ = 0.48097. Esta pequeña variación no cambia en nada el diseño, solo se pasa del 10% al 9% en D_{DIS}.

Circuito de clamp

Para proteger la llave se utiliza un circuito que límita la excursión de la tensión en el drain del MOSFET, a continuación se detallara el porqué de la necesidad de añadir este circuito y el diseño del mismo.

Cuando la llave se cierra circula corriente tanto por la inductancia primaria como en la inductancia de pérdida, como se muestra en la figura 2.21 en el circuito de la izquierda. En este circuito no aparece el secundario por que no circula corriente por el diodo secundario. Luego de T_{ON}, la llave se abre y la corriente en L_P y L_{LEAK} llega a I_{PP}, quedando ahora un circuito equivalente como que el que se muestra en el circuito de la derecha de la figura 2.21.

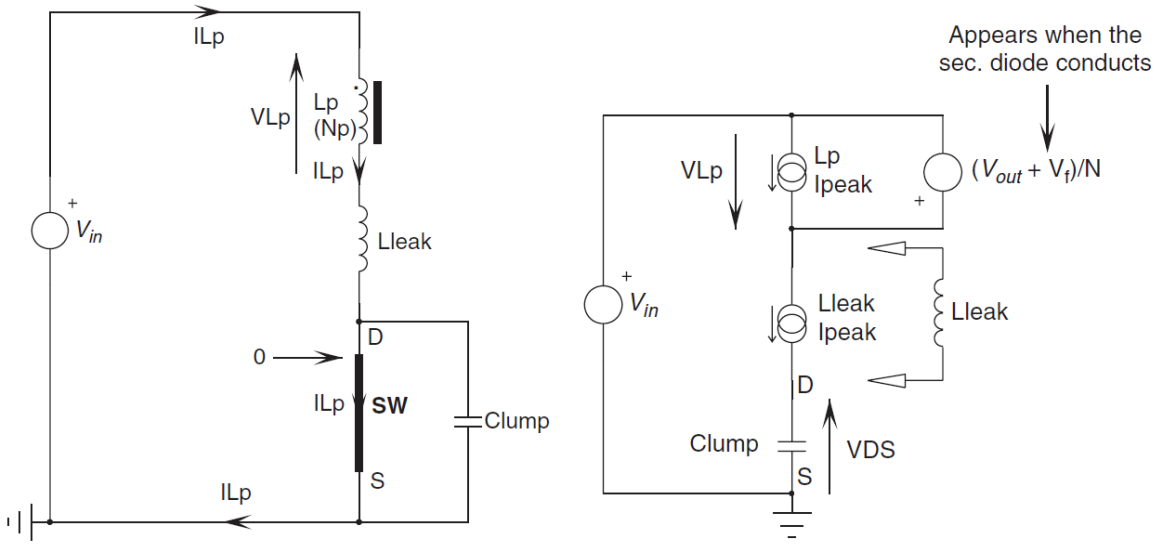


Figura 2.21: Circuitos equivalentes en el primario con la llave cerrada y abierta.

En el circuito equivalente con la llave abierta y el diodo de salida ya conduciendo, se modela a la inductancia del primario como un generador de corriente de valor I_{PP} y con un generador de tensión de valor igual a la tensión reflejada desde el secundario. Mientras que L_{LEAK} se modela como un generador de corriente de valor I_{PP}, y también se agrega al modelo la capacidad parásita que se ve en el drain del MOSFET, C_P.

Se produce un pico de tensión producto de la corriente de L_{LEAK} en la capacidad parásita. La impedancia característica del tanque LC, formada por L_{LEAK} y C_P, es la que fundamentalmente define el valor de este pico de tensión. La ecuación 2.16 determina el valor pico máximo de la tensión drain-source del MOSFET, $V_{DS(MAX)}$, que se da en esta situación:

$$V_{DS(MAX)} = V_{BULK(MAX)} + (V_0 + V_F) * N_{PS} + \sqrt{\frac{L_{LEAK}}{C_P}} * I_{PP} \qquad (2.16)$$

Se debe tener el cuidado de que este pico de tensión no produzca la ruptura por avalancha del MOSFET. Esta tensión suele del orden de los KV, y es necesario diseñar una red para prevenir la destrucción de la llave.

La solución más simple es colocar un capacitor de mayor valor al parásito. Esto tiene como ventaja que se achica la pendiente del pico de tensión y al mismo tiempo lo limita. Sin embargo, si el valley switching no reduce lo suficiente la tensión en el momento de cerrar la llave nuevamente, se producen grandes pérdidas producto de la potencia disipada en esta capacidad.

Una mejor solución es limitar VDS mediante un generador de tensión de baja impedancia y un diodo rápido. El generador de tensión de baja impedancia puede ser mediante un circuito RC o un TVS (Transient Voltage Suppressor), luego se analizarán ambas opciones pero por ahora se supondrá un generador de tensión ideal.

En la figura 2.22 se muestra el circuito de clamp cuando la llave se abre, pero ahora dejamos de suponer que el diodo del secundario empieza a conducir inmediatamente.

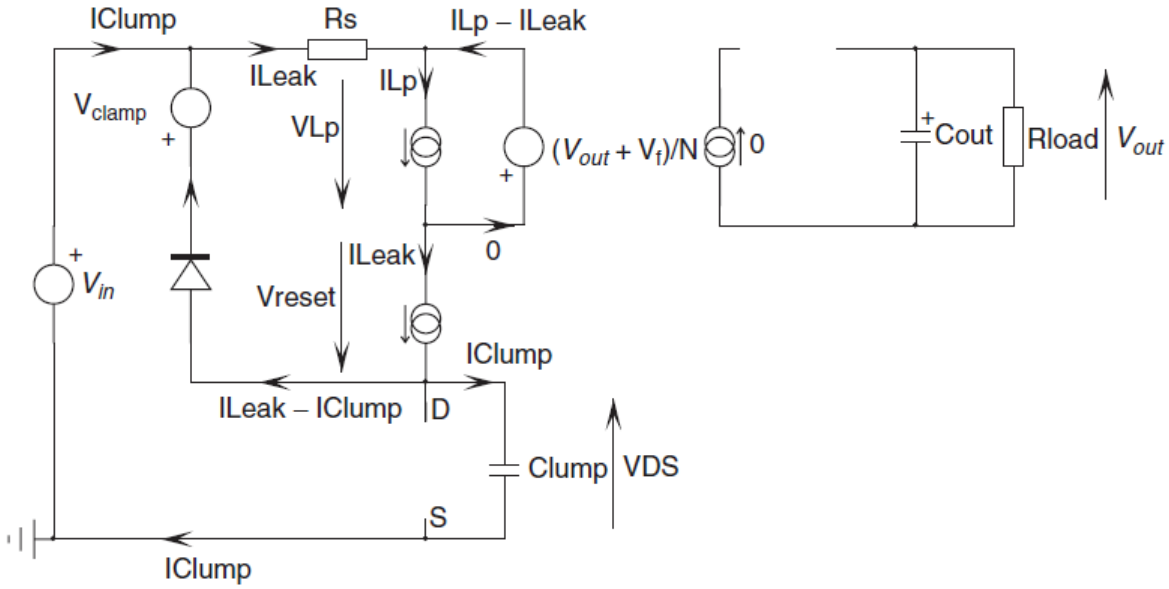


Figura 2.22: Circuito de clamp con la llave recién abierta.

Cuando la llave se abre, la tensión empieza a aumentar de forma abrupta, con una velocidad solamente limitada por C_P. Cuando V_{DS} llega a un valor igual a la suma de V_{BULK} y la tensión de clamp, V_{CLAMP}, el diodo serie del circuito de clamp empieza a conducir y limita rápidamente la excursión en la tensión de drain. De esta manera, eligiendo convenientemente el nivel de V_{CLAMP} se limita V_{DS} a un valor seguro.

Mientras que en el secundario la tensión en el ánodo del diodo de salida pasa de negativa, la tensión de entrada inversa, a positiva. Cuando la tensión de drain supera la tensión de entrada más la reflejada desde el secundario, el diodo queda polarizado en directa. Sin embargo, como se muestra en la figura 2.22, L_{LEAK} toma corriente de L_P. Aunque el diodo de salida este polarizado en directa, no puede alcanzar el valor $I_{PP}*N_{PS}$ instantáneamente, ya que inicialmente el inductor de pérdida se lleva toda la corriente. En consecuencia, el crecimiento de la corriente en el secundario queda limitado por L_{LEAK} y la tensión en la misma.

En la figura 2.23, ahora se muestra el circuito ya con el diodo de clamp limitando la excursión de V_{DS}, el de salida en conducción y la tensión de L_{LEAK}.

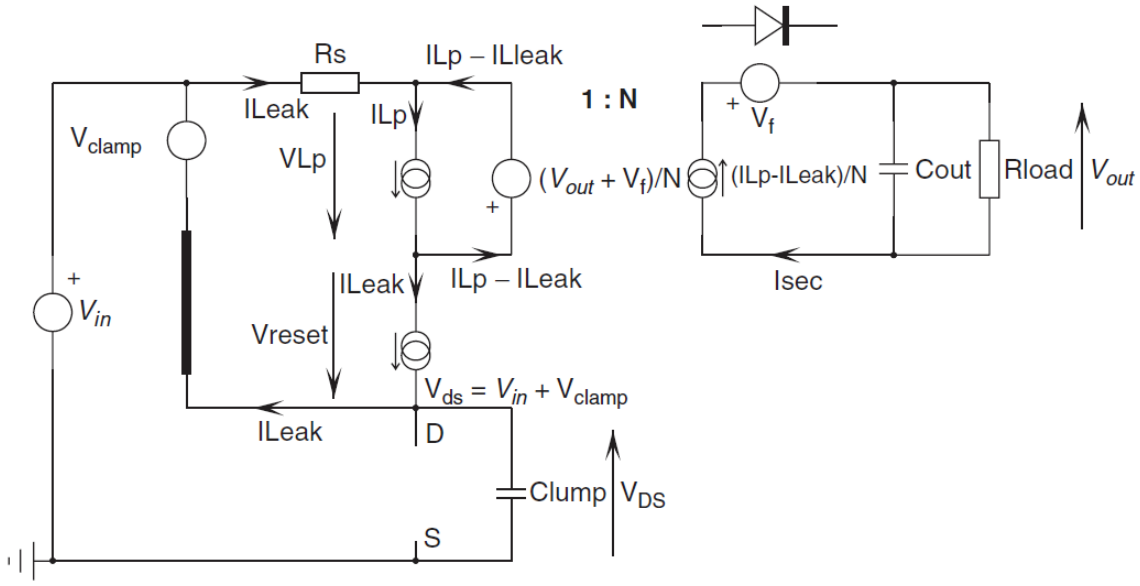


Figura 2.23: Circuito de clamp con V_{DS} limitada y el diodo secundario en conducción.

Como se puede ver en el circuito, existe ahora una tensión de reset, V_{RESET}, en el inductor de pérdida de valor igual al expresado en la ecuación 2.17:

$$V_R = V_{CLAMP} - (V_0 + V_F) * N_{PS} \qquad (2.17)$$

Además, la corriente en L_{LEAK} empieza a caer desde su valor pico con una pendiente dada por la ecuación 2.18:

$$S_{L_{LEAK}} = \frac{V_{RESET}}{L_{LEAK}} = \frac{V_{CLAMP} - (V_0 + V_F) * N_{PS}}{L_{LEAK}} \qquad (2.18)$$

Ahora la corriente en el secundario deja de ser cero y sigue el comportamiento expresado por la ecuación 2.19:

$$I_{SEC} = \left(I_{L_P} - I_{L_{LEAK}}\right) * N_{PS} \qquad (2.19)$$

Cuando la corriente en la inductancia de pérdida llega a cero, el diodo de clamp deja de conducir y el diodo de salida llega a su máximo. Pero llega a un valor menor al teórico, debido a que mientras se desenergiza la inductancia de pérdida, la misma toma corriente de L_P. Todas estas corrientes se pueden observar en la figura 2.24.

Para calcular el valor al que llega la corriente pico en el primario que es transferido al secundario, $I_{PP'}$, se utiliza la ecuación 2.20, donde S_P es la pendiente con la que vuelve a cero la corriente en el primario:

$$I_{PP}' = I_{PP} - S_P * \Delta t = I_{PP} - \frac{(V_0 + V_F) * N_{PS}}{L_P} * \Delta t \qquad (2.20)$$

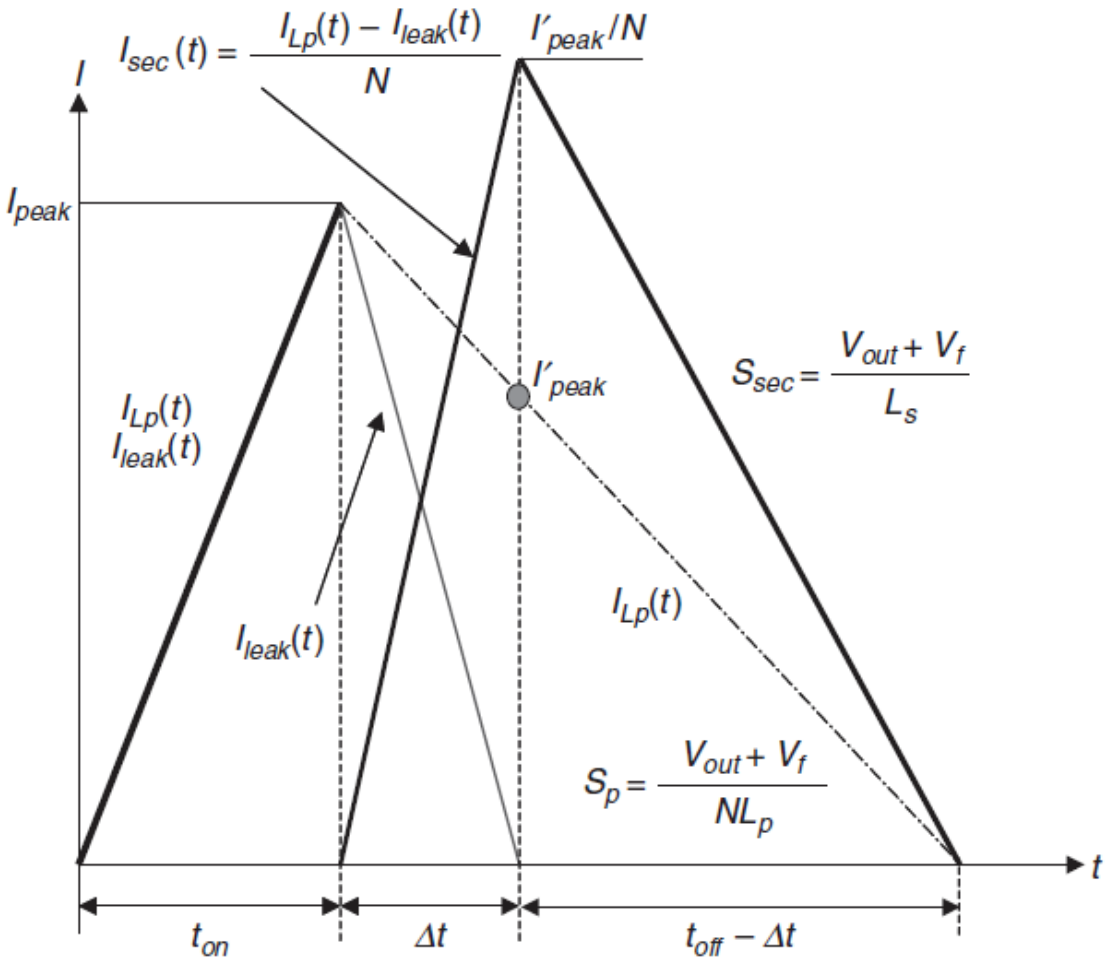


Figura 2.24: Corrientes del primario, de pérdida y en el secundario.

Δt es el tiempo necesario para descargar la inductancia de pérdida y está dado por la ecuación 2.21:

$$\Delta t = \frac{I_{PP}}{S_{L_{LEAK}}} = I_{PP} * \frac{L_{LEAK}}{V_{RESET}} = \frac{I_{PP} * L_{LEAK}}{(V_{CLAMP} - (V_0 + V_F) * N_{PS})} \qquad (2.21)$$

Introduciendo la ecuación 2.19 en la 2.18 y despejando la relación entre la corriente pico real e ideal, se obtiene la 2.22:

$$\frac{I_{PP}'}{I_{PP}} = \left[1 - \frac{L_{LEAK}}{L_P} * \frac{1}{\frac{V_{CLAMP}}{(V_0 + V_F) * N_{PS}} - 1}\right] \qquad (2.22)$$

Esta ecuación muestra claramente cuáles son las dos formas de acercar a uno este cociente. Una forma es reduciendo la inductancia de pérdida en comparación a la del primario, por lo tanto se requiere un buen diseño del transformador. La otra es utilizar una tensión de clamp mayor a la reflejada desde el secundario al primario. De esta forma, adoptando un MOSFET con una V_{DS} mayor, se consigue desenergizar L_{LEAK} más rápido y además reducir el stress en el circuito de clamp, ambos resultados ayudan a mejorar la eficiencia de la SMPS.

La corriente pico en el secundario es expresada por la ecuación 2.23:

$$I_{SP} = I'_{PP} * N_{PS} = I_{PP}\left[1 - \frac{L_{LEAK}}{L_P} * \frac{1}{\frac{V_{CLAMP}}{(V_0+V_F)*N_{PS}}-1}\right] * N_{PS} \qquad (2.23)$$

Una vez que el diodo de clamp deja de conducir, la inductancia de pérdida está completamente descargada, resultando ahora el circuito que se ve en la figura 2.25.

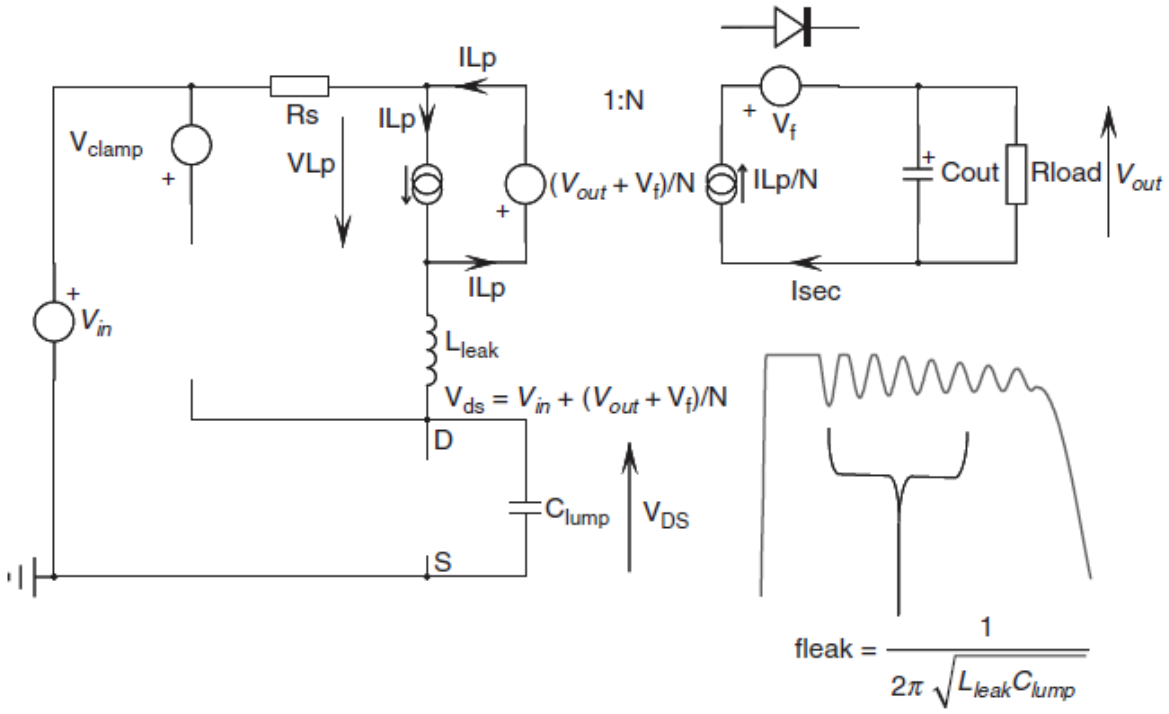


Figura 2.25: Circuito con L_{LEAK} descargada.

Cuando el diodo deja de conducir, V_{DS} debería caer a la suma de la tensión de entrada y la reflejada desde el secundario. Sin embargo, C_P impide que instantáneamente vaya a ese nivel. Por lo tanto, C_P y L_{LEAK} empiezan a intercambiar energía de forma oscilatoria a una frecuencia de oscilación dada por la ecuación 2.24:

$$f_{OSC} = \frac{1}{2\pi*\sqrt{L_{LEAK}*C_P}} \qquad (2.24)$$

Debido a las pérdidas óhmicas se presenta una caída exponencial, como se ve en la forma de onda de V_{DS} en la figura 2.25. Al final de T_{DM}, el diodo de salida deja de conducir y ahora la tensión en el drain debe caer a la tensión de entrada. Pero vuelve a ocurrir lo mismo que en el caso anterior, solo que ahora la oscilación es entre L_P y C_P, y por lo tanto de mucha menor frecuencia que en el caso anterior.

Ahora que se dejó en claro la necesidad de una red de clamp con un fuente de tensión, existen dos posibilidades para su implementación. Una es utilizar un circuito RCD y otra es mediante un TVS. Primero analizaremos el circuito RCD, que se puede observar en la figura 2.26.

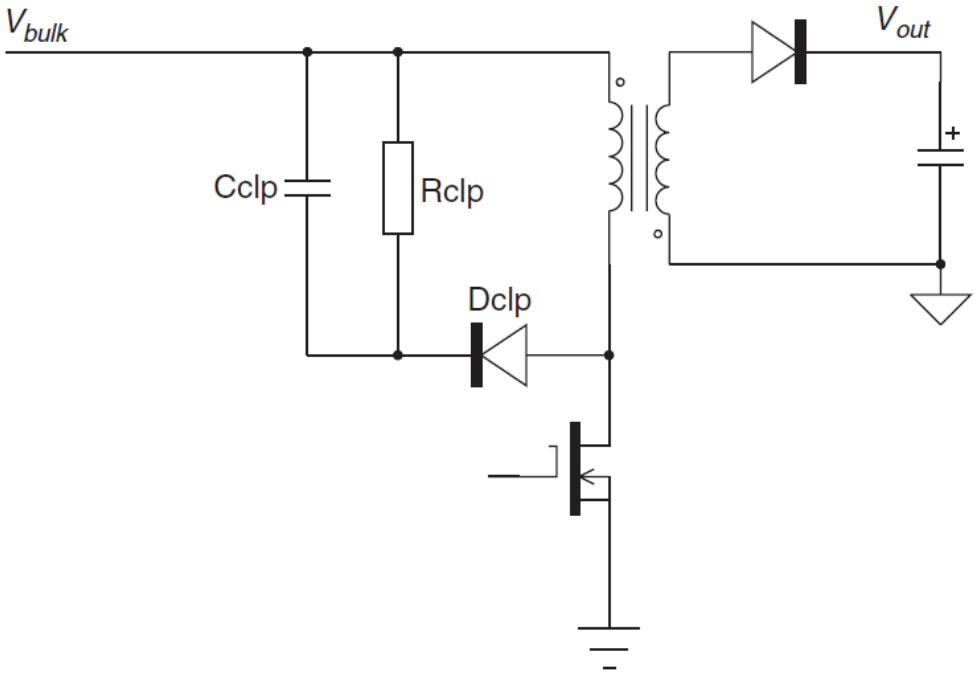


Figura 2.26: Circuito de clamp RCD.

La idea detrás de esta topología, es que la resistencia R_{CLAMP} disipe toda la energía en L_{LEAK} mientras que el capacitor C_{CLAMP} define el ripple en la fuente de tensión. En la ecuación 2.25 se expresa la corriente que circula en D_{CLAMP}:

$$I_{D_{CLAMP}}(t) = I_{PP} * (\frac{\Delta t - t}{\Delta t}) \qquad (2.25)$$

Esta corriente circula mientras se descarga el inductor de pérdidas y disipada una potencia promedio, asumiendo V_{CLAMP} constante, dada por la ecuación 2.26:

$$P_{CLAMP} = f_{SW} * \int_0^{\Delta t} V_{CLAMP} * I_{D_{CLAMP}}(t) * dt = \frac{1}{2} * f_{SW} * V_{CLAMP} * I_{PP} * \Delta t \qquad (2.26)$$

Si ahora reemplazamos Δt por la expresión en la ecuación 2.21, y además como toda la potencia es disipada en forma de calor en la resistencia se puede reemplazar a la potencia promedio por la tensión de clamp al cuadrado sobre la resistencia de clamp. Y si luego despejamos, se obtiene el valor para R_{CLAMP}, dado por la ecuación 2.27:

$$R_{CLAMP} = \frac{2 * V_{CLAMP} * [V_{CLAMP} - (V_0 + V_F) * N_{PS}]}{f_{SW} * L_{LEAK} * {I_{PP}}^2} \qquad (2.27)$$

Por otro lado, C_{CLAMP} define el ripple en V_{CLAMP}. Para calcular el capacitor para una tensión de ripple dada se parte de la ecuación 2.28, que es la ecuación para la carga del capacitor:

$$\Delta V = \frac{Q}{C_{CLAMP}} = \frac{I_{PP} * \frac{\Delta t}{2}}{C_{CLAMP}} \qquad (28)$$

Si ahora se reemplaza nuevamente la ecuación 2.21 en lugar de Δt y se despeja L_{LEAK} de la ecuación 2.27, se obtiene la ecuación 2.29:

$$C_{CLAMP} = \frac{V_{CLAMP}}{R_{CLAMP} * f_{SW} * \Delta V} \qquad (2.29)$$

Esta ecuación permite el cálculo en función de la tensión de ripple deseada. Como este capacitor maneja pulsos de corriente, es necesario calcular la corriente RMS, que está dada por la ecuación 2.30:

$$I_{RMS_{CLAMP}} = \sqrt{\frac{1}{T_{SW}} * \int_0^{\Delta t} \left[I_{PP} * \frac{(\Delta t - t)}{\Delta t} \right]^2 * dt} = I_{PP} * \sqrt{\frac{\Delta t}{3 * T_{SW}}} \qquad (2.30)$$

A continuación se muestran algunas variantes para este circuito en la figura 2.27. El objetivo de estas modificaciones es eliminar las oscilaciones en V_{DS}.

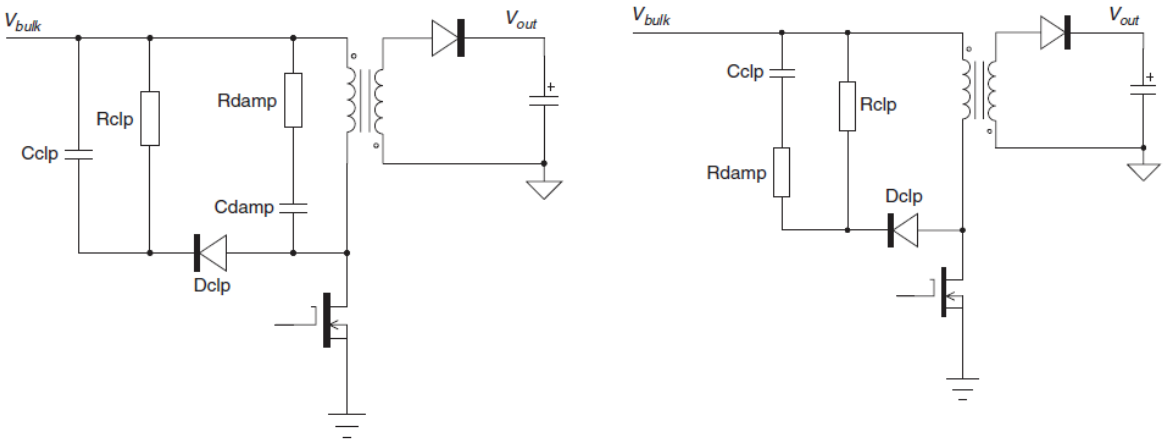


Figura 2.27: Variantes del circuito RCD para eliminar oscilaciones.

La idea detrás de estas redes es aumentar las pérdidas óhmicas en el camino de las oscilaciones, oscilaciones generadas por los elementos parásitos. El cálculo para la resistencia R_{DAMP} en el circuito de la derecha, parte de la ecuación del factor de calidad que se expresa en el ecuación 2.31:

$$R_{DAMP} = \frac{2\pi * f_{OSC} * L_{LEAK}}{Q} \qquad (2.31)$$

Sin embargo, esta resistencia provoca un aumento en el ripple de V_{CLAMP}. Por este motivo, surge la alternativa de la izquierda. Donde ahora ya no se toca la red de clamp y si el transformador. Aunque al estar conectado directamente al transformador puede traer problemas de eficiencia.

En conclusión, implementar la red de clamp con circuito RDC tiene como principal ventaja el bajo costo. Sin embargo, presenta como desventajas la variación de V_{CLAMP} con I_{PP} y si la capacidad resulta de un valor relativamente grande, puede producir pérdidas CV^2 considerables.

La otra alternativa es un utilizar un TVS, circuito que se muestra en la figura 2.28. El TVS no es otra cosa que un diodo de avalancha, que es capaz de soportar grandes pulsos de potencia.

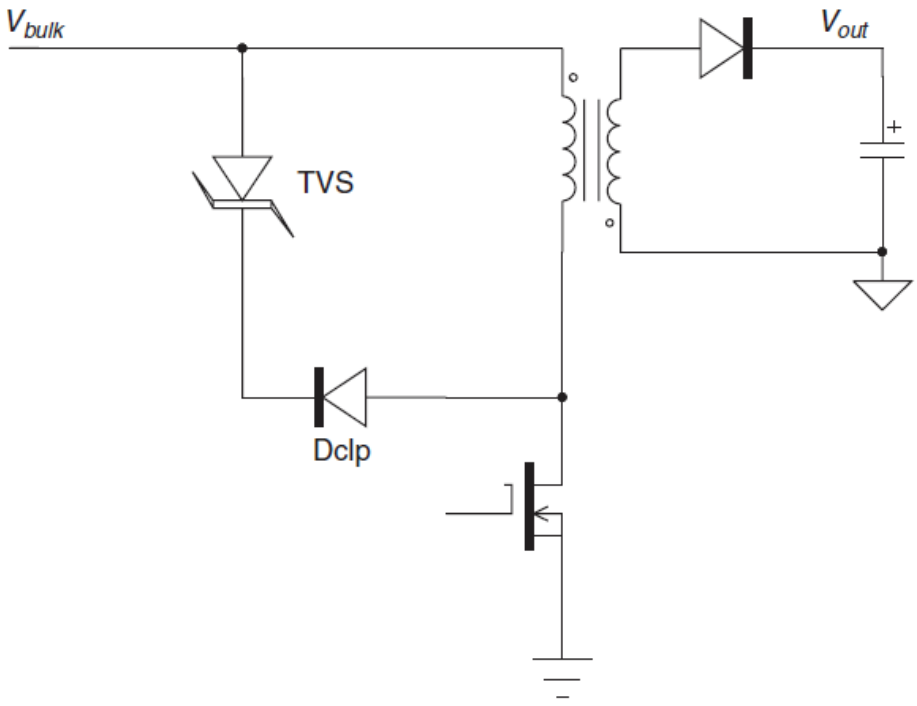


Figura 2.28: Circuito de clamp con TVS.

Toda la potencia, que en la configuración RCD disipaba el resistor, ahora la disipa el TVS, por lo tanto es necesario tener en cuenta este parámetro en el momento de la elección del TVS. La ecuación 2.26 expresa la potencia disipada por el circuito de clamp, pero reemplazando V_{CLAMP} por la tensión de avalancha del TVS, V_Z, y Δt por la ecuación 2.21 se obtiene la 2.32:

$$P_{TVS} = \frac{1}{2} * f_{SW} * L_{LEAK} * {I_{PP}}^2 * \left[\frac{V_Z}{V_Z - (V_0 + V_F) * N_{PS}}\right] \qquad (2.32)$$

Para esta aplicación el TVS ofrece las siguientes ventajas:

- Una tensión de clamp más estable con respecto a la configuración RCD, sin ripple y con muy poca variación con I_{PP}.
- Sin carga en la salida, el pulso generado por L_{LEAK} no suele alcanzar V_Z, por lo tanto el TVS se mantiene transparente a la SMPS y aumenta la eficiencia. Generalmente en condiciones de muy poca carga los convertidores más modernos suelen adoptar una técnica llamada skip cycle, en la que el convertidor omite pulsos para poder regular a muy bajas potencias. Esta característica no permite la implementación de un circuito de clamp RCD, ya que para mantener V_{CLAMP} el capacitor necesita pulsos para cargarse. En definitiva con un TVS es posible obtener regulación para una menor potencia en la salida, con respecto a RCD.
- Este dispositivo presenta una capacidad muy baja, lo cual es beneficioso para las pérdidas CV^2.

La desventaja que presenta este dispositivo es el alto costo relativo, comparado con la solución de bajo costo que presenta RCD.

Finalmente se decidió por el uso de un TVS, por las ventajas que presenta para la eficiencia de la SMPS. Solo resta definir V_{CLAMP}, parámetro que, como se explicó anteriormente, conviene que sea lo mayor posible con respecto a la tensión reflejada desde el secundario. Pero este valor está limitado por V_{DS}. Por lo tanto, la solución pareciera elegir un MOSFET con un valor alto de V_{DS}, sin embargo, las características del MOSFET suelen empeorar a medida que aumenta este parámetro. Se terminó optando por un MOSFET con $V_{DS(MAX)}$ = 1000V, lo que dejando un margen de aproximadamente un 20% de seguridad, nos permite llegar a los 800V en la tensión de drain-source. Teniendo en cuenta que la máxima V_{BULK} es:

$$V_{BULK(MAX)} = V_{LINEA(MAX)} * \sqrt{2} = 253V_{RMS} * \sqrt{2} \cong 358V$$

Esto deja un margen para una tensión de clamp de un poco más de 400V, por lo que se adoptó un TVS con una V_Z de 400V.

Circuito del bobinado auxiliar

La señal del bobinado auxiliar es sensada mediante un divisor resistivo, para obtener información de la entrada y de la salida, y también para alimentar al controlador. El primer paso aquí es determinar la relación de vueltas entre el bobinado auxiliar y el bobinado secundario, N_{AS}. Este parámetro se elige dependiendo con la tensión de alimentación, V_{DD}, con la que se quiera trabajar. Mediante la ecuación 2.33 se calcula N_{AS}:

$$N_{AS} = \frac{V_{DD}+V_{FA}}{V_0+V_F} = \frac{10V+0.5V}{5V+0.7V} \cong 1.84 \qquad (2.33)$$

Donde VFA es la tensión en directa del diodo rectificador del auxiliar y V_F es la del diodo de salida. Los valores anteriores dependen de la corriente que circula por los diodos, y como en la salida hay picos de corriente muy elevados, V_F es mayor a V_{FA}. Para obtener una tensión de salida de aproximadamente 10V, se adopta N_{AS} = 2 y se recalcula V_{DD}, resultando con un valor de 10.9V.

Luego se calcula las dos resistencias del divisor. Primero calculamos mediante la ecuación 2.34 la relación de vueltas entre primario y auxiliar:

$$N_{PA} = \frac{N_P}{N_A} = \frac{N_{PS}}{N_{AS}} = \frac{N_P*N_S}{N_A*N_S} = \frac{38}{2} = 19 \qquad (2.34)$$

Ahora mediante la ecuación 2.35 calculamos la resistencia superior del divisor, encargada de sensar la tensión de entrada:

$$R_{S1} = \frac{V_{BULK(MIN)}}{N_{PA}*I_{VSL(RUN)}} = \frac{210V}{19*225\mu A} \cong 49.12K\Omega \qquad (2.35)$$

Donde $I_{VSL(RUN)}$, es el nivel de corriente que sale del pin VS que permite que inicie el controlador. Se adoptó el valor comercial de 48.7KΩ para R_{S1}.

Ya con R_{S1}, haciendo uso de la ecuación 2.36 se calculó el valor para la resistencia inferior del divisor:

$$R_{S2} = \frac{R_{S1} * V_{OVP}}{N_{AS} * (V_{OV} - V_F) - V_{OVP}} = \frac{48.7K\Omega * 4.6V}{2 * (6..25V - 0.7V) - 4.6V} \cong 34K\Omega \qquad (2.36)$$

Donde V_{OV}, es la máxima tensión permitida en la salida y V_{OVP} es el máximo de nivel tensión permitido en el pin VS en el momento de sensar V_0. Como se verá más adelante, la tensión de salida está limitada a 5.7V aproximadamente, por lo tanto a esta protección se le asignó un valor mayor, permitiéndose un sobrepico del orden del 25%. Se llevó a R_{S2} al valor comercial de 33.2KΩ.

Limitador en la salida

El controlador ofrece una protección contra la sobretensión en la salida, pero se detiene el controlador. Por este motivo, se añadió un limitador en la salida que permite un sobrepico en V_0, pero manteniendo la regulación. El circuito consiste en un transistor operando en zona lineal con un diodo zener para limitar la tensión de salida, el cual se muestra se muestra en la figura 2.29:

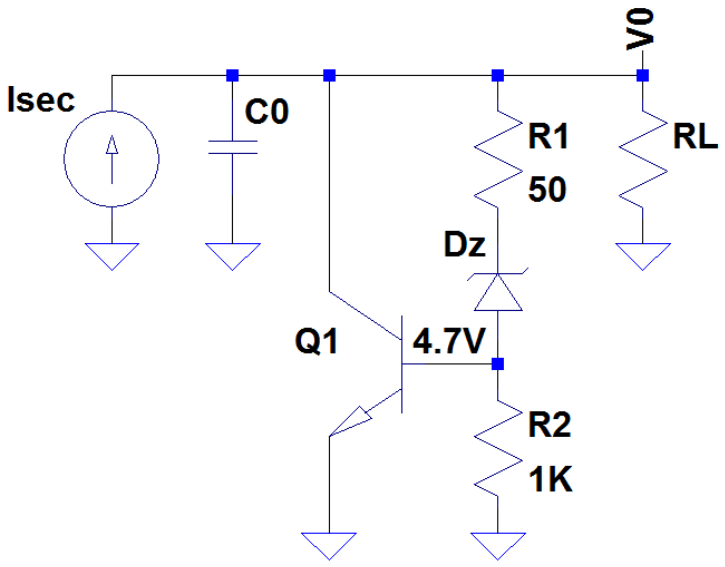


Figura 2.29: Circuito limitador en la salida.

El convertidor Flyback se comporta como un generador de corriente, genera una corriente en forma de diente de sierra que es filtrada por el capacitor de salida y por la carga solo circula la corriente continua I_0. Cuando el convertidor envía potencia en exceso, la corriente de salida aumenta y la tensión en la carga tiende a subir. Pero la ecuación 2.37 describe la tensión de salida con el limitador actuando:

$$V_{0(MAX)} = V_{CE}(I_C) = V_{BE}(I_B) + V_Z(I_{R1}) + I_{R1} * R_1 \qquad (2.37)$$

En esta expresión se ve como V_0 está limitada por los diodos base-emisor y zener. Para aumentar la tensión en la salida, el limitador toma corriente de I_0 para aumentar I_{R1} e I_B de forma que aumenten las tensiones en los diodos. Por lo tanto, cuanto mayor sea la corriente en exceso en I_0, mayor será la corriente por el diodo zener y el diodo base-emisor, aunque debido a la pequeña

impedancia que presentan los diodos en su zona de trabajo, solo se produce un pequeño aumento en la tensión.

Con un zener de potencia o un TVS se podría lograr lo mismo, sin embargo, estos dispositivos están limitados en corriente o potencia. Por lo tanto, se optó por utilizar un transistor para el manejo de corriente.

Circuito de la realimentación

Como se detalló anteriormente, se necesita realimentar información desde el circuito secundario. Esto se hace mediante la combinación de un opto-acoplador, para mantener la aislación eléctrica entre primario y secundario, y un TL431, que combina una referencia de precisión y un amplificador de error. Primero comenzaremos con un análisis de las distintas configuraciones para el circuito de realimentación, cuya diferencia esta fundamentalmente en la respuesta dinámica. El primer circuito a analizar es el más simple y se muestra en la figura 2.30.

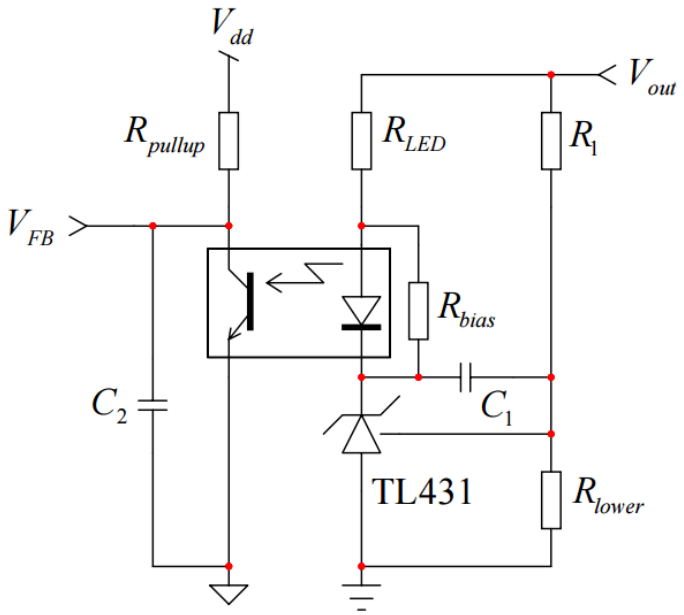


Figura 2.30: Primera opción para el circuito de realimentación.

El funcionamiento de este circuito ya fue explicado en la sección 1.5, ahora nos concentraremos en la respuesta dinámica que presenta el circuito de realimentación. En la figura 2.31 se muestra el esquema para AC. En el modelo de alterna, la tensión de referencia se hace cero, y debido a las características del amplificador operacional, V+ y V- son iguales a cero. Por esto último, R_{LOWER} desaparece del modelo dinámico. Esta resistencia solo tiene importancia en continua, donde establece junto R_1 la tensión de salida y también proporciona alimentación al pin REF del TL431.

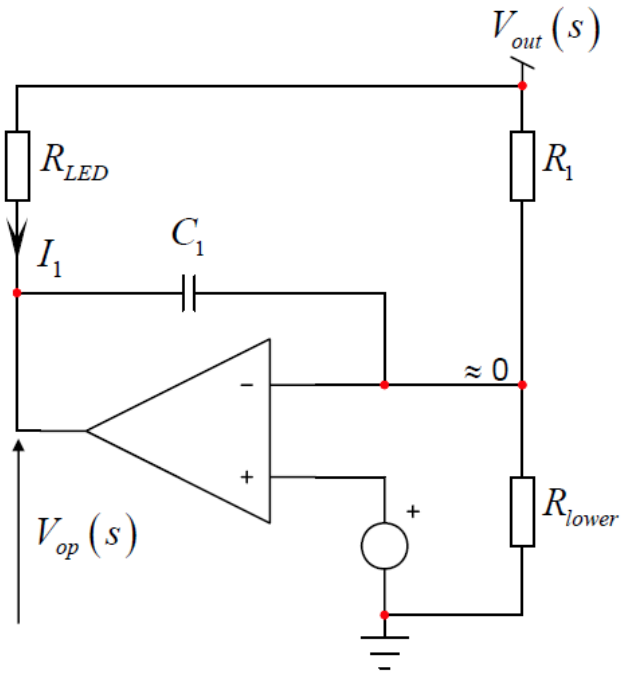


Figura 2.31: Modelo dinámico del circuito de realimentación.

En este modelo tampoco fue tenida en cuenta la resistencia que asegura la mínima corriente de alimentación requerida en el cátodo del TL431, R_{BIAS}. Esta resistencia genera un divisor de corriente de I_1. Pero como la resistencia dinámica que presenta del diodo del opto-acoplador es muy chica en comparación en comparación a R_{BIAS}, en el modelo dinámico esto simplemente genera una ganancia cercana a la unidad.

Teniendo en cuenta la simplificación anterior, la ecuación 2.38 muestra la expresión para la corriente I_1:

$$I_1(s) = \frac{V_0(s) - V_{OP}(s)}{R_{LED}} \qquad (2.38)$$

Pero V_{OP} está dada por la ecuación 2.39:

$$V_{OP}(s) = \frac{-V_0(s)}{s * C_1 * R_1} \qquad (2.39)$$

Ahora introduciendo la ecuación 2.39 en la 2.38 y despejando la transferencia de $V_0(s)$ a $I_1(s)$, se obtiene la ecuación 2.40:

$$\frac{I_1(s)}{V_0(s)} = -\frac{1}{R_{LED}} \frac{(1 + sR_1C_1)}{sR_1C_1} \qquad (2.40)$$

I_1 es la corriente que circula por el diodo del opto-acoplador, para obtener la corriente del fototransistor es necesario multiplicar a I_1 por el CTR el opto-acoplador. La corriente del fototransistor genera la tensión de realimentación mediante la resistencia de pull-up, R_{PULLUP}. Pero falta añadir otra singularidad, que es generada por la capacidad parásita del fototransistor, C_2.

Considerando lo anteriormente explicado, se puede escribir la transferencia de la tensión de salida a la tensión de realimentación, como se expresa en la ecuación 2.41:

$$\frac{V_{FB}(s)}{V_0(s)} = -\frac{R_{PULLUP}CTR}{R_{LED}} \frac{(1+sC_1R_1)}{sC_1R_1(1+sC_2R_{PULLUP})} \qquad (2.41)$$

Esta transferencia representa la respuesta dinámica de este circuito de realimentación. Sin embargo, la configuración que se presenta en la figura 2.32 tiene algunas ventajas con respecto a la anterior.

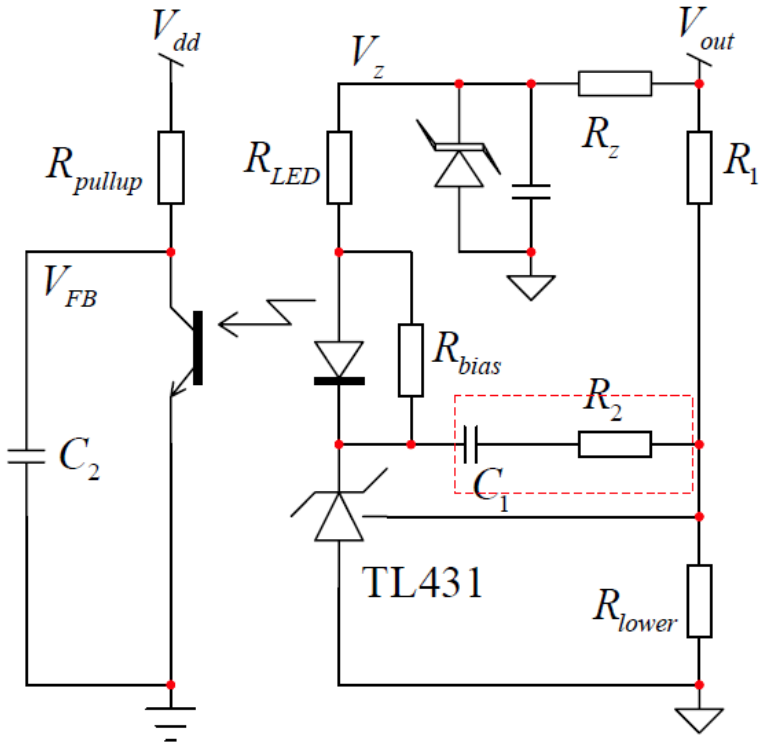


Figura 2.32: Segunda opción para el circuito de realimentación.

Ahora se alimenta al circuito de realimentación mediante un diodo zener, y ya no desde la tensión de salida. Con esto desaparece el cero en la transferencia, por lo que se agrega R_2. Mediante el agregado de esta resistencia, vuelve a haber un cero en la transferencia. Si se hace el mismo procedimiento que para el circuito anterior, se obtiene la transferencia expresada por la ecuación 2.42:

$$\frac{V_{FB}(s)}{V_0(s)} = -\frac{R_{PULLUP}CTR}{R_{LED}} \frac{(1+sC_1R_2)}{sC_1R_1(1+sC_2R_{PULLUP})} \qquad (2.42)$$

Comparando las transferencias de los dos circuitos de realimentación, no se observan muchas diferencias. Aunque el segundo circuito presenta dos ventajas con respecto al primero:

- La tensión de alimentación del circuito es más estable. Una vez que la salida supere el valor V_Z, la tensión de alimentación se mantendrá estable, independientemente de las variaciones en V_0.

- También relacionado con la alimentación, se puede entregar la corriente al diodo zener desde la salida de +15V y utilizar una tensión de zener mayor. Con esto se logra que la tensión en el cátodo del TL431 tenga un mayor margen de variación.
- Mayor flexibilidad en la compensación. Como se mencionó anteriormente, ahora que la tensión de alimentación no depende más de V_0, desaparece el cero en la transferencia. Pero este cero que deja de estar, tiene la misma constante de tiempo que el polo en el origen, C_1R_1. Ahora con el agregado de R_2, las constantes de tiempo son diferentes entre el cero y el polo en el origen. Esto permite mayor flexibilidad en el momento de compensar el lazo de control.

Estos tres puntos determinaron que se utilice la segunda configuración para el circuito de realimentación. Aunque también existen otras alternativas, como las que se presentan en la figura 2.33 y 2.34.

Estas dos alternativas agregan tanto un cero como un polo más a la transferencia, que pueden ser utilizados para mejorar la compensación. Y su vez, la diferencia entre ambos, es la misma que hay entre el primer y segundo circuito, es de donde proviene la alimentación. Por esto último, el cuarto circuito presenta las mismas ventajas sobre el tercero, que aquellas que presenta el segundo sobre el primero. Sin embargo, la compensación en este caso no requiere de una red compleja y se puede resolver de manera más sencilla con el segundo circuito de realimentación.

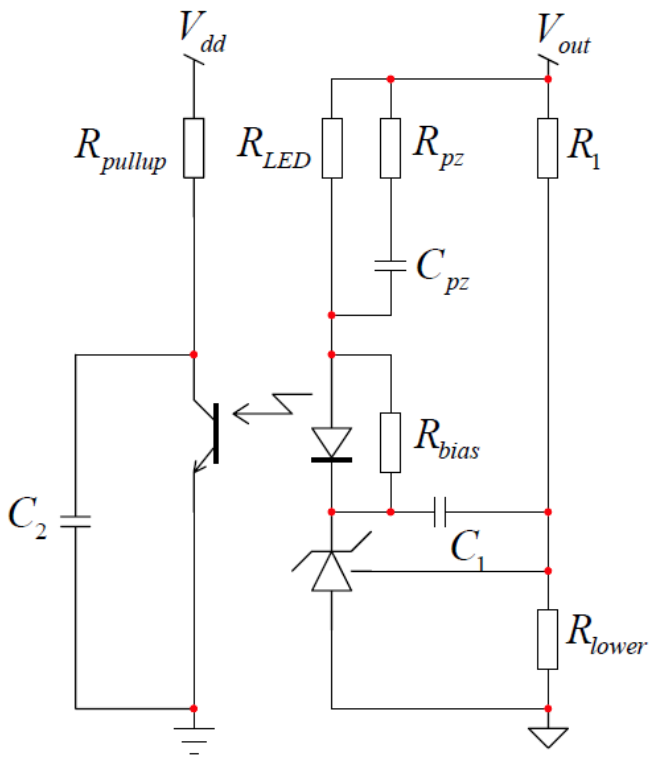


Figura 2.33: Tercera alternativa para el circuito de realimentación.

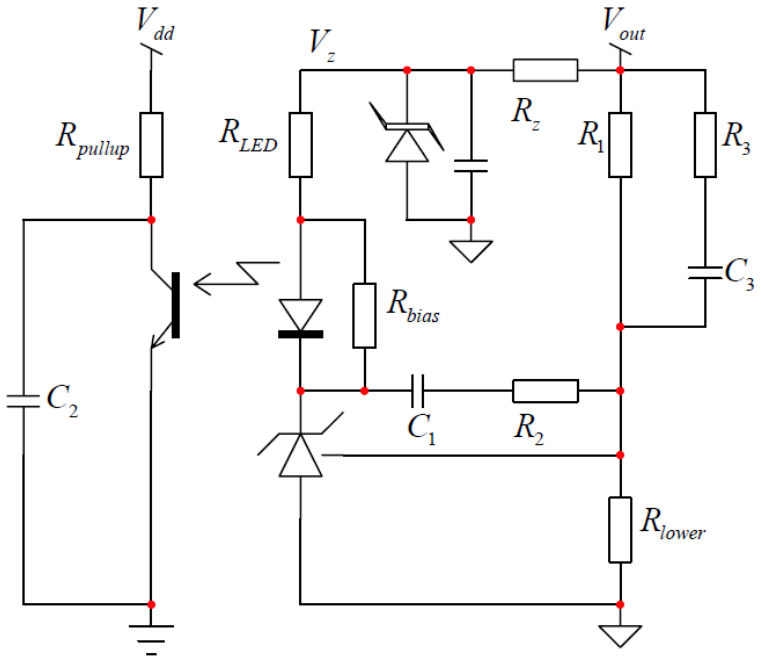


Figura 2.34: Cuarta alternativa para el circuito de realimentación.

En conclusión se adoptó la segunda configuración, pero al controlador hay que realimentarle corriente, y no tensión como se hace en el circuito de la figura 2.32. Considerando esto último, el circuito de realimentación queda como lo muestra la figura 2.35.

Recordando que internamente el pin FB presenta una resistencia de 14kΩ con un diodo en serie, como se observa en el diagrama en bloques interno del UCC28740 en la figura 2.18, se forma un divisor resistivo entre dicha impedancia, R_i, y la resistencia R_{F8}.

Ahora teniendo esto en cuenta, en la ecuación 2.43 se presenta la respuesta dinámica del circuito de realimentación seleccionado.

$$\frac{I_{FB}(s)}{V_0(s)} \approx \frac{R_i}{R_i+R_{F8}} \frac{CTR}{[1+sC_{OPT}(R_{F8}//R_i+R_{F7})]} \frac{-1/R_{F5}(1+sC_{F1}R_{F3})}{sC_{F1}R_{F1}} \qquad (2.43)$$

Por otro lado, el modelo de la planta es el de un convertidor Flyback operando en DCM con control en modo corriente. El modelo simplificado presenta una sola singularidad, un polo con la constante de tiempo de C_0 y la carga que ve la fuente, R_L.

El capacitor de salida, además de influir en la dinámica de la plata, también define el ripple en la tensión de salida. La frecuencia fundamental del ripple es la de conmutación y mientras más grande sea la constante de tiempo del filtro RC de la salida, menos ripple habrá en V_0. En consecuencia, un valor alto en C_0 implica menos ripple y una planta lenta, mientras que un valor bajo implica todo lo contrario.

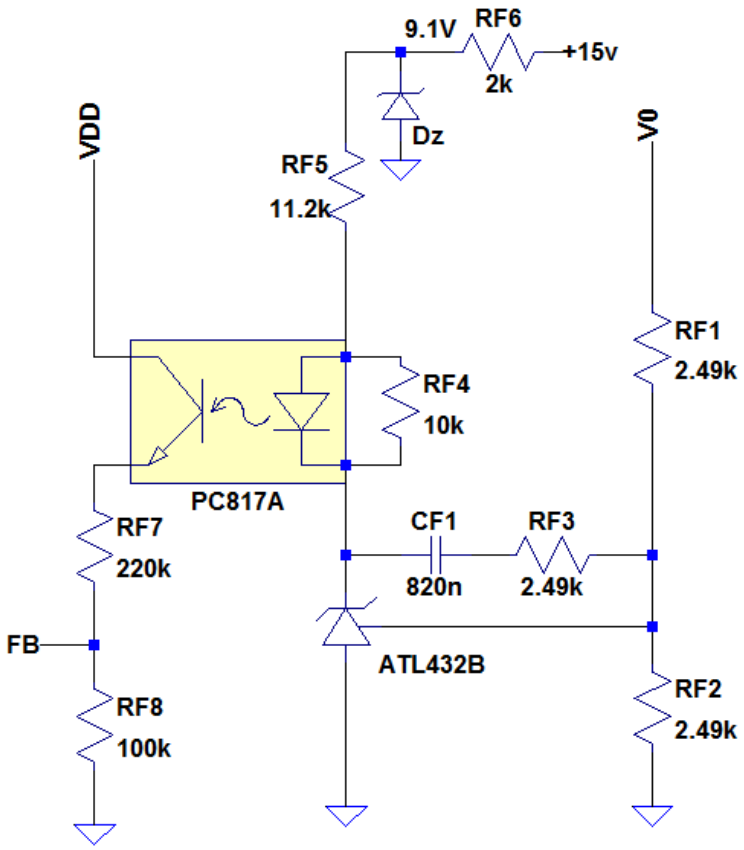


Figura 2.35: Circuito de realimentación.

Pero además de la constante de tiempo del filtro, existe otra limitación para reducir el ripple, el ESR (Resistencia serie equivalente) del capacitor. Esta resistencia produce ripple en la tensión de salida que no es filtrada. Con lo cual, se eligió un capacitor de un valor elevado, que permita una corriente RMS elevada y con un ESR bajo. La corriente RMS en el capacitor es muy elevada en la topología Flyback. Esta corriente disipa potencia mediante ESR, provocando un aumento de temperatura en el capacitor y reduciendo su tiempo de vida, motivo por el cual era necesario un capacitor que soportara corrientes RMS de gran valor. Además, para mejorar aún más la ESR de C_0, se colocó en paralelo a éste un capacitor cerámico con ESR aún más bajo. Finalmente se adoptó un valor de 470µF para C_0 y de 33µF para el capacitor cerámico.

En el momento de estabilizar, se priorizó compensar para cualquier carga externa en la SMPS. Esto último implica que el peor caso para estabilizar está dado por la carga que presenta el mismo circuito de realimentación, ya que es el caso de la constante de tiempo más lenta.

En conclusión, debido al valor elevado en el capacitor de salida y la poca carga que se presenta en el peor caso, la planta presenta una dinámica lenta.

Un modelo más preciso muestra otras singularidades, sin embargo son todas de muy alta frecuencia. Y debido a que se compensara la SMPS para cualquier carga externa, las singularidades de alta frecuencia no influirán en la dinámica del lazo.

Si bien no se realizó una compensación basada en técnicas para el análisis de la estabilidad, si se conocía la transferencia del circuito de realimentación y el modelo dinámico de la planta. De esta forma, sabiendo que rol juega cada componente en la dinámica del lazo y su función en continua, se pudieron adoptar los valores de los componentes que permitieran conseguir la respuesta dinámica deseada.

Salvo por los componentes de la realimentación local en el TL431, C_{F1} y R_{F3}, el resto de los componentes cumple una función en continua. A continuación analizaremos la función y el cálculo para cada componente de la realimentación.

Antes de comenzar el análisis, se detallan algunas características del TL431 que son relevantes para el diseño. Existen muchas variantes para el TL431, en cuanto a la precisión, las corrientes de alimentación, el valor de la tensión de referencia, etc. Dentro de estas posibilidades se escogió el ATL432B, esta versión cuenta con una tensión de referencia de 2.5V, tolerancia del 0.5%, una corriente de alimentación en el cátodo de 35µA y de 0.150µA en el pin REF.

R_{F1} y R_{F2}

Las resistencias R_{F1} y R_{F2} en continua establecen la tensión de salida y proporcionan la corriente de alimentación al pin REF del ATL432B.

La corriente I_{BIAS} que se muestra en el circuito de la figura 2.36, es la corriente de alimentación de que demanda el pin REF. Esta corriente influye en la tensión de salida. Mirando el circuito se obtiene la ecuación 2.44:

$$V_{LOWER} = (I_{BRIDGE} - I_{BIAS}) * R_{LOWER} \qquad (2.44)$$

Mientras que la corriente I_{BRIDGE} esta expresada en la ecuación 2.45:

$$I_{BRIDGE} = \frac{(V_0 - V_{LOWER})}{R_{UPPER}} \qquad (2.45)$$

Pero en régimen permanente, V_{LOWER} es igual a la tensión de referencia. Si introducimos este cambio, la expresión de la ecuación 2.45 en la 2.44 y además despejamos la tensión de salida, se obtiene la ecuación 2.46:

$$V_0 = \left(\frac{R_{UPPER}}{R_{LOWER}} + 1\right) * V_{REF} + R_{LOWER} * I_{BIAS} \qquad (2.46)$$

Lo primero que se observa, es que para obtener una tensión de 5V en la salida es necesario que ambas resistencias sean iguales. Sin embargo, aparece un término que produce un error en V_0. Este término depende de la corriente de alimentación que demanda el pin REF. Para minimizar este error, la solución es que I_{BIAS}, que es como máximo 0.15µA para el ATL432B, sea despreciable frente I_{BRIDGE}.

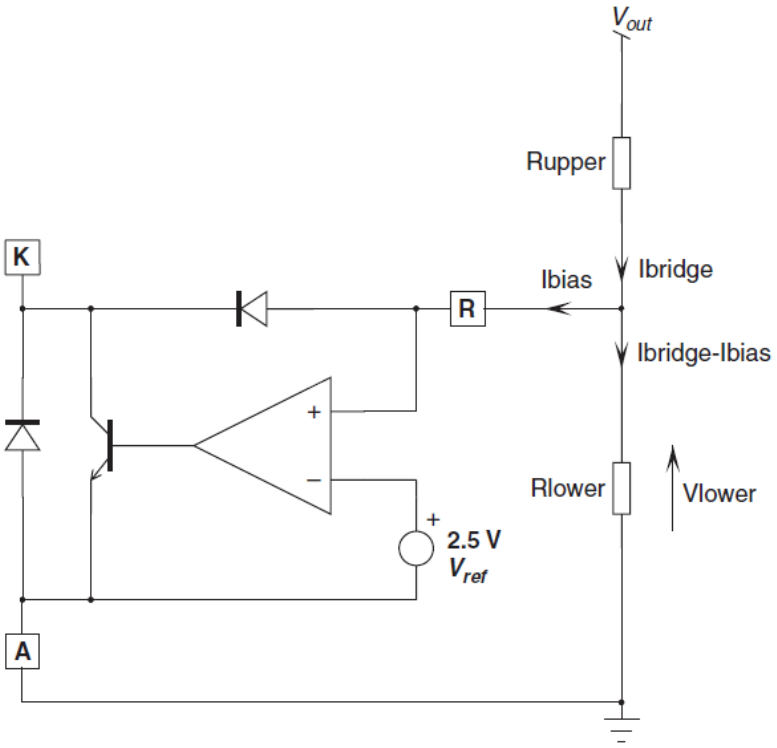


Figura 2.36: Circuito que muestra como se establece la tensión de salida.

Para adoptar el valor de estas resistencias, existe una relación de compromiso entre distintas cuestiones:

- Como se mostró cuando se hizo un análisis de la respuesta dinámica de este circuito, este valor influye en la ubicación de uno de los polos de la transferencia (ver ecuación 2.44). A medida que disminuye este valor, más rápida es la respuesta. Si en cambio se aumenta este valor, la respuesta es más lenta.
- Un valor elevado en estas resistencias implica un mayor error en la tensión de salida, como lo muestra la ecuación 2.47.
- Estas dos resistencias también presentan una carga para la fuente de alimentación. Si se desea obtener una baja potencia en la salida sin carga, estas resistencias tienen que ser elevadas.

Considerando los puntos anteriores, se adoptó un valor comercial de 2.49KΩ para R_{F1} y R_{F2}. Para esta elección se priorizo tener un error pequeño en la tensión de salida. Estas resistencias tienen una tolerancia de 0.1%. Esta tolerancia es cinco veces mejor a la que presenta el ATL432B, de esta forma se introduce poco error en V_0.

C_{F1} y R_{F3}

Mientras que elección del valor de R_{F1} y R_{F2} se tomó de forma tal que la tensión de salida en continua tenga el menor error posible, para obtener la respuesta dinámica deseada se ajustaron los valores de C_{F1} y de R_{F3}. Estos valores definen la posición del polo y el cero de la transferencia del amplificador de error, por lo tanto además de buscar la dinámica pretendida para el lazo de realimentación de V_0, también se tiene que asegurar estabilidad en el lazo interno del ATL432B.

R_{F4}

Por otro lado, en continua la resistencia R_{F4} es la que asegura la correcta alimentación del regulador shunt. El ATL432B necesita de una corriente de cátodo, I_{KA}, de al menos 35µA para su correcto funcionamiento. Aprovechando la tensión en directa de diodo LED del opto-acoplador, V_{FOPT}, se coloca una resistencia para suministrar la corriente de alimentación requerida. Si bien V_{FOPT} depende de la corriente que circule por el diodo, que para las bajas corrientes que maneja el controlador, esta tensión será cercana a 1V. Con lo cual se adoptó un valor 10KΩ para R_{F4}, produciendo una corriente de alimentación de aproximadamente 100µA.

R_{F8}

La resistencia R_{F8} se coloca para acelerar la respuesta del opto-acoplador, establecer un corriente mínima y reducir el efecto de las corrientes de oscuridad del fototransistor. Se coloca en paralelo al pin FB, formando un divisor de corriente con la resistencia interna del pin FB, de valor nominal de 14KΩ. Este paralelo, sumado a R_{F7} dan como resultado la resistencia de carga que ve el fototransistor. Y como se analizó anteriormente, la carga y la capacidad parásita del fototransistor determinan la posición de uno de los polos de la transferencia del circuito de realimentación (ver ecuación 2.44). Por lo tanto, adoptando un valor bajo se reduce la resistencia de carga, provocando una aceleración en la respuesta. Sin embargo, existe una relación de compromiso entre acelerar la velocidad de la respuesta del lazo reduciendo R_{F8}, y el consumo del circuito.

Por otro lado, la corriente mínima que se fija mediante esta resistencia debe ser mayor a las corrientes de oscuridad del opto-acoplador. La mínima corriente en el pin FB, en la cual hay regulación, es de 0.5µA. Esta corriente es mayor a las corrientes oscuras del opto-acoplador en cualquier condición de operación, por lo tanto no es necesario imponer un límite superior para R_{F8}.

En conclusión, al no requerirse una respuesta rápida por parte del lazo, se adoptó un valor de 100KΩ para esta resistencia.

R_{F7}

La función que cumple esta resistencia es limitar la corriente en el pin de FB. Esto es necesario ya que se puede dar el caso de pasar de una situación de carga máxima, donde I_{FB} es mínima, a una situación con carga mínima, donde I_{FB} es máxima. Como respuesta el lazo de control saturará el fototransistor y circulará una corriente I_{FB} mayor a la permitida por el UCC28740. Por lo tanto se elige un valor para R_{F7} que limite la tensión en el pin FB a 1V, que se corresponde con una I_{FB} de 32µA, dando aproximadamente el valor máximo para esta corriente.

La ecuación 2.47 muestra cual es la máxima corriente que debería circular por el fototransistor:

$$I_{C(MAX)} = \frac{V_{FB(MAX)}}{R_{F8}} + I_{FB(MAX)} = \frac{1V}{100K\Omega} + 32\mu A = 10\mu A + 32\mu A = 42\mu A \qquad (2.47)$$

Ahora ya conociendo este valor, la ecuación 2.48 muestra como es el cálculo para esta resistencia:

$$R_{F7} = \frac{V_{DD} - V_{CE(SAT)} - V_{FB(MAX)}}{I_{C(MAX)}} = \frac{10.9V - 0.2V - 1V}{42\mu A} \cong 231K\Omega \qquad (2.48)$$

Finalmente, se adopta un valor comercial de 220KΩ para esta resistencia, y recalculando I_C máxima resulta de 44µA.

R_{F5}

Esta resistencia cumple la misma función que la anterior, solo que esta limita la corriente en el diodo LED del opto-acoplador. La máxima corriente en R_{F5} que permite regulación está dada por la ecuación 2.49:

$$I_{R_{F5}(MAX)} = \frac{I_{C(MAX)}}{CTR_{MIN}} + \frac{V_{F_{OPT}}}{R_{F4}} = \frac{42\mu A}{0.10} + \frac{1V}{10K\Omega} = 420\mu A + 100\mu A = 520\mu A \qquad (2.49)$$

El CTR mínimo es muy chico debido a las bajas corrientes a las que opera el UCC28740. Considerando esta corriente máxima, que se da cuando V_{KA} es mínimo, se obtiene la expresión para calcular esta resistencia en la ecuación 2.50:

$$R_{F5} = \frac{V_Z - V_{F_{OPT}} - V_{KA(MIN)}}{I_{R_{F5}(MAX)}} = \frac{9.1V - 1V - 2.5V}{520\mu A} \cong 10.8K\Omega \qquad (2.50)$$

Un valor comercial de 11.2KΩ es adoptado para R_{F5}.

R_{F6}

Esta resistencia se coloca para establecer la corriente en el diodo zener. Con un valor de 2KΩ para esta resistencia se obtiene una corriente de:

$$I_{R_{F6}} = \frac{V_{CC} - V_Z}{R_{F6}} \cong \frac{15V - 9V}{2K\Omega} \cong 3mA$$

D_Z

Por lo general se utiliza el diodo zener desde la salida principal, pero 5V dejaba poco margen para la regulación. Con lo cual, se eligió un valor de 9.1V para el zener y suministrarle corriente desde la salida $+V_{CC}$.

2.8 Transformador

Lo primero a definir en el diseño del transformador es el núcleo, tanto el material, el tamaño y la forma. El propósito fundamental del núcleo es proveer un camino que facilite el acoplamiento entre dos o más elementos magnéticos. En un transformador convencional no se requiere el almacenamiento de energía, por el contrario, es una característica que se tiene que evitar en el diseño de tales transformadores, en cambio el transformador Flyback si lo requiere. El

transformador Flyback es en realidad un inductor con bobinados primario y secundario y un gap que almacena la energía necesaria. El núcleo brinda un camino de baja reluctancia para el flujo, con el fin de acoplar de forma eficiente los devanados primario y secundario. Los bobinados separados proveen aislación eléctrica entre los circuitos primario y secundario, y la relación de vueltas entre los bobinados posibilita la conversión de tensiones.

Debido a la elevada frecuencia a la que opera la SMPS y el bajo costo, la mejor opción es utilizar un núcleo de ferrite. El ferrite es el de menores pérdidas a alta frecuencia, en comparación con los otros materiales. También presenta un flujo de saturación mucho menor que otros materiales, sin embargo en la SMPS la restricción la imponen las pérdidas, producto de las grandes corrientes RMS de la topología Flyback. La principal desventaja del ferrite es que es menos robusto que los otros materiales. En conclusión, se eligió el material N87 de TDK. Este material está especificado para uso en SMPS que trabajan a altas frecuencias y dentro de las opciones disponibles ofrece bajas pérdidas y una densidad de flujo magnético de saturación elevada.

Una vez que el material es elegido, lo que sigue es determinar el tamaño del núcleo. Como una primera aproximación se utilizó el método del producto de áreas, que se detalla a continuación y consiste en encontrar el producto entre el área efectiva de la sección del núcleo y el área de la ventana del trasformador. Partiendo desde la ley de Faraday para el bobinado primario se obtiene la ecuación 2.51, que expresa la tensión RMS para el primario:

$$V_P = \frac{d\varphi}{dt} = 4.62 D N_P A B 2\pi f \qquad (2.51)$$

Donde D es el ciclo de trabajo de la señal PWM, N_P es el número de vueltas de la bobina del primario, A es la sección del núcleo, B es la densidad de flujo magnético y f es la frecuencia de conmutación.

Aplicando lo mismo al bobinado secundario se obtiene la ecuación 2.52:

$$V_S = \frac{d\varphi}{dt} = 4.62 D N_S A B 2\pi f \qquad (2.52)$$

Donde el coeficiente de 4.62 es un factor que tiene en cuenta la conversión entre densidad de flujo pico y RMS, para una corriente con forma de onda diente de sierra. Ahora planteamos la ecuación 2.53 para el área de la ventana como la suma del espacio necesario para bobinar el primario y el secundario:

$$K_W A_W = S_P N_P + S_S N_S = \frac{I_P}{J} N_P + \frac{I_S}{J} N_S \qquad (2.53)$$

Donde K_W es el factor de utilización de la ventana. Con la inclusión de este término se tiene en cuenta todo el espacio que no es utilizado para bobinar, esto incluye márgenes que se dejan por razones de seguridad, capas de aislación, el esmalte de cobre, etc. S_P y S_S son las secciones de los alambres de cobre que se utilizan para el bobinado primario y secundario respectivamente. La sección está dada por la corriente RMS en cada bobinado y por la densidad de corriente adoptada. Si despejamos el número de vueltas en el primario y el secundario de las ecuaciones 2.51 y 2.52 respectivamente, y lo introducimos en la ecuación 2.53 y luego despejamos A_W se obtiene la ecuación 2.54:

$$A_W = \frac{V_P}{4.62DK_W A_e B_{AC} f}\frac{I_P}{J} + \frac{V_S}{4.62DK_W A_e B_{AC} f}\frac{I_S}{J} \qquad (2.54)$$

Donde A_e es el área efectiva de la sección del núcleo y B_{AC} es la densidad de flujo en alterna. Luego si despejamos el producto de área se obtiene la ecuación 2.55:

$$AP = A_W A_e = \frac{V_P I_P - V_S I_S}{4.62DK_W JB_{AC} f} = \frac{P_{IN} + P_0}{4.62DK_W JB_{AC} f} = \frac{(1+\eta)P_0}{4.62D\eta K_W JB_{AC} f} 10^4\ mm^4 \qquad (2.55)$$

Se ve como aparecen en la ecuación 2.56 las potencias de entrada y de salida, y se relacionan las potencias mediante la eficiencia. También se incluye un término al final de la ecuación que pasa de m^4 a mm^4.

Con esta ecuación estamos en condiciones de hacer una primera aproximación para el AP. Para el cálculo se toma como densidad de flujo en alterna 150mT, que es aproximadamente la mitad de la densidad de flujo de saturación para un núcleo de ferrite. También se adopta un típico valor para el factor de utilización de la ventana, 30%. Mientras que se tomó una densidad de corriente de 300 A/cm^2, ya que con este valor se obtiene una buena disipación de calor en el alambre de cobre. El cálculo resulta:

$$AP = \frac{(1+0.85)*11W*10^4}{4.62*0.481*0.85*0.3*300\frac{A}{cm^2}*0.15T*100KHZ} \cong 0.08cm^4 \cong 800mm^4$$

Ya con una primera aproximación del producto de áreas, se selecciona una forma para el núcleo. Se optó por una forma de núcleo RM (Rectangular Modular), ya que encierra de mejor manera el flujo y deja expuesto solo una pequeña parte del bobinado, con respecto a otras formas.

Si bien con el núcleo RM 8 se cumple con el AP calculado con buen margen, debido a la cantidad de bobinados (cinco), a la inexperiencia en el bobinado de transformadores y para mejorar aún más la eficiencia del transformador, se termina adoptando un núcleo RM 10. Esta elección aún debe ser respaldada por el diseño del transformador, fundamentalmente por las pérdidas y por un esquema que verifique que se tenga el suficiente espacio.

Ya con el núcleo definido, ahora se debe determinar el número de vueltas para cada bobinado y la densidad de flujo. La densidad de flujo está relacionada con las pérdidas en el transformador, es por eso que primero hay que estimar las pérdidas. La potencia en la salida es siempre menor a la potencia en la entrada. La diferencia de potencia es disipada en calor, debido a las pérdidas en el núcleo y el bobinado, por medio de las áreas expuestas del transformador. Sin embargo, es difícil predecir el aumento de temperatura con precisión. Un enfoque consiste en incluir juntas las pérdidas en el núcleo y bobinado, y asumir que la energía térmica es disipada de forma uniforme en la superficie del núcleo y el bobinado para cualquier temperatura ambiente. Esta suposición es válida debido a que la mayor parte de la superficie del transformador es núcleo de ferrite, que tiene una conductividad térmica muy baja, y la menor parte es del bobinado. Considerando estas suposiciones, se obtiene la fórmula experimental expresada en la ecuación 2.56:

$$\Delta T = [\frac{P_T(mW)}{S(cm^2)}]^{0.833}\ ^\circ C \qquad (2.56)$$

Donde P_T es la potencia disipada en forma de calor debido a las pérdidas tanto en el núcleo de ferrite como en el bobinado, S es la superficie expuesta del transformador, incluyendo núcleo y bobinados, y ΔT es el aumento de temperatura en el transformador en °C.

Por diseño se pretende obtener un aumento de temperatura de aproximadamente 30°C. Entonces calculando el área expuesta para un núcleo RM 10, se utiliza la ecuación 2.56 para determinar las pérdidas que permitan obtener el aumento de temperatura deseado. En la figura 2.37 se puede observar el esquema de un núcleo RM 10.

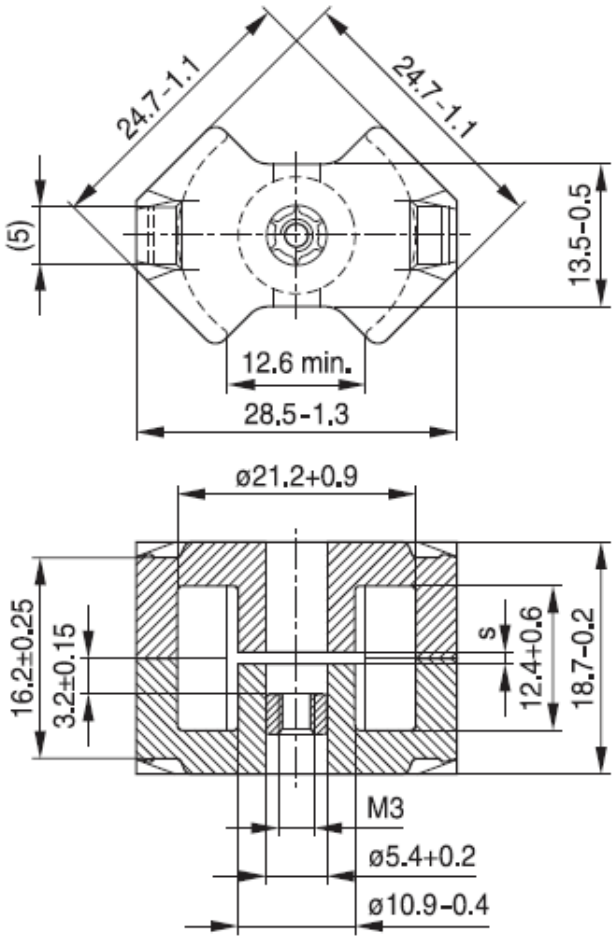


Figura 2.37: Esquema del núcleo RM 10

En la ecuación 2.57 se muestra el cálculo aproximado para el área de la superficie expuesta del transformador con el núcleo utilizado:

$$S = (13.5mm * 28.5mm) + 2 * (16.2mm * 13.5mm) + 2 * (16.2mm * 28.5mm) =$$

$$= 1745.55mm^2 = 17.45555cm^2 \qquad (2.57)$$

Este cálculo incluye toda superficie que disipe calor, es decir, las cuatro caras laterales y la cara superior. La cara inferior no se incluye, ya que al estar contra la PCB no disipa calor.

Ahora entramos en la ecuación 2.56 con la superficie expuesta del transformador y el aumento de temperatura deseada para obtener las pérdidas en el transformador:

$$P_T = (\Delta T * S^{0.833})^{1/0.833} \cong 1035.6mW$$

Esta es la suma de las pérdidas producto del núcleo y el bobinado. Pero se desconoce cuánto de las pérdidas totales asignarle al núcleo y cuanto al cobre en el bobinado. En la figura 2.38 se observa un gráfico de las potencias disipadas por el núcleo, el bobinado y las totales, en función de la densidad de flujo magnético pico, ΔB.

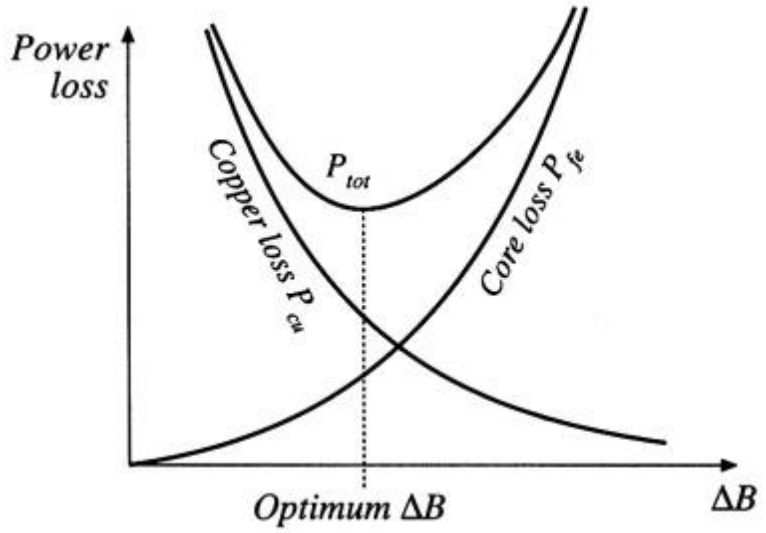


Figura 2.38: Potencias disipadas en el transformador en función de ΔB.

El grafico muestra como las pérdidas en el núcleo aumentan a medida que incrementa la densidad de flujo pico, esto es debido a que se hacen más significativas las pérdidas por histéresis. En cambio, las pérdidas en el cobre del bobinado disminuyen con el aumento de ΔB, ya que es necesario un menor número de vueltas en el bobinado. Existe un ΔB óptimo, donde la potencia disipada en el transformador es mínima. Como se puede observar en el gráfico, en este punto las pérdidas en el núcleo y el bobinado son aproximadamente iguales. Es por esto último, que se asigna la mitad de las pérdidas en el núcleo y la otra mitad en el bobinado:

$$P_T = P_{CU} + P_{CORE} \cong 517.8mW + 517.8mW \cong 1035.6mW$$

El fabricante especifica las pérdidas por volumen en el núcleo en función de la densidad de flujo magnético en alterna. El núcleo RM 10 tiene un volumen, V_E, de 4310mm^3. Por lo tanto las pérdidas por volumen en el núcleo son:

$$P_V = \frac{P_{CORE}}{V_E} = \frac{517.8mW}{4310mm^3} \cong 0.12\,\frac{mW}{mm^3} \cong 120\,\frac{KW}{m^3}$$

En la figura 2.39 se observa el grafico de las pérdidas por unidad de volumen dado por el fabricante. Se marcó con rojo el ΔB que a 100KHZ y a una temperatura ambiente de 25°C, producen las pérdidas por volumen pretendidas.

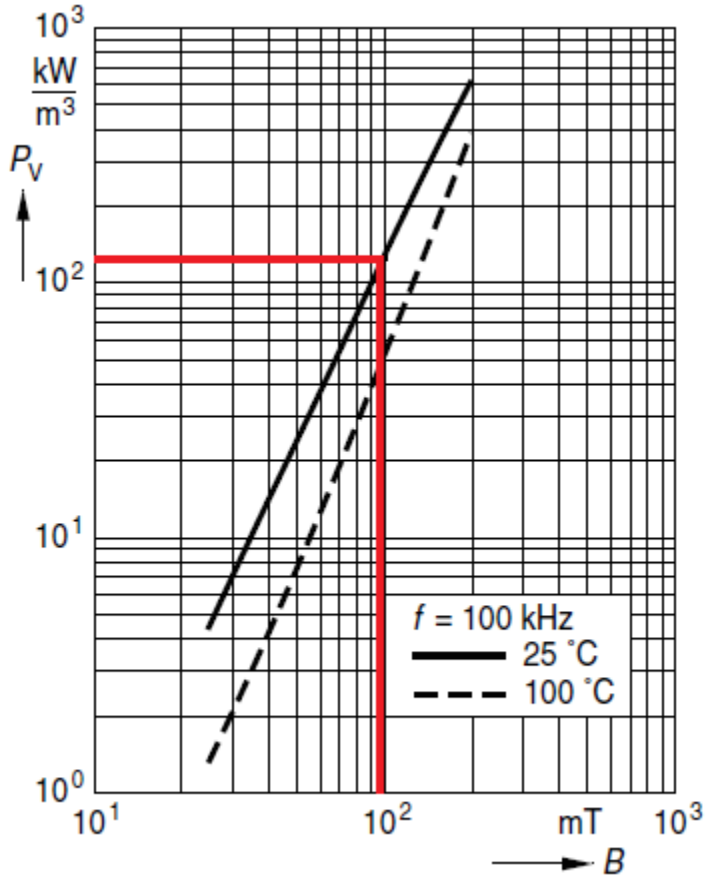


Figura 2.39: P_V en función de ΔB.

Observando el grafico se ve que para obtener la potencia de pérdida en el núcleo deseada, se debe operar con una densidad de flujo magnético en alterna de aproximadamente 100mT. Ahora se plantea la ecuación 2.58, que nos permite obtener el número de vueltas en el primario:

$$N_P = \frac{L_P * I_P}{A_E * \Delta B} = \frac{3.92mH * 0.2576A}{9.8 * 10^{-5} m^2 * 100mT} = 103.04 \qquad (2.58)$$

Donde A_E es el área efectiva del núcleo. Recordando que la relación de vueltas entre primario y secundario es 38, se decide tomar tres vueltas para el bobinado secundario, resultando 114 vueltas para el bobinado primario. Tomar este número ligeramente superior para el número de vueltas, favorece a reducir las pérdidas en el núcleo y además favorece a la aislación, debido a que se tiene una tensión menor por cada vuelta. Ahora recalculando la densidad de flujo magnético mediante la ecuación 2.58 se obtiene:

$$\Delta B = \frac{L_P * I_P}{A_E * N_P} = \frac{3.92mH * 0.2576A}{9.8 * 10^{-5} m^2 * 114} \cong 90.4mT$$

El siguiente paso consiste en determinar el gap en el núcleo para obtener el valor de inductancia deseado. Sin gap el valor aproximado de la inductancia del primario seria el dado por la ecuación 2.59:

$$L_P = A_L * {N_P}^2 = 4200nH * 114^2 \cong 54.6mH \quad (2.59)$$

Donde A_L es el factor de inductancia dado por el fabricante.

Con el agregado de un gap se puede obtener un valor de inductancia menor. Para determinar el valor se recurre al circuito magnético de la figura 2.40.

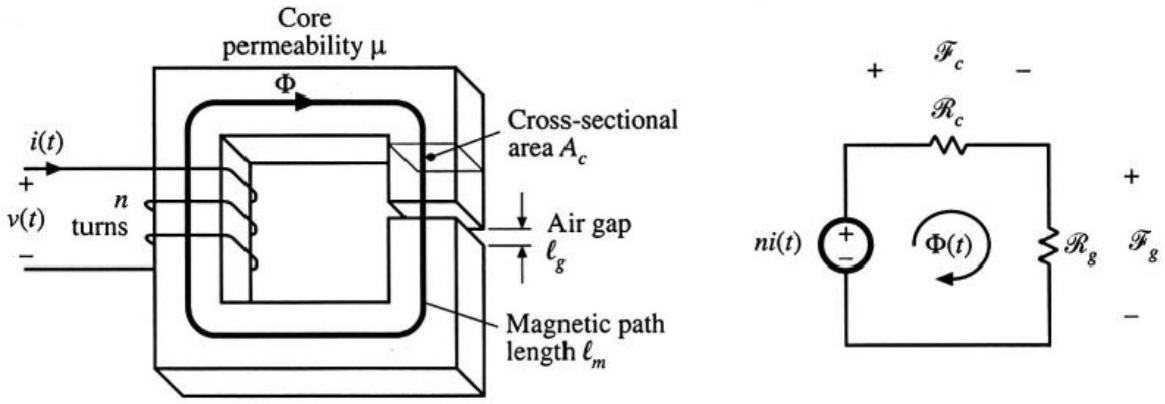


Figura 2.40: Circuito magnético de un inductor con gap.

Se plantea la ecuación 2.60 para resolver el circuito y la ley de Faraday en la ecuación 2.61:

$$n * i(t) = \Phi(t) * \left(\Re_c + \Re_g\right) \quad (2.60)$$

$$v(t) = n\frac{d\Phi}{dt} \quad (2.61)$$

Si ahora se despeja el flujo magnético de la ecuación 2.60 y se lo introduce en la 2.61, resulta la ecuación 2.62:

$$v(t) = \frac{n^2}{\Re_c + \Re_g}\frac{di(t)}{dt} \quad (2.62)$$

Esta ecuación deja en claro cuál es la expresión para la inductancia, mostrada en la ecuación 2.63:

$$L = \frac{n^2}{\Re_c + \Re_g} \quad (2.63)$$

Las ecuaciones 2.64 y 2.65 muestran las expresiones para las reluctancias del núcleo y el gap respectivamente:

$$\Re_c = \frac{\ell_c}{\mu_c * A_C} \quad (2.64)$$

$$\Re_g = \frac{\ell_g}{\mu_g * A_C} \quad (2.65)$$

Introduciendo estas expresiones en la ecuación 2.63 y despejando la longitud del gap, se obtiene la ecuación 2.66:

$$\ell_g = \left(\frac{A_L * {N_P}^2}{L_P} - 1\right) * \frac{\ell_c}{\mu_r} = \left(\frac{4200nH * 114^2}{3.92mH} - 1\right) * \frac{44mm}{1500} \cong 0.38mm \qquad (2.66)$$

Donde A_L es el factor de inductancia, que tiene en cuenta las características magnéticas del material. Este y el resto de los valores son proporcionados por el fabricante en la hoja de datos del núcleo.

Se ve como a medida que aumenta el gap, aumenta la reluctancia del circuito magnético y por lo tanto disminuye el valor de la inductancia. Si el valor de la reluctancia del gap es mayor a la del núcleo, se logra insensibilizar a L de las variaciones de la permeabilidad. La permeabilidad es difícil de especificar, ya que esta depende de factores como la temperatura y el punto de operación en el núcleo. Con lo cual, se logra un control mucho mayor sobre el valor de L. Sin embargo, debido a la dispersión en el flujo magnético en el entorno del gap, la sección efectiva del núcleo aumenta. Considerando esto último y sumando la dispersión en los parámetros otorgados por el fabricante, seguramente el valor calculado en la ecuación 2.66 tendrá que ser ligeramente ajustado.

Otra ventaja de la utilización del gap es que se permiten mayores valores en la corriente del bobinado sin que haya saturación. En la figura 2.41 se muestra como las características B vs H cambian con la inclusión del gap y el consecuente aumento de la reluctancia.

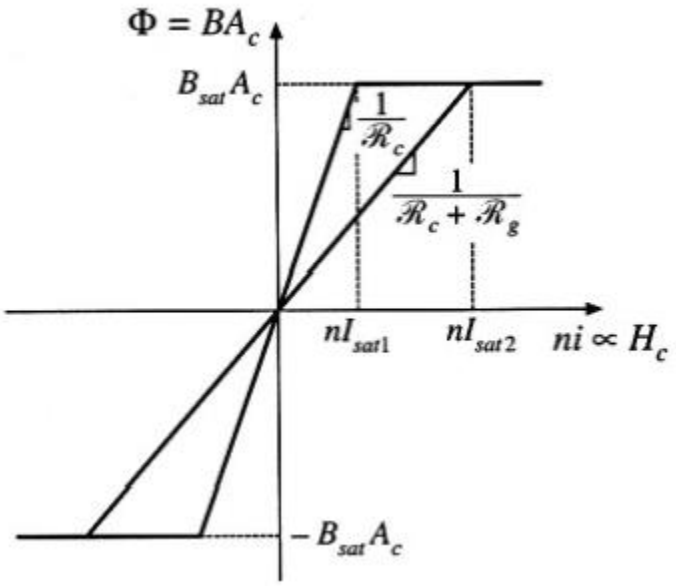


Figura 2.41: B vs H con y sin gap.

Lo siguiente es definir el número de vueltas para cada uno de los bobinados restantes. Empezamos con el bobinado auxiliar, cuyo número de vueltas está dado por la ecuación 2.67:

$$N_A = N_{AS} * N_S = 2 * 3 = 6 \qquad (2.67)$$

Luego se determina el número de vueltas para los bobinados $\pm V_{CC}$. Este valor lo impone la tensión que se pretende obtener en estos bobinados, determinada por la ecuación 2.68:

$$\pm V_{CC} = (V_F(7A) + V_0) * \frac{N_{V_{CC}}}{3} - V_F(1A) \cong 5.63V * \frac{8}{3} - 0.43V \cong 14.6V \qquad (2.68)$$

Adoptando un valor de ocho vueltas para ambos bobinados se obtiene un valor de tensión cercano a los 15V pretendidos.

El último paso del diseño del transformador consiste en definir la sección del alambre de cobre a utilizar en cada bobinado. La sección depende de la corriente RMS en cada bobinado y la densidad de corriente, J, que se pretenda. Esta relación es mostrada en la ecuación 2.69:

$$S = \frac{I}{J} \qquad (2.69)$$

Cuanto mayor sea J, mayor será la potencia disipada en el alambre. Por lo tanto debe adoptarse una densidad de corriente que permita que se disipe poca potencia en el bobinado. Se decidió tomar J = 3A/m^2. Sin embargo, en el caso de grandes corrientes RMS (como es el caso en el secundario de la topología Flyback) la sección del alambre no se puede aumentar de forma indefinida. En alta frecuencia la corriente que pasa por un alambre deja de circular por el centro y lo hace solo cerca de la superficie del conductor, como se muestra en la figura 2.42.

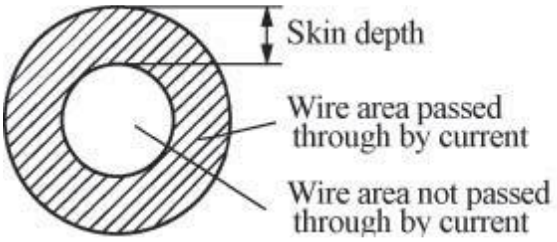


Figura 2.42: Efecto skin.

Por lo tanto se disminuye significativamente la sección efectiva del conductor, lo que deriva en el aumento de la resistencia en alterna, la disminución de la densidad de corriente y el aumento de temperatura en el conductor, producto de la potencia disipada. En la ecuación 2.70 se expresa la profundidad skin:

$$\delta = \sqrt{\frac{\rho}{\pi f \mu}} \qquad (2.70)$$

Donde ρ es la resistividad del conductor, f es la frecuencia y μ la permeabilidad. Para el cobre a una temperatura de 25°C y una frecuencia de 100KHZ resulta como la ecuación 2.71:

$$\delta = \frac{6.62}{\sqrt{f}} = \frac{6.62}{\sqrt{100KHZ}} \cong 0.21mm \qquad (2.71)$$

Teniendo esto en cuenta, para que la resistencia de alterna con respecto a la de continua no se vea modificada, se debe tomar un diámetro máximo para el conductor de dos veces δ:

$$d_{MAX} = 2 * \delta = 2 * 0.21mm = 0.42mm$$

En caso de que el diámetro del alambre para mantener la densidad de corriente sea mayor a d_{MAX}, se deberá buscar una alternativa.

Ahora calcularemos la corriente RMS en el primario para determinar la sección necesaria en el alambre de cobre. La corriente RMS en el primario está dada por la ecuación 2.72:

$$I_{P_{RMS}} = \sqrt{\frac{1}{T}\int_0^T i_P(t)^2 dt} = \sqrt{\frac{1}{T}\int_0^{T_{ON}} \left(\frac{V_{IN}*t}{L}\right)^2 dt} = I_{PP}\sqrt{\frac{D_{ON(MAX)}}{3}} = 0.2576A * \sqrt{\frac{0.481}{3}} \cong 103.2mA \qquad (2.72)$$

Utilizando la ecuación 2.69 con esta corriente RMS se obtiene:

$$S_P = \frac{I_{P_{RMS}}}{J} = \frac{103.2mA}{3\frac{A}{mm^2}} = 0.0344mm^2$$

Ahora utilizando la relación entre el diámetro y la sección en la ecuación 2.73, se obtiene el diámetro del alambre:

$$d_P = \sqrt{\frac{4*S_P}{\pi}} = \sqrt{\frac{4*0.0.344mm^2}{\pi}} \cong 0.21mm \qquad (2.73)$$

El diámetro es inferior a dos veces la profundidad del efecto skin, por lo tanto no hay inconvenientes y se adopta un diámetro comercial de 0.25mm en el alambre de cobre para bobinar el primario.

A continuación se repite el proceso, pero ahora para el secundario principal. Primero determinaremos la corriente pico en el bobinado secundario mediante la ecuación 2.74:

$$I_{SP} = I_{PP} * N_{PS} * \frac{P_S}{P_T} = 0.2576A * 38 * \frac{1.5A*5V}{11W} \cong 6.68A \qquad (2.74)$$

Recordemos que en la sección anterior se estableció que potencia manejara cada bobinado en la salida, de ahí surge la fracción que aparece en esta ecuación. Ahora si calculamos la corriente RMS usando nuevamente la ecuación 2.72:

$$I_{S_{RMS}} = I_{SP}\sqrt{\frac{D_{MAG(MAX)}}{3}} = 6.68A * \sqrt{\frac{0.425}{3}} \cong 2.51A$$

Luego con la ecuación 2.69 determinamos la sección del alambre para este bobinado:

$$S_S = \frac{I_{S_{RMS}}}{J} = \frac{2.51A}{3\frac{A}{mm^2}} = 0.8366mm^2$$

Y a continuación se calcula el diámetro con la ecuación 2.73:

$$d_S = \sqrt{\frac{4*S_S}{\pi}} = \sqrt{\frac{4*0.8366mm^2}{\pi}} \cong 1.03mm$$

Este valor supera a d_{MAX}, razón por la cual no se bobinó el secundario principal con un alambre de cobre. Para evitar la reducción de J producto del efecto skin, se decidió utilizar una lámina de cobre, cuyo corte transversal se puede ver en la figura 2.43.

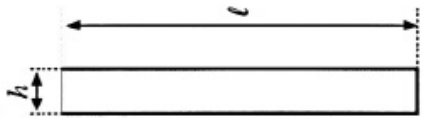


Figura 2.43: Lámina de cobre.

Si el espesor h de la lámina es menor o igual a d_{MAX}, no se reduce la sección efectiva del conductor. A su vez el ancho l debe ser tal de mantener S_S. Se adoptó una lámina de 0.4mm de espesor y se cortó con un ancho de aproximadamente 2mm, resultando una sección cercana a 0.8mm^2. El ancho seleccionado permite acomodar en una sola capa el bobinado secundario, como se verá más adelante en el esquema del transformador.

Continuamos con el bobinado $+V_{CC}$. Primero calculando la corriente pico mediante la ecuación 2.75:

$$I_{V_{CC}P} = I_{PP} * \frac{N_P}{N_{V_{CC}}} * \frac{P_{+V_{CC}}}{P_T} = 0.2576A * \frac{8}{3} * \frac{0.2A*14.6V}{11W} \cong 0.975A \qquad (2.75)$$

Con la ecuación 2.72 ahora calculamos la corriente RMS:

$$I_{V_{CC_{RMS}}} = I_{V_{CC}P}\sqrt{\frac{D_{MAG(MAX)}}{3}} = 0.975A * \sqrt{\frac{0.425}{3}} \cong 0.366A$$

Luego determinamos la sección del alambre con la 2.69:

$$S_{V_{CC}} = \frac{I_{V_{CC_{RMS}}}}{J} = \frac{0.366A}{3\frac{A}{mm^2}} = 0.122mm^2$$

Finalmente calculamos el diámetro requerido con la ecuación 2.73:

$$d_{V_{CC}} = \sqrt{\frac{4*S_{V_{CC}}}{\pi}} = \sqrt{\frac{4*0.122mm^2}{\pi}} \cong 0.4mm$$

Este diámetro no presenta inconvenientes con el efecto skin, por lo que se adopta un diámetro comercial de 0.4mm para bobinar.

Como tanto el bobinado auxiliar como el de $-V_{CC}$ manejan muy poca potencia, se optó por bobinar con el diámetro mínimo, que coincide con el del primario, de 0.25mm.

Hay una última consideración a hacer respecto al diseño del transformador y está relacionado con efecto de proximidad. Cuando existen conductores muy cerca uno del otro y por uno circula una corriente, se genera un flujo que intenta penetrar el otro conductor, pero se induce una corriente en el segundo conductor en un intento de oponerse al flujo generado por el primero. Este efecto es mostrado para un transformador, con tres capas en el primario, tres capas en el secundario y con la profundidad del efecto skin mucho menor que el diámetro de los conductores, en la figura 2.44.

Si los conductores están lo suficientemente cerca, la corriente inducida en el segundo conductor será igual en modulo y opuesta a la que circula por el primero. Pero como los conductores

están en serie, debe circular la misma corriente neta por ambos, con lo cual en el otro lado del segundo conductor circula una corriente dos veces superior a la que circula por el primero. Esto se repite en el resto de los conductores adyacentes en cada capa como lo muestra la figura 2.44. También se incluye un gráfico con la distribución de la densidad de corriente en cada capa de los bobinados.

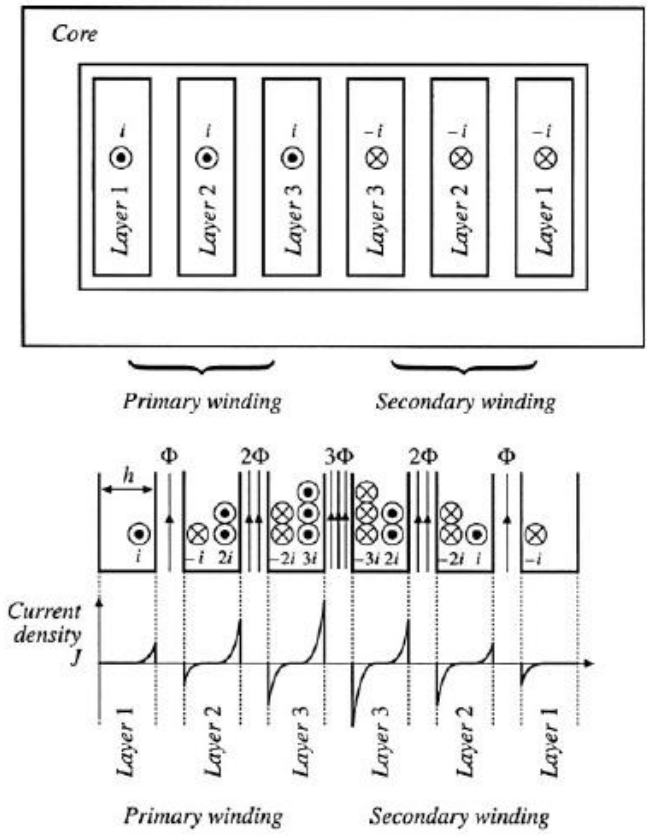


Figura 2.44: Efecto proximidad en un transformador.

Este efecto aumenta significativamente la resistencia de alterna en las capas interiores, provocando una mayor disipación de potencia. En la figura 2.45 se presenta una gráfico de la relación entre la potencia disipada por efecto skin y proximidad, contra el cociente entre el espesor del conductor y la profundidad de penetración, con el número de capas como parámetro. Con las precauciones tomadas contra el efecto skin, se consiguió obtener el cociente φ menor a 1. Y para combatir el efecto proximidad se buscó bobinar el secundario en una sola capa y además utilizar la técnica de interleaving.

Con esta técnica se reducen de forma significativa las pérdidas generadas por el efecto de proximidad. El interleaving consiste en alternar los bobinados primario y secundario.

Haciendo esto se consigue el efecto que tendría reducir el número de capas en los bobinados. Para ilustrar esto, en la figura 2.46 se muestra el diagrama de la distribución de la fuerza magneto motriz, MMF, para un transformador bobinado de forma convencional. La MMF está relacionada con la distribución de corriente que se mostró en la figura 2.44. Ahora miremos la figura 2.47, donde se utiliza interleaving en el bobinado, para ver cómo cambia la distribución de la fuerza magneto motriz.

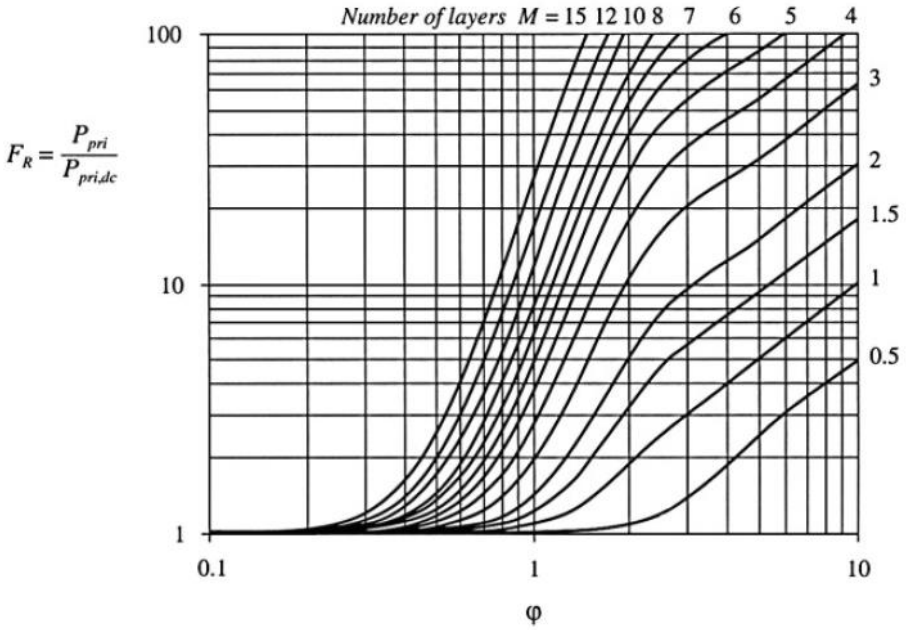


Figura 2.45: Aumento en la potencia disipada por el efecto skin y proximidad.

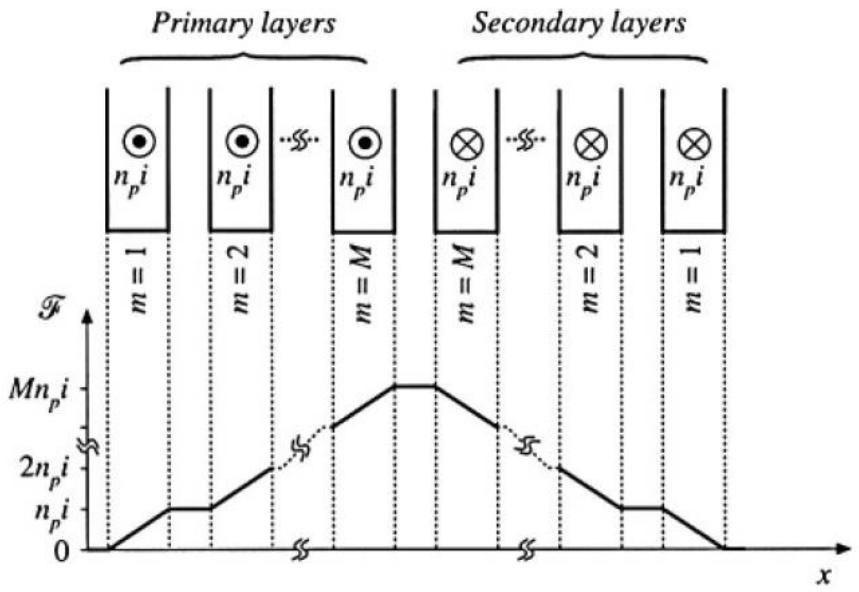


Figura 2.46: Distribución de MMF sin interleaving.

Este cambio en el diagrama de MMF produce el efecto de reducir la cantidad de capas del bobinado, lo que mejora notablemente la disipación de potencia como se muestro en el gráfico de la figura 2.45.

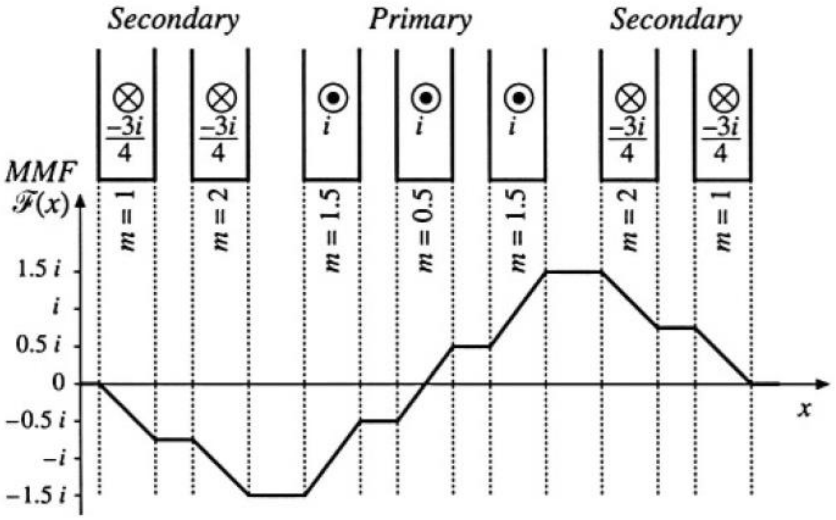


Figura 2.47: Distribución MMF con interleaving.

Haciendo estas consideraciones, se definió utilizar interleaving en el bobinado. Se dividió en dos el bobinado primario, con el secundario principal en el medio. A continuación se muestran el carrete RM 10 y el esquema de como quedaron los bobinados en el transformador en las figuras 2.48 y 2.49 respectivamente.

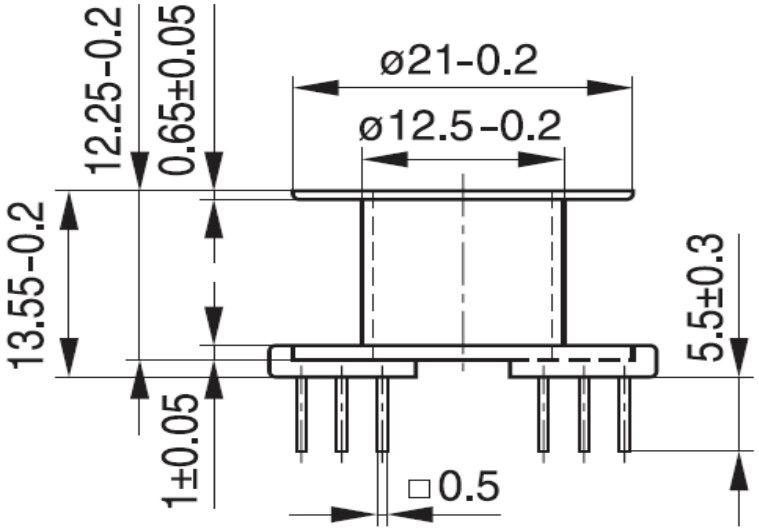


Figura 2.48: Carrete RM 10.

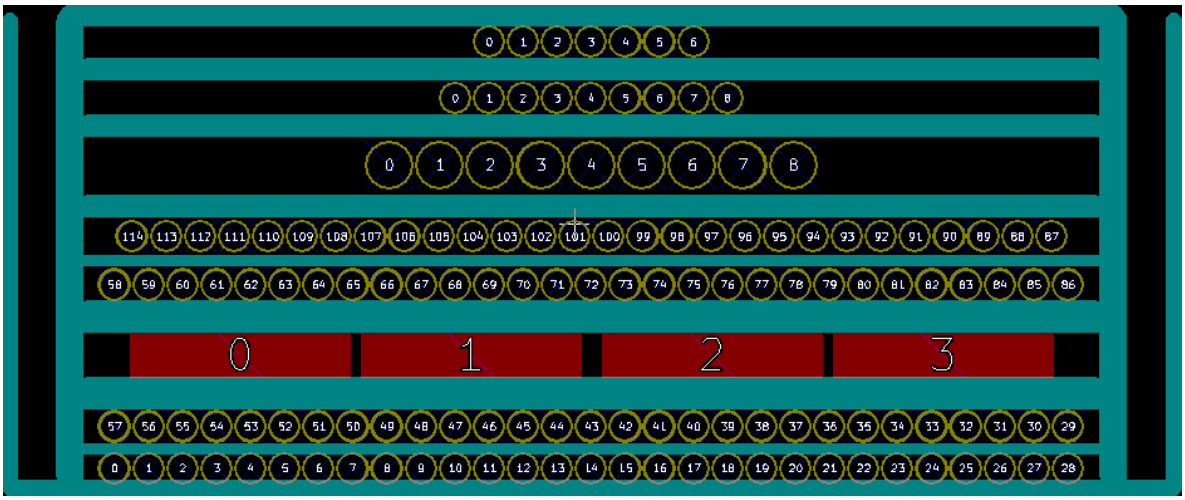


Figura 2.49: Esquema final del transformador.

Por cuestiones de seguridad, puede verse un margen de seguridad de aproximadamente 1mm de cada lado. Las líneas verdes representan a la cinta de mylar que se utilizó para brindar aislación. Entre capas de un mismo bobinado se dio una sola vuelta, mientras que para aislar entre el primario y secundario principal se dieron tres vueltas. Luego para separar entre el resto de bobinados y después del último bobinado, se dieron dos vueltas. En conclusión, a pesar de no haberse aprovechado de la mejor manera el tamaño del transformador, debido a la dificultad que presenta el tener que bobinar de forma manual y la inexperiencia, el resultado final es positivo.

2.9 Filtro EMI

Las interferencias electromagnéticas (EMI) son el acoplamiento no deseado de señales de un circuito a otro. Se clasifican en interferencias por conducción y por radiación. El primer tipo de interferencia se propaga por medio de una conexión común a los dos circuitos, mientras que el segundo tipo es mediante radiación electromagnética.

En la topología Flyback, la tensión en el drain es el punto donde se generan los mayores inconvenientes. Ya que en V_D además de los altos valores de tensión, también tiene una dV/dt muy elevada. A partir de esta tensión y a través de las capacidades parásitas en el circuito, circulan corrientes de alta frecuencia que van hacia la carga y hacia la línea.

Existen dos tipos de interferencia en la línea. Una es de modo común, CM, donde la corriente circula tanto por el neutro como por fase y retorna por tierra. El otro es de modo diferencial, DM, donde la corriente circula por fase y retorna por neutro.

En la figura 2.50 se muestra el circuito con todas las capacidades parásitas que producen interferencia de modo común por conducción. Donde C_{S1} es la capacidad entre el drain del MOSFET y tierra, C_{OSS} es la capacidad de salida del MOSFET, C_{S2} es entre el circuito secundario y tierra, de C_{BD1} a C_{BD4} son las capacidades en cada uno de los diodos del puente rectificador, C_{AC} es la capacidad en

la línea y por ultimo de C_{W1} a C_{W6} son las capacidades en el transformador. Además se incluyen el ESR y ESL de la capacidad de entrada.

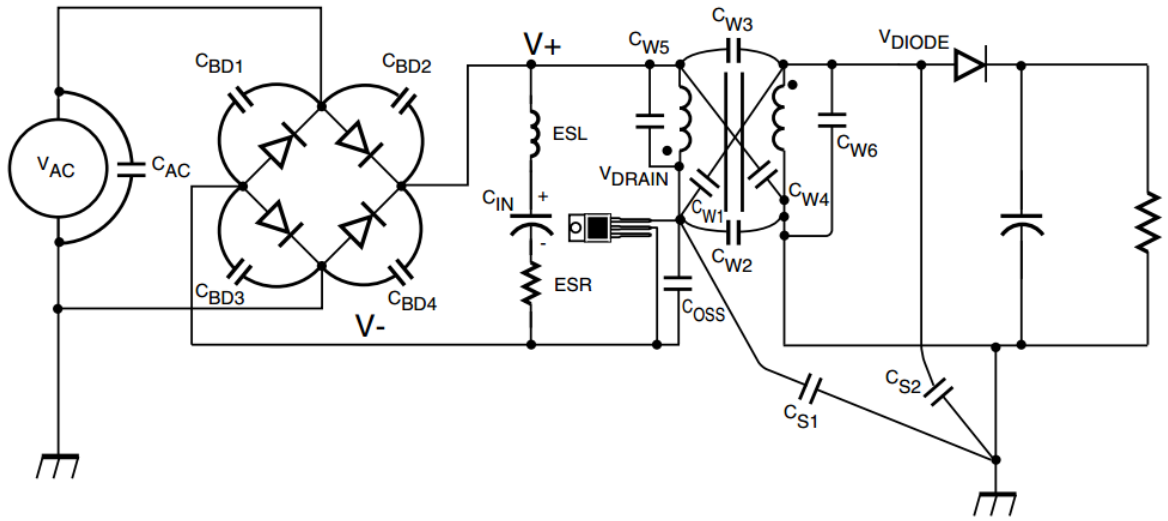


Figura 2.50: SMPS con todas las capacidades parásitas que producen EMI de CM por conducción.

La tensión en el drain produce corrientes a través de las capacidades C_{0SS}, C_{W2}, C_{W1}, C_{S1} y C_{W5}, que poseen un valor dado por la ecuación 2.76:

$$i_C = C\frac{dV_D}{dt} \qquad (2.76)$$

Esta expresión deja en claro porque la tensión en el drain del MOSFET es el punto más crítico, debido tanto a la elevada tensión en este punto como a los tiempos de encendido y apagado tan rápidos en la llave. Generalmente, como es el caso del controlador UCC28740, los circuitos de drive del MOSFET limitan la velocidad de encendido, para proteger a la misma llave y también para reducir estas corrientes. Obviamente reduciendo la capacidad parásita en los componentes, se reduce también la magnitud de esta corriente. Mediante el layout se pueden reducir las capacidades distribuidas, reduciendo las áreas creadas por los lazos de corriente.

Estas corrientes en su camino de retorno al nodo V_D, circulan por el neutro y la fase de la tensión de línea, generando interferencias. Por este motivo es necesario añadir un filtro de modo común. En la figura 2.51 se puede observar el filtro de modo común, que consiste en un inductor de modo común en las líneas, un capacitor entre fase y tierra y otro capacitor entre neutro y tierra. Las corrientes en las capacidades parásitas, generadas en V_D, generan una tensión en el neutro y la fase mediante los capacitores del filtro. El inductor de modo común y la impedancia de la línea forman un divisor de tensión, para la tensión mencionada anteriormente. Por lo tanto, para obtener una baja interferencia de modo común, los capacitores del filtro y el inductor deben ser del mayor valor posible. El nivel de interferencia permitido esta estandarizado según la aplicación. En la figura 2.52 se encuentra la curva en dBµV que se debe cumplir para ISM (Industrial, Scientific and Medical).

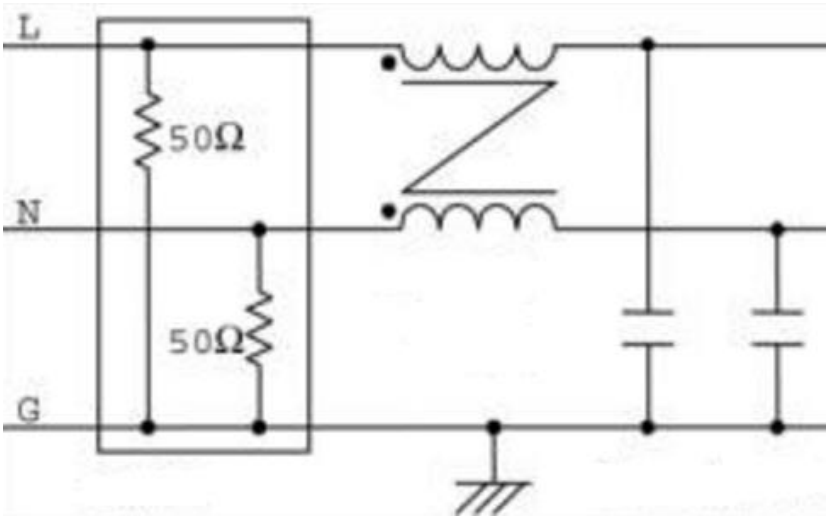


Figura 2.51: Filtro de CM.

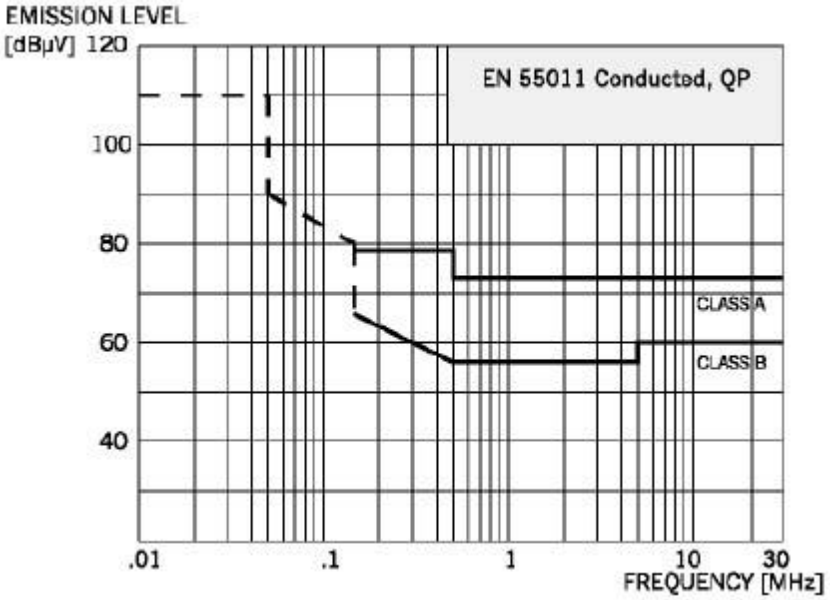


Figura 2.52: Niveles de interferencia EMI permitidos para ISM.

En nuestro caso se siguió la curva indicada como Clase A, que es para aplicaciones industriales. Otro límite estandarizado es el impuesto para la corriente entre fase y tierra o entre neutro y tierra, de esta forma también se impone limite a los capacitores del filtro. En la ecuación 2.77 se calcula el límite para estos capacitores:

$$I = 2\pi * f * 2C_{CM} * \frac{V_{AC}}{2} \rightarrow 1mA = 2\pi * 50HZ * C_{CM} * 220V_{RMS} \qquad (2.77)$$

Para esta aplicación se limitó esta corriente a aproximadamente 1mA, dando como resultado una capacidad máxima de 14.5nF para cada uno de estos capacitores. Finalmente se adoptó el valor comercial de 10nF.

En cuanto al valor de L_{CM} se eligió un valor de 10mH, ya que este valor es el que nos permitía cumplir con lo impuesto por la curva de la figura 2.52. En la sección de las simulaciones se mostrarán los resultados.

Con respecto a las interferencias de modo diferencial, estas son causadas principalmente por la corriente en el bobinado primario. En la figura 2.53 se puede observar tanto al circuito que muestra de donde proviene la interferencia y el modelo equivalente.

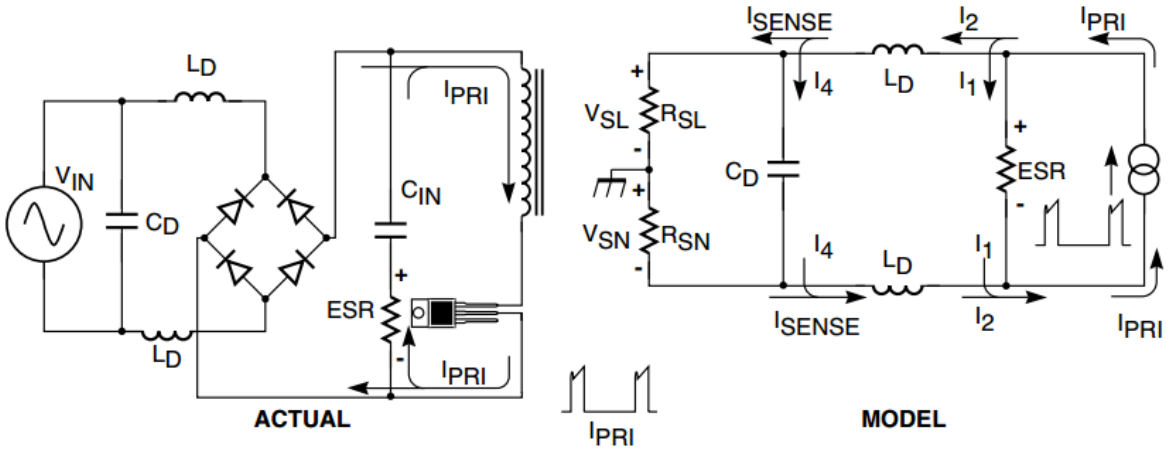


Figura 2.53: Circuito y modelo equivalente para EMI por interferencias por DM.

En el modelo se plantea un generador de corriente que representa la corriente en el primario, a alta frecuencia se reemplaza el capacitor de entrada por su ESR, se considera que el puente rectificador está conduciendo y se lo reemplaza por un cortocircuito y se incluye las impedancias de 50Ω de cada línea. La mayoría de la corriente del primario circula por el ESR del capacitor de entrada, ya que la impedancia que presenta este es muy pequeña comparada con la del filtro. Por lo tanto se puede plantear un modelo aún más simplificado, presentado en la figura 2.54.

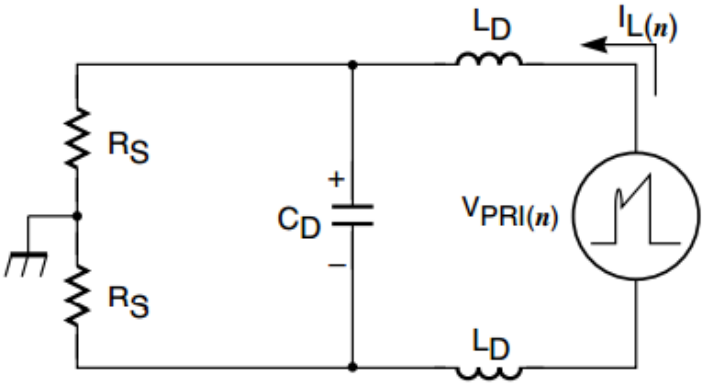


Figura 2.54: Modelo simplificado para DM.

Ahora simplemente queda una fuente de tensión, que es la tensión RMS generada por la corriente de primario RMS circulando por el ESR de C_{IN} y el filtro de modo diferencial, que actúa como filtro pasabajos para atenuar la tensión de interferencia en cada una de las líneas al nivel deseado.

Para el diseño de esta SMPS se optó por añadir un capacitor cerámico con muy bajo ESR en paralelo a C_{BULK}, para bajar el ESR total. Teniendo esto en cuenta, recordando que además la corriente RMS en el primario es relativamente pequeña (del orden de 100mA), se desprecia las interferencias de modo diferencial.

Tanto el filtro de modo común y el capacitor cerámico añadido son para combatir las interferencias en la línea, sin embargo, aún resta contrarrestar el EMI en la salida. Como se ha mostrado en el modelo para la generación de EMI de CM por conducción, existen capacidades parásitas entre los bobinados del transformador. Esta capacidad acopla primario y secundario e impide la aislación a alta frecuencia, además se conducen corrientes de modo común que interfieren usualmente en la masa del circuito de salida.

Para contrarrestar esto se debe reducir la capacidad parásita, lo que por razones constructivas lleva a aumentar la inductancia de pérdida, o bien añadir un filtro de modo común. Se optó por colocar una capacidad entre las masas del primario y secundario, como se muestra en la figura 2.55.

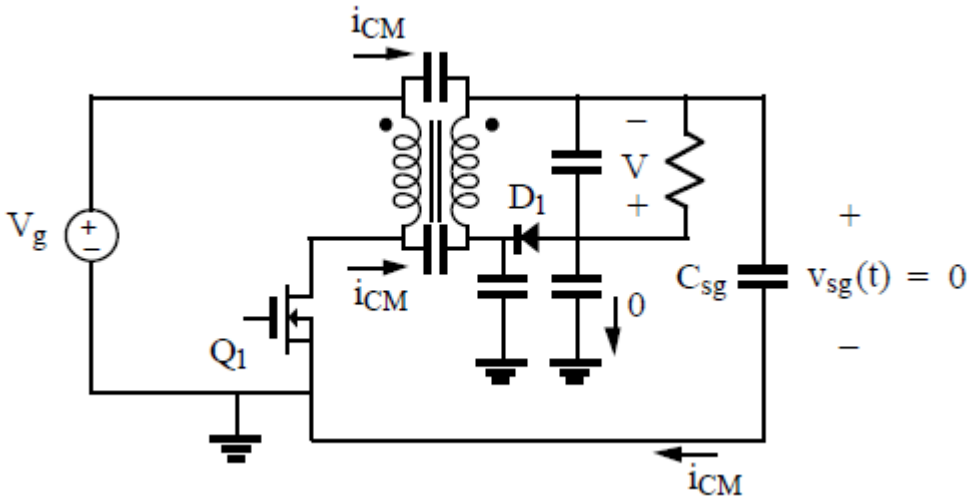


Figura 2.55: Capacitor para filtrar interferencias de modo común.

Si la capacidad añadida es mucho más grande que la capacidad parásita entre los bobinados del transformador, casi toda la corriente de modo común circulará por esta capacidad. Si además el capacitor es lo suficientemente grande, tendrá un ripple de tensión despreciable y desaparecerá el ruido de alta frecuencia en la referencia del circuito secundario.

Ahora resta definir el valor de esta capacidad, por lo que primero se debe estimar la capacidad entre bobinados del transformador. La ecuación 2.78 muestra el cálculo aproximado, partiendo de la ecuación del capacitor:

$$C_P = \frac{N*\varepsilon_0*\varepsilon_r*A}{d} \cong \frac{2*8.85*10^{-12}{}^{C^2}/_{Nm}*3*2\pi*({}^{12.5mm}/_{2}+2*0.25mm)*10.6mm}{0.3mm} \cong 80pF \qquad (2.78)$$

Observando las figuras 2.48 y 2.49 se pueden obtener los parámetros de esta ecuación. El factor N es dos debido a que el secundario está en la mitad del bobinado primario, con lo cual la capacidad se duplica. Luego la permitividad relativa del Mylar es tres y el grosor de la cinta se estima en 0.1mm, por eso al ser tres vueltas de cinta la distancia entre bobinados es de 0.3mm. Mientras que para el área además de tener en cuenta la geometría del carrete se sumó dos veces el diámetro del alambre del primario.

Como la capacidad añadida debe ser mucho mayor al valor indicado por el ecuación 2.78, se decide adoptar un valor de 470pF.

2.10 Protección en la entrada

Varistor

Como la entrada de la SMPS es la tensión de línea, y en la misma se producen transitorios de alta tensión, surge la necesidad de añadir una protección al circuito. Mediante un estudio estadístico, se ha estandarizado la probabilidad de ocurrencia de estos sobrepicos, la amplitud, la forma y la duración de estos pulsos. En la figura 2.56, se presenta una forma de onda estandarizada para estos sobrepicos. Donde se define la corriente pico, el tiempo t_1, de 8µS hasta que la corriente llega al 90% de su valor pico, y el tiempo t_2, de 20µS hasta que la corriente desciende al 50% de su valor pico.

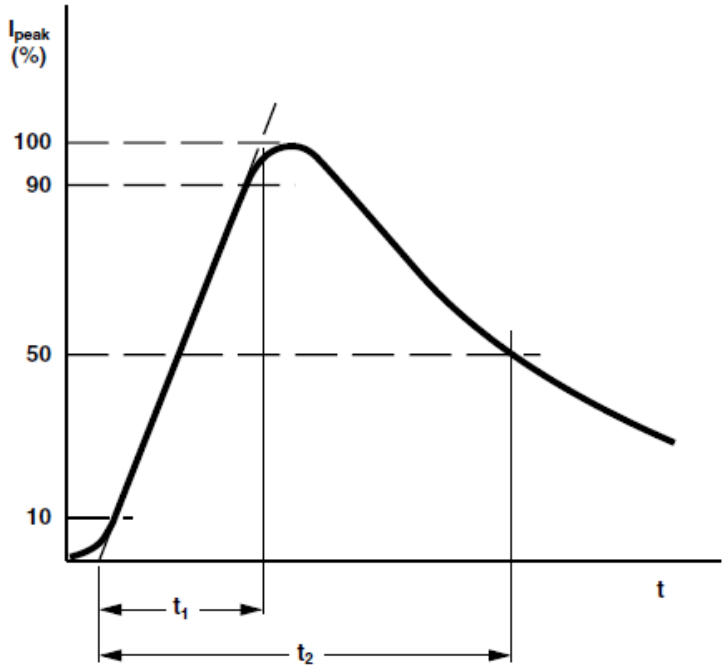


Figura 2.56: Forma de onda estandarizada para el sobrepico de tensión en la línea.

Como elemento de protección suele utilizarse un VDR (Voltage Dependent Resistor) o un TVS. El TVS presenta como ventaja frente al VDR, una menor tensión de clamping, una rápida respuesta y alta confiabilidad. Cuando son usados dentro de los parámetros dados por el fabricante, sus especificaciones no se ven afectadas por el paso del tiempo o la cantidad de veces que sirven como protección. Por otro lado, el VDR soporta mayor cantidad de energía, puede limitar sobrepicos tanto positivos como negativos y tiene un menor costo que el TVS. Teniendo en cuenta estas últimas razones, se optó por el uso de un VDR.

El VDR presenta una carga muy pequeña para pequeñas corrientes, mientras que para grandes corrientes se reduce su impedancia y limita la tensión a un valor seguro para proteger al circuito.

El primer paso en la selección del dispositivo es definir la tensión de alterna nominal. La línea entrega una tensión de alterna de $220V_{RMS}$ nominal, sin embargo, se prevé una variación del 15% para este valor, resultando $253V_{RMS}$. Por lo tanto, para tener un margen de seguridad se elige un varistor con tensión nominal de $275V_{RMS}$.

El siguiente paso es determinar, dependiendo de la corriente pico del pulso y la cantidad de veces que se pretenda que actué el VDR como protección, la energía que tiene que absorber el VDR. El fabricante ofrece un curva con la que se puede determinar la cantidad de pulsos, de que amplitud y de que duración, puede soportar el dispositivo. La figura 2.57 muestra la curva mencionada para el VDR seleccionado. Se puede ver que soporta 100 pulsos de corriente de más de 1KA con una duración de 20µS, o bien, 10 pulsos de 100A y duración de 1000µS.

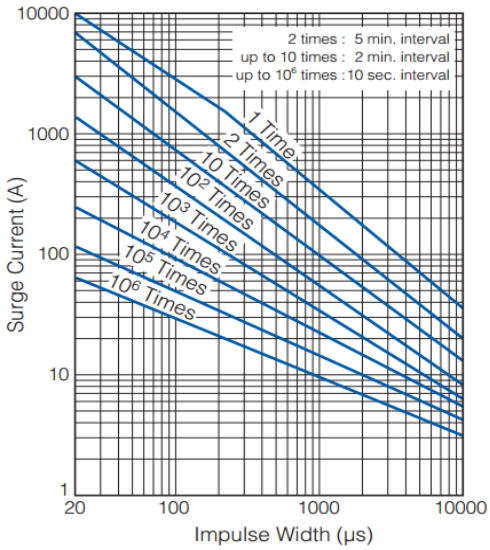


Figura 2.57: Relación entre la amplitud y duración del pico de corriente.

En la figura 2.58 se presenta la familia de curvas de log V vs log I para los distintos VDR ofrecidos por el fabricante, donde con el 431 se puede identificar a la curva del VDR utilizado en la SMPS. Este VDR ofrece limitar la tensión a menos de 900V para una corriente pico de 1KA.

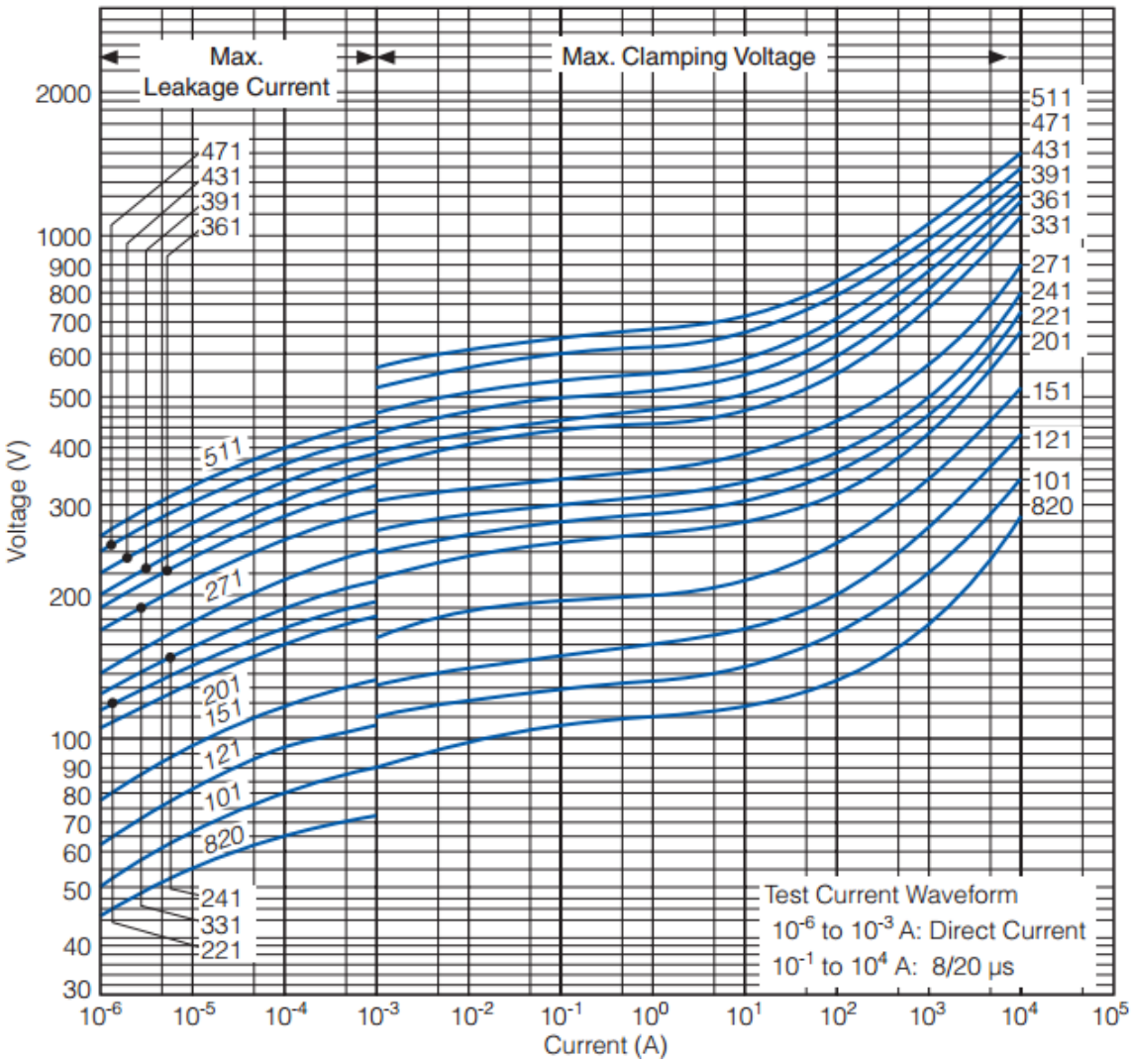


Figura 2.58: log V vs log I del VDR seleccionado.

Mediante la ecuación 2.79 se determina la energía absorbida por el varistor, que en este dispositivo tiene un máximo de 303J.

$$E = V_P * I_P * t_2 = 303J \qquad (2.79)$$

La energía que soporta el VDR es proporcional a su volumen. Por lo tanto, aquellos dispositivos que soporten una gran cantidad de energía son de gran volumen. Luego de absorber la energía máxima, el fabricante asegura que las especificaciones se mantendrán en un rango del ±10% de los valores nominales.

Fusible

Se utiliza un fuse para proteger al resto del circuito y evitar incendios, en caso de cortocircuito o sobrecarga en el circuito. El primero de los parámetros del fusible a definir es la

tensión nominal, de $250V_{RMS}$ en este caso. Luego la corriente nominal del fusible, corriente que debe ser mucho mayor a la corriente promedio máxima de la SMPS. La corriente promedio máxima está dada por la ecuación 2.80:

$$I_{MAX} = \frac{P_{IN}}{V_{MIN}} = \frac{13W}{210V} \cong 62mA \quad (2.80)$$

El siguiente parámetro es la máxima corriente que el fusible es capaz de interrumpir de forma segura, donde debe definirse la corriente máxima en caso de corto circuito. Por otro lado, está la integral de fundición, i^2t, que se puede observar en la figura 2.59.

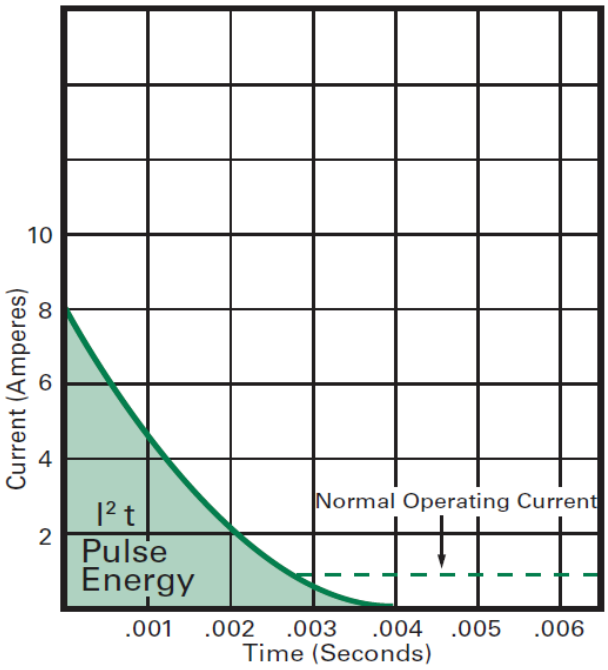


Figura 2.59: Curva i^2t.

Esta curva es una medida de la energía necesaria para fundir el fusible. De acuerdo a las características de esta curva se definen tres tipos fusibles, de acción rápida, media y lenta. Otra consideración con respecto a este parámetro, es tener en cuenta los grandes picos de corriente al momento de cargar inicialmente el capacitor de entrada. Se debe estimar la energía en el momento de este pico para asegurar que no se funda el fusible.

El último parámetro a tener en cuenta es la temperatura ambiente a la que opera el fusible. Cuanto mayor sea la temperatura ambiente, menor será la vida del fusible, mientras que se da el caso contrario cuando la temperatura ambiente es menor. El fabricante ofrece curvas de cómo debe afectarse a la corriente nominal del fusible con el cambio en la temperatura ambiente.

Para proteger este circuito, se optó por usar dos fusibles. Uno con una corriente nominal menor, que actuará antes y se coloca en un portafusible para su fácil reemplazo. Mientras que el otro es de mayor corriente nominal, que se coloca soldado en la placa.

2.11 Componentes

A continuación se hará un repaso por los componentes utilizados y las características tenidas en cuenta en el momento de la elección. Cabe mencionar que ante la elección de componentes con características similares, se ha dado prioridad a aquellos que ofrezcan un modelo SPICE para la simulación del circuito.

El circuito se ha divido en tres partes, siendo la mostrada en la figura 2.60 la primera.

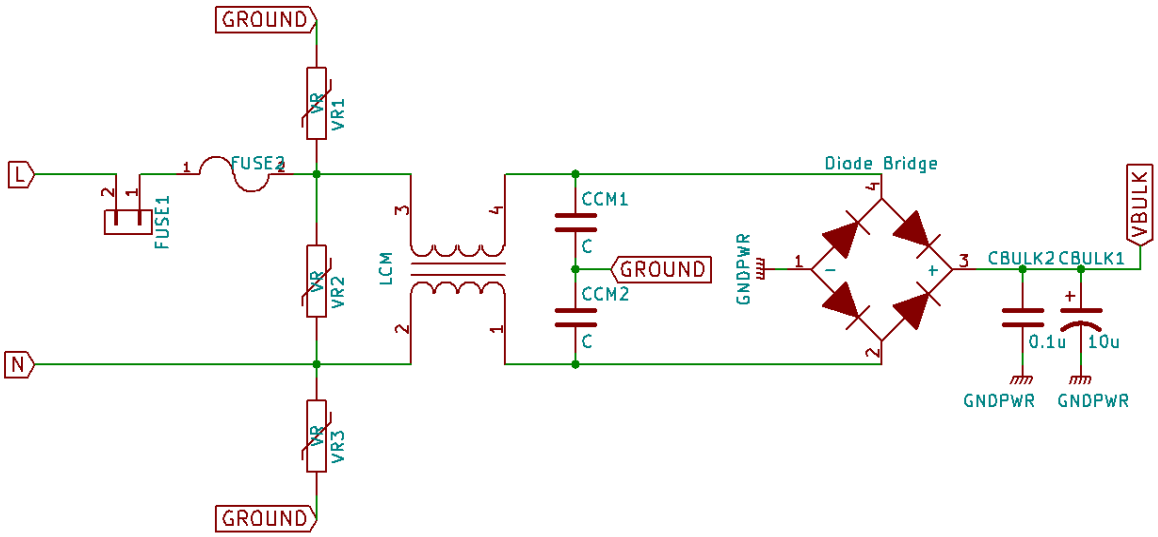


Figura 2.60: Primera parte del circuito de la SMPS.

Comenzamos con los dos fusibles, que van conectados en serie con la fase. Uno de montaje superficial soldado a la placa, de 4A de corriente nominal, 250V_{RMS} y de respuesta media. Otro con un porta fusible, para facilitar la reposición en caso de falla, de 2A de corriente nominal, 250V_{RMS} y de respuesta rápida.

A continuación aparecen los VDR para protección, se colocan tres, uno entre fase y tierra, otro en entre neutro y tierra y finalmente uno entre neutro y fase. Son de 275V_{RMS}, corriente pico no-repetitiva de 10KA y un máximo de 303J de energía.

Luego aparece el filtro de EMI, comenzando con el inductor de modo común de 10mH. Dentro de las alternativas disponibles se optó por aquella que tuviera mejor respuesta en frecuencia, que se determina mediante la curva otorgada por el fabricante, EPCOS en este caso. El inductor elegido presenta un comportamiento inductivo hasta una frecuencia cercana a 1MHZ. También se tuvo en cuenta la corriente máxima en L_{CM}, de 500mA, contra la corriente promedio máxima en la fuente de aproximadamente 62mA, dados por la ecuación 2.81.

En cuanto a los capacitores de modo común, son de 10nF, 250V_{RMS}, coeficiente de temperatura F y son de seguridad Y2. Se colocan dos, uno entre fase y tierra y el otro entre neutro y tierra.

Seguimos con el puente rectificador. De 800V, suficiente como soportar la tensión de clamping del varistor en caso de un sobrepico en la línea. Y 1A de corriente promedio, muy superior a la corriente promedio máxima de la SMPS.

Restan los capacitores C_{BULK1} y C_{BULK2}. El primero es electrolítico de 10µF y 450V. De entre todas las opciones disponibles se eligió el de mayor horas de funcionamiento, 10.000 Hrs, y el de mayor ripple de corriente, 120mA. Pero además se agregó un capacitor cerámico de 0.1µF, 630V, de bajo ESL y ESR. Esta última característica ayuda a mitigar las interferencias de modo diferencial en la línea.

En la figura 2.61 se muestra la segunda parte del circuito.

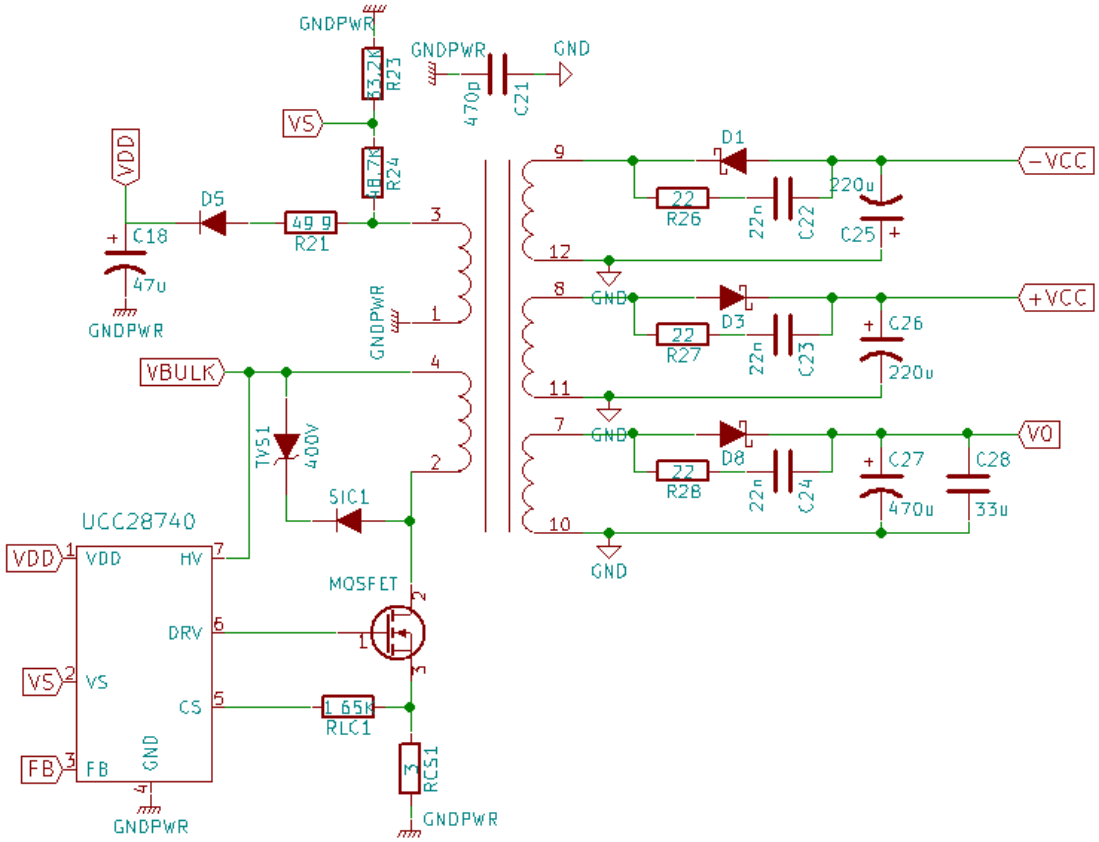


Figura 2.61: Segunda parte del circuito de la SMPS.

El controlador UCC28740 ya fue analizado en una de las secciones anteriores. Sin embargo el valor de la resistencia R_{LC} aún no se ha determinado. Esta resistencia es utilizada para proveer compensación de la línea, para eliminar los cambios en I_{PP} producto de las variaciones en V_{BULK} debido al retardo de un comparador interno del controlador y el del MOSFET. El valor de esta resistencia está dada por la ecuación 2.81, proporcionada por el fabricante:

$$R_{LC} = \frac{K_{LC} * R_{S1} * R_{CS} * t_D * N_{PA}}{L_P} = \frac{25 * 48.7K\Omega * 3\Omega * (50nS + 40nS) * 19}{3.92mH} \cong 1.6K\Omega \qquad (2.81)$$

Donde K_{LC} es una contante de conversión de corriente del UCC28740. Mientras que el tiempo de delay es la suma del retardo interno del controlador, de 50nS, y el tiempo de apagado del MOSFET, de 40nS. En esta ecuación aparece R_{CS} ya que sensa la corriente en el primario, mientras que R_{S1} es la que sensa V_{BULK}.

En cuanto a R_{CS}, esta es una resistencia de 3Ω que se encarga se sensar la corriente en la llave. Debido a los picos que aparecen en la corriente de la llave en los momentos de conmutación, esta resistencia se eligió de forma específica para que cumpla con esta función.

Por otro lado, está el circuito de clamp. Empezando con el TVS, que tiene una tensión de reposo inversa de 400V, una corriente pico máxima de 2.3A y permite gran disipación de potencia. Para el diodo serie se ha optado por un silicon-carbide (SiC) Schottky de 1200V, este diodo presenta velocidades de conmutación muy superiores y tiene una baja capacitancia parásita. Estas características permiten obtener una mayor eficiencia en el circuito de clamp y la SMPS.

Ahora analizaremos uno de los componentes más importantes del diseño, el MOSFET. Para obtener una mayor eficiencia en el circuito de clamp, se adoptó un MOSFET con una tensión drain-source de ruptura de 1000V. Este dispositivo presenta tres tipos de pérdidas que son significativas:

- Conmutación: Son las pérdidas que se generan cuando conmuta la llave y tanto su tensión como corriente son distintas de cero.
- Conducción: Estas pérdidas son producidas por la resistencia de encendido del dispositivo, R_{DSON}, cuando la llave está conduciendo.
- Capacitiva: Son las pérdidas producto de la potencia que se disipa en la capacidad parásita del MOSFET en el momento de encendido.

Se buscó un dispositivo que permita minimizar estas pérdidas y que se repartan de forma equitativa entre los tres tipos. Como resultado de esta búsqueda se adoptó el MOSFET STP2NK100Z de STMicroelectronics. A continuación calcularemos las pérdidas con este dispositivo, empezando por las de conmutación, dadas por la ecuación 2.82:

$$P_{SW} = \frac{1}{2} * I_{PP} * V_{DS} * t_{OFF} * f_{SW} = \frac{1}{2} * 0.2576A * 800V * 41.5nS * 100KHZ \cong 430mW \qquad (2.82)$$

En el momento de apagado de la llave la corriente está en su valor pico, mientras que la V_{DS} aumenta desde cero hasta su valor máximo, dado por la acción del circuito de clamp, en t_{OFF}. Como en el momento de encendido la corriente está en cero, no se producen pérdidas.

La ecuación 2.83 es utilizada para el cálculo de las pérdidas por conducción:

$$P_C = R_{DS(ON)} * {I_{P_{RMS}}}^2 = 8.5\Omega * 103mA^2 \cong 90mW \qquad (2.83)$$

Por ultimo las pérdidas capacitivas están dadas por la ecuación 2.84:

$$P_{CAP} = \frac{1}{2} * C_0 * {V_{DS}}^2 * f_{SW} = \frac{1}{2} * 28pF * 358V^2 * 100KHZ \cong 180mW \qquad (2.84)$$

Cuando la llave se enciende se disipa potencia en la capacidad parásita en el MOSFET. Y como el convertidor opera en DCM, el peor caso para V_{DS} es la máxima tensión de entrada. Sin

embargo, esto solo se dará si el convertidor opera de forma muy discontinua, ya que en otra circunstancia el controlador utilizara la técnica de valley-switching para reducir estas pérdidas.

En definitiva, se expresan las pérdidas totales en este dispositivo en la ecuación 2.85:

$$P_{MOS} = P_{SW} + P_C + P_{CAP} \cong 430mW + 90mW + 180mW \cong 700mW \qquad (2.85)$$

Ahora se calcula la temperatura de juntura que alcanzará el MOSFET mediante la ecuación 2.86:

$$T_J = P_{MOS} * R_{J-AMB} + T_{AMB} = 0.7W * 62.5\frac{^\circ c}{W} + 40°C = 43.75°C + 40°C = 83.75°C \qquad (2.86)$$

Este dispositivo tiene una T_J máxima de 150°C, lo que da un buen margen con respecto a lo indicado por la ecuación 2.86. Sin embargo, para tener mayor seguridad se utiliza un pequeño disipador con una resistencia térmica de 10°C/W. De esta forma el aumento de temperatura se mantiene en un nivel muy bajo.

En el bobinado auxiliar aparecen las resistencias de sensado RS1 y RS2, ya calculadas anteriormente, el diodo rectificador y el capacitor. Para el diodo rectificador se utilizó un Schottky, ya que aprovechando sus pequeños tiempos de conmutación, se podrá obtener un mejor muestreo de la tensión de salida. El fabricante recomienda una capacidad de 10µF, se decidió por un capacitor más conservador, de 47µF.

Con respecto al capacitor que se coloca entre las masas del circuito primario y secundario, para reducir las interferencias de modo común en la carga, se utilizó uno de seguridad Y2.

Ya en el circuito secundario aparecen los diodos rectificadores en la salida. Para aumentar la eficiencia se seleccionaron diodos Schottky, con tiempos de recuperación muy rápidos y una tensión en directa pequeña.

La tensión en inversa del diodo es otro parámetro importante que se debió elegir. La ecuación 2.87 muestra el cálculo para la misma:

$$V_R = \frac{V_{BULK(MAX)}}{N_{PS}} + V_0 = \frac{358V}{38} + 5V \cong 14.42V \quad (2.87)$$

En definitiva, se escogió un valor de 100V para este parámetro, dejando un margen de seguridad importante y dando espacio a oscilaciones de alta frecuencia que podrían aparecer.

El fabricante, Vishay Semiconductor Diodes Division en este caso, ofrece una curva de la potencia promedio disipada en el diodo en función de la corriente promedio, con el ciclo de trabajo de la onda cuadrada como parámetro. Se puede observar en la figura 2.62 el gráfico mencionado para el V8P10. El que mayor potencia disipada es el del secundario principal, al que se diseñó para una corriente promedio máxima de 1.5A y un ciclo de trabajo máximo de 0.425. Si se ingresa con estos datos en la curva de la figura 2.62, resulta una potencia de pérdida de aproximadamente 0.8W.

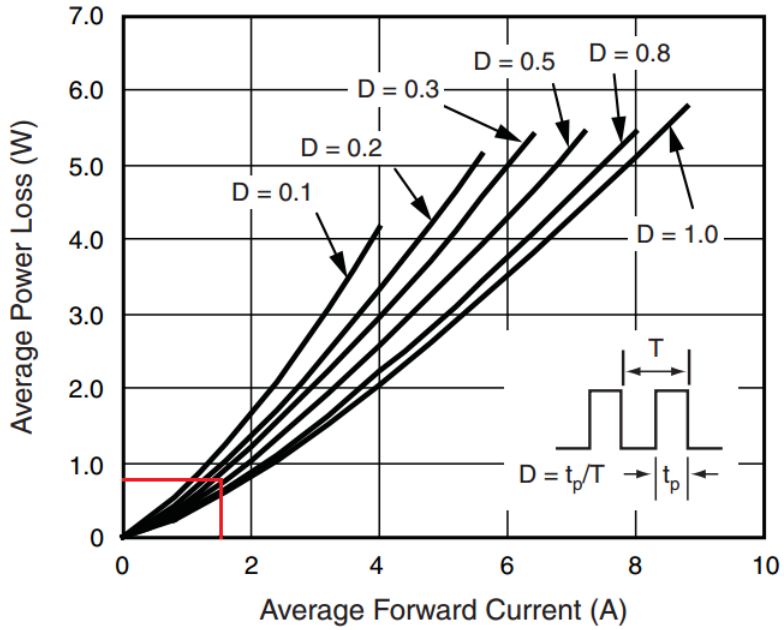


Figura 2.62: Potencia de pérdida en el diodo vs corriente promedio.

Ahora calculamos la temperatura de juntura que alcanzara el diodo mediante la ecuación 2.88:

$$T_J = P_D * R_{J-AMB} + T_{AMB} = 0.8W * 60\frac{^\circ C}{W} + 40°C = 88°C \qquad (2.88)$$

Este valor tiene un buen margen con respecto a los 150°C para la máxima temperatura de juntura permitida. Sin embargo, en el diseño de la PCB de utilizó un pad más grande para este diodo, para que pueda disipar aún mejor el calor.

Otra consideración con respecto a este diodo, es su capacidad parásita. Esta capacidad junto con la inductancia del secundario, genera oscilaciones de alta frecuencia y grandes sobrepicos de tensión. Para solucionar este inconveniente se añade una red RC en paralelo al diodo, como se puede apreciar en la figura 2.61, para bajar el Q de estas oscilaciones. De la hoja de datos del fabricante se saca que para una tensión inversa del orden de los 15V, se tiene una capacidad de aproximadamente 200pF. Con lo cual, se asignó un valor de 2nF para el capacitor de damping, 10 veces mayor al que presenta el diodo. Mientras que para obtener un valor de Q cercano a 0.5 se debe cumplir lo expresado por la ecuación 2.89:

$$Q = \frac{1}{\omega_O R_D C_D} \cong 0.5 \quad (2.89)$$

Como falta conocer la frecuencia de oscilación del tanque LC, lo primero que determinamos es el valor de la inductancia del secundario, dado por la ecuación 2.90:

$$L_S = \frac{L_P}{{N_{PS}}^2} = \frac{3.92mH}{38^2} \cong 2.71\mu H \qquad (2.90)$$

Ahora con este valor, utilizamos la ecuación 2.91 para calcular la frecuencia de oscilación:

$$f_0 = \frac{1}{2\pi\sqrt{L_S * C_D}} \cong 6.8MHZ \qquad (2.91)$$

Por lo tanto, ingresando a la ecuación 2.91 y despejando el valor de la resistencia de damping, se obtiene un valor aproximado de 23.4Ω. Finalmente se toma el valor comercial de 22Ω para esta resistencia.

Solo restan los capacitores de salida para analizar en esta parte del circuito. En cuanto a la salida principal, se han colocado dos, uno electrolítico y otro cerámico. El electrolítico es de 470µF, 16V, tan solo 10mΩ de ESR y un ripple de corriente de 5.1A. Mientras que el capacitor cerámico mejora aún más el ESR del capacitor de salida. La hoja de ensayo de este capacitor indica que tiene un ESR cercano a 1mΩ para una frecuencia de 100KHZ. Por otro lado, en los otros dos secundarios las corrientes RMS no son tan grandes, el capacitor no tiene por qué ser de tan buena calidad. Estos capacitores se adoptaron de 220µF, 25V, 20mΩ y 3.8A de ripple de corriente.

A continuación se muestra la parte restante del circuito, limitador y realimentación, en la figura 2.63.

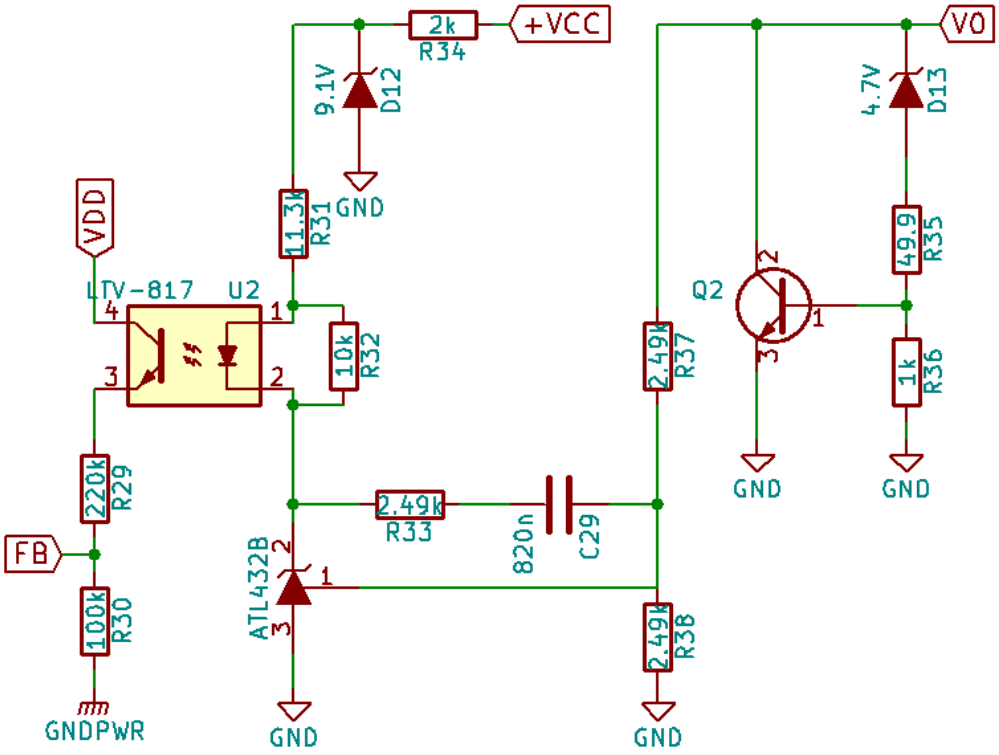


Figura 2.63: Ultima parte del circuito de la SMPS.

El circuito de realimentación ya se explicado en profundidad y no hay ninguna consideración más para hacer, simplemente mencionar que las resistencias que sensan la tensión de salida se han adoptado con una tolerancia de 0.1% para mantener bajo el error en V_0. Además se ha utilizado el opto-acoplador LTV-817S, un dispositivo típico para estas aplicaciones.

En cuanto al circuito limitador, se ha adoptado un diodo zener con una tolerancia de 3%, ya que sobretodo este valor es el que determinara a que nivel de tensión quedará limitada la excursión

de V_0. Por otro lado, el transistor posee una $V_{CE(MAX)}$ de 30V y una corriente de colector máxima de 5A.

2.12 Simulación de la SMPS

Se ha realizado una simulación de la fuente en el programa LTSPICE de Linear Technology. El propósito de realizar la simulación era sobretodo obtener una respuesta dinámica adecuada en el peor caso, es decir, sin carga externa. También sirvió mucho para el análisis del circuito, comparar componentes, etc.

El circuito utilizado en la simulación es muy similar al mostrado en las figuras 2.60, 2.61 y 2.63, y se puede observar en la figura 2.64.

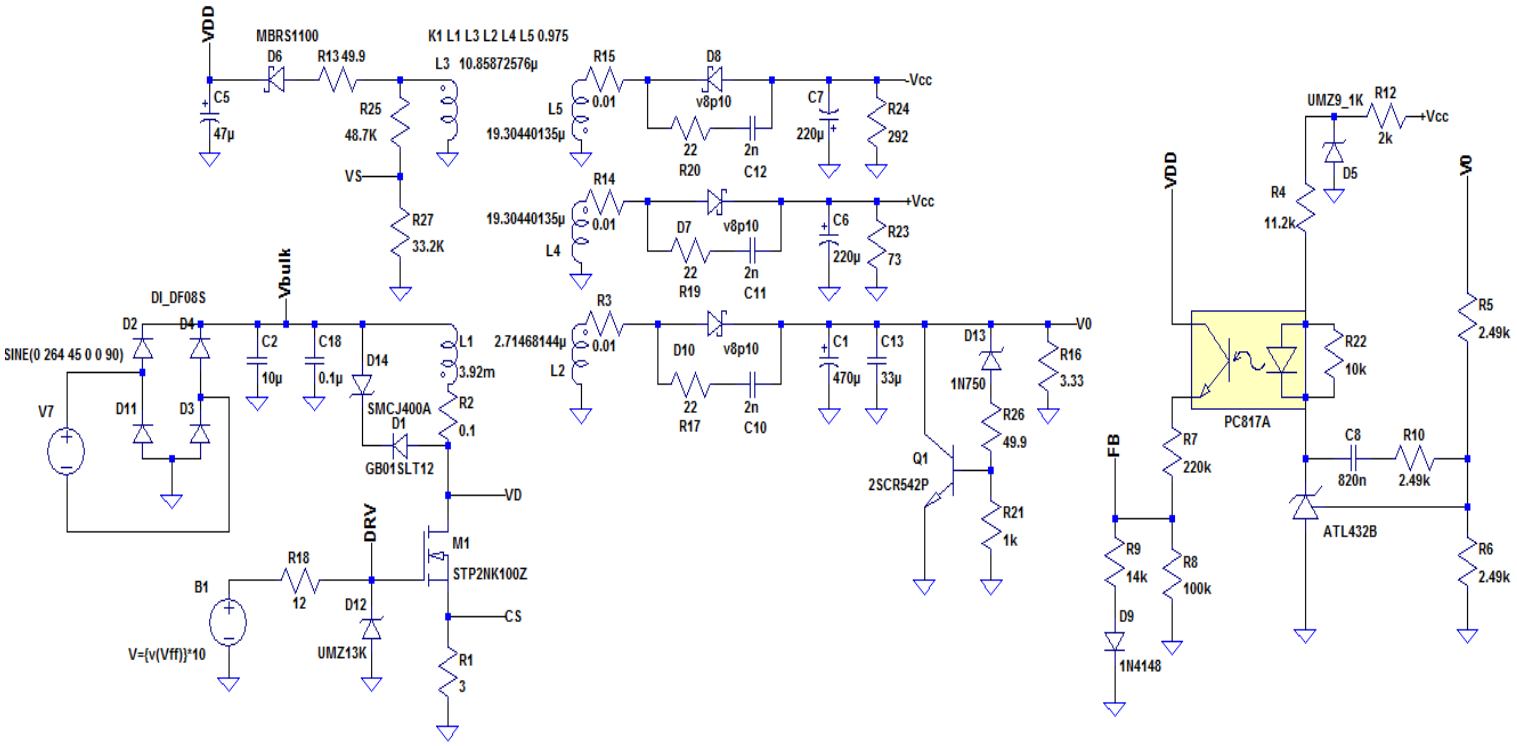


Figura 2.64: Circuito utilizado en la simulación.

Este circuito presenta algunas diferencias con el circuito real. Para empezar no se incluyó el filtro EMI, ya que la presencia de este no modifica el funcionamiento normal de la fuente. Aunque si se realizó la simulación del filtro con un circuito equivalente que se analizará más adelante. El circuito de drive se ha modelado con una resistencia de 12Ω, que representa a la R_{DSON} del MOSFET interno del circuito de drive del UCC28740 que impone la corriente de apagado, y además un diodo zener interno del controlador que limita la tensión en el gate a 14V. También se ha añadido la impedancia de entrada que presenta el pin FB, compuesta por una resistencia de 14KΩ y un diodo.

No se ha incluido la resistencia R_{LC}, ya que no se busca simular la característica de compensación de la línea que utiliza el controlador. En cuanto al transformador, en cada bobinado se añadió una resistencia serie que representa la resistencia del cobre y además se ha supuesto una inductancia de pérdida del 5%. Por último, en cada salida se agregó una resistencia de carga, que en el circuito real no están.

Para controlar la potencia que se entrega a la carga, el controlador lo que modifica es tanto I_{PP} como f_{SW}. Estas dos variables se determinan de acuerdo a la potencia que demanda la salida y se determinan con las curvas de la figura 2.15. El resto de las características del UCC28740 se dejan afuera de la simulación, incluyendo el modo de operación especial en el arranque, el valley-switching y el control de corriente en la salida. Por lo tanto, para incluir al controlador en la simulación se utilizó un circuito que sea capaz de variar la frecuencia y la corriente pico en el primario, de acuerdo a lo indicado por la figura 2.15. El circuito utilizado para realizar esta función se puede observar en la figura 2.65.

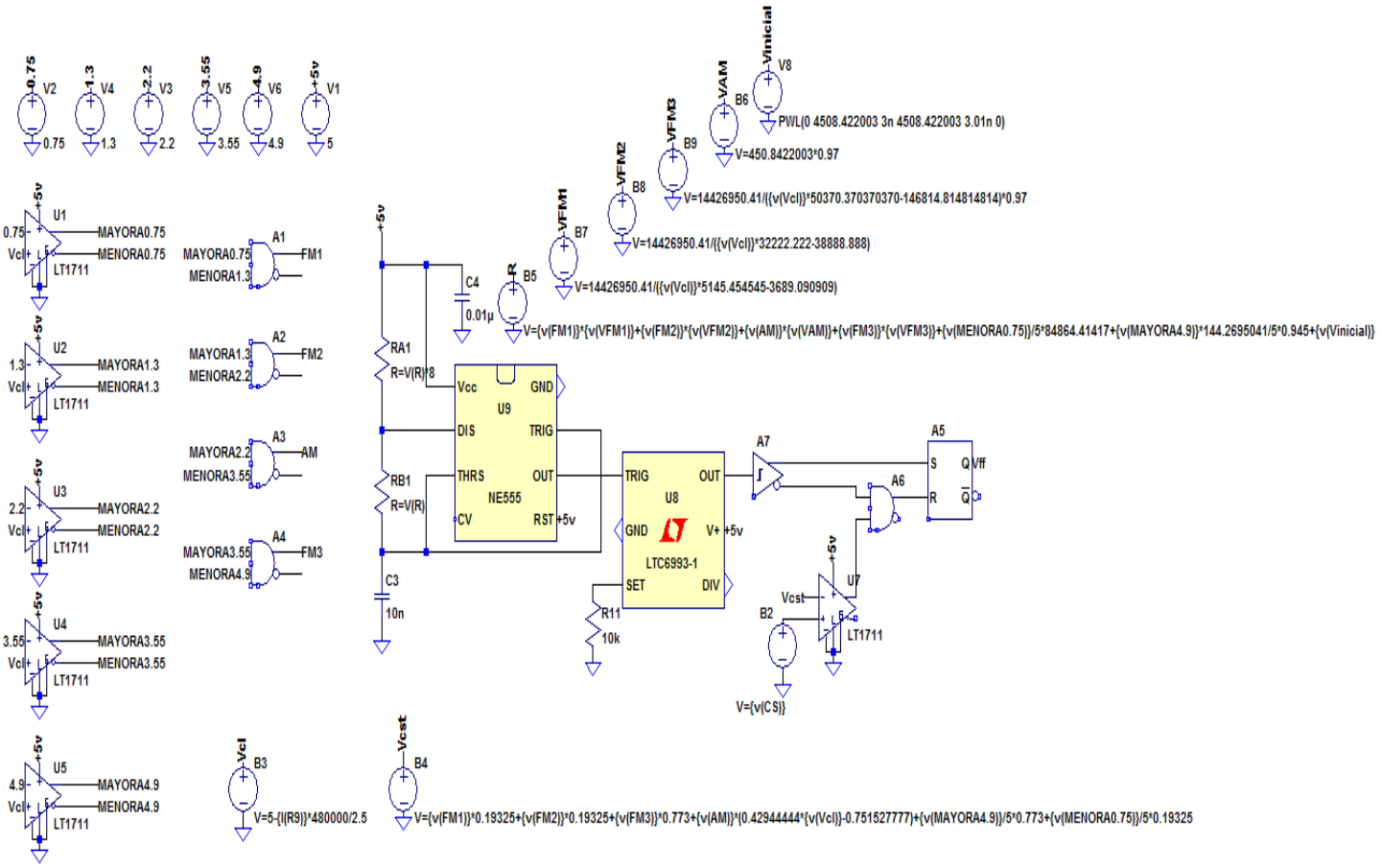


Figura 2.65: Circuito utilizado para simular al UCC28740.

Lo primero que hace este circuito es generar la tensión de control, V_{CL}, que es la variable interna del controlador que impone tanto la I_{PP} y f_{SW}. Esta tensión se genera a partir de la corriente

que ingresa al pin FB. I_{FB} es divida por 2.5 y luego mediante una resistencia de pull-up de 480KΩ a una tensión de 5V, genera V_{CL}, esto es hecho mediante el generador de función V_{CL} mostrado en la figura.

La figura 2.15 muestra cuatro regiones que utilizan un determinado método de control, con lo cual, se utilizan una serie de comparadores que sirven para saber dentro de que zona se está operando.

Para imponer la I_{PP} que determina la tensión de control, se utiliza el generador de tensión llamado V_{CST}. La corriente en el primario es sensada mediante R_{CS}, generando una tensión que es comparada con V_{CST}. De esta forma la corriente pico en el primario será V_{CST} dividida R_{CS}, momento en el cual termina el tiempo de encendido de la llave. El valor del generador V_{CST} es función de V_{CL}, siguiendo lo impuesto por la curva de la figura 2.15.

Para generar la f_{SW} variable se utiliza un circuito con un integrado 555. Esto se consigue utilizando la configuración astable del 555 con una resistencia que varía de acuerdo a la frecuencia del clock pretendida. Por lo tanto, de acuerdo a lo impuesto por la curva de la figura 2.15 se determina la frecuencia en función de V_{CL}. Luego se genera un valor para la resistencia mencionada que sea función de la frecuencia. La salida del 555 ingresa a un monoestable. Esto se hace para generar el tiempo de blanking, necesario para que no haya un pulso falso en el RESET del FF cuando se generan sobrepicos en la corriente de source. Finalmente aparece el FF, cuya salida es la que será utilizada para comandar el MOSFET.

Ya se ha explicado el funcionamiento del circuito utilizado para simular la dinámica de la SMPS, ahora se presentarán los resultados. A continuación se analizarán las formas de ondas más importantes, para el peor caso en cuanto a la potencia, esto es con carga máxima y mínima tensión de entrada.

Resultados para carga máxima y mínima tensión de entrada

Tensión de salida principal

En esta salida se ha colocado una resistencia de carga de 3.33Ω, lo que impone una corriente de 1.5A y una potencia de 7.5W en esta salida. En la figura 2.66 se puede observar la forma de onda de V_0.

Se puede apreciar en el gráfico de más abajo que la tensión de salida no tiene error de continua. Con respecto al ripple, se pueden ver dos componentes de distinta frecuencia. Una que se ve claramente en el grafico del medio, tiene una frecuencia dada por el lazo de control. Mientras que la otra componente es de 100KHZ, la frecuencia de conmutación. Sumando ambos fenómenos se obtiene una tensión de ripple aproximada de 100mV.

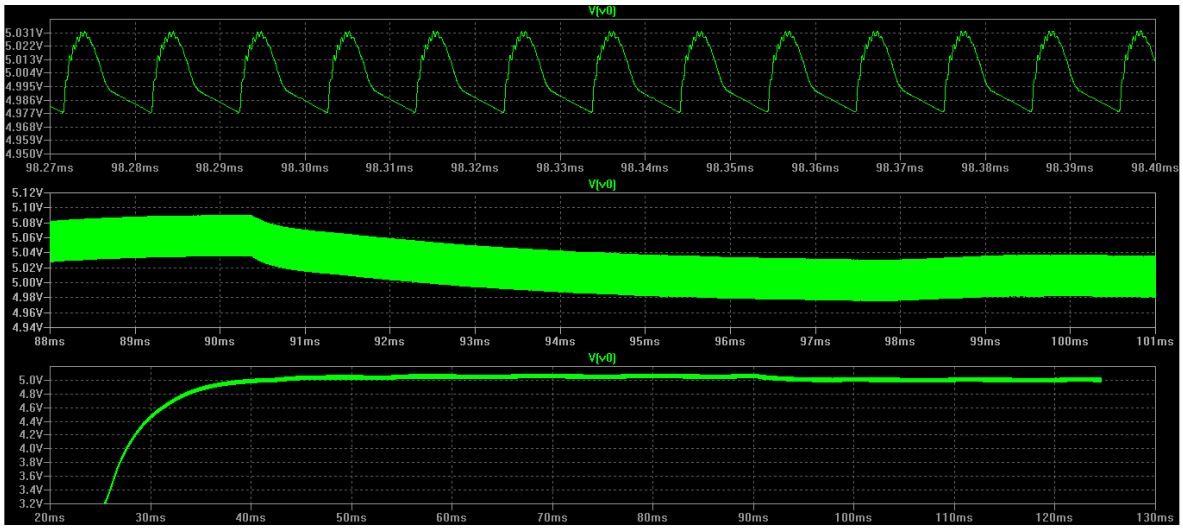


Figura 2.66: Forma de onda de la tensión de salida.

Corriente de realimentación

A continuación en la figura 2.67 se muestra la corriente de realimentación y la tensión de control. Al principio se entrega máxima potencia, esto implica I_{FB} al mínimo y V_{CL} al máximo. Cuando la tensión de salida supera los 5V, el lazo aumenta la corriente I_{FB} ligeramente por debajo de su valor mínimo. Por lo tanto, se empieza a conmutar con una frecuencia levemente inferior a la máxima hasta alcanzar de nuevo los 5V. Es decir, que la corriente de realimentación aumenta cuando la tensión de salida está por encima de 5V y disminuye cuando V_0 está debajo de 5V. Con lo cual, se genera una oscilación, cuya frecuencia está dado por la velocidad del lazo de control.

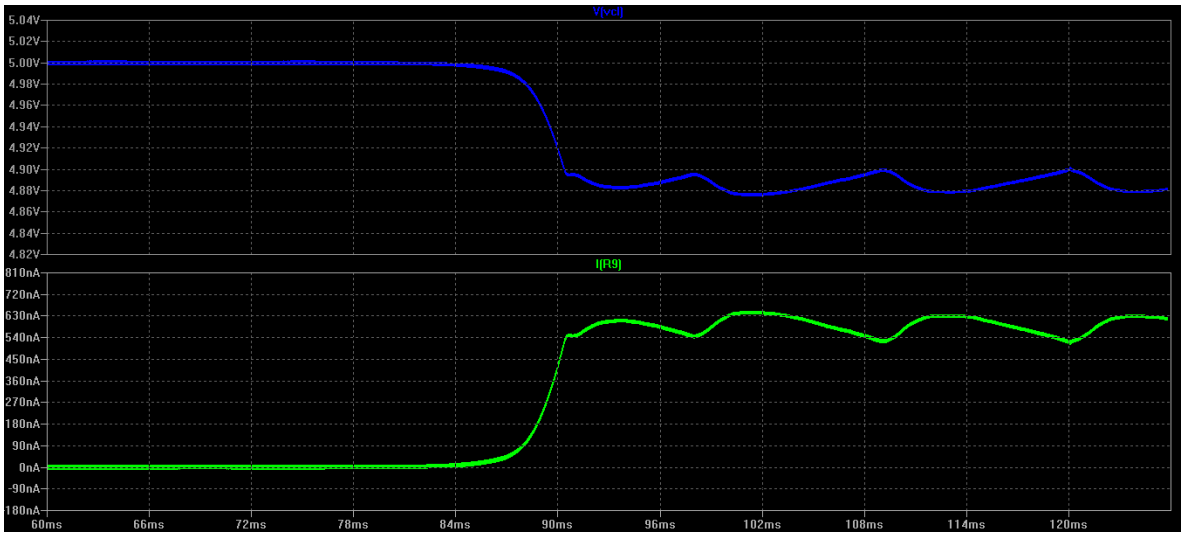


Figura 2.67: I_{FB} en verde y V_{CL} en azul.

$\pm V_{CC}$

Las formas de onda se pueden apreciar en la figura 2.68. Obviamente estas formas de onda presentan el mismo comportamiento que V_0, con la diferencia del valor que alcanzan. Su valor en continua es cercano a los 15V, un poco superior a lo calculado en la ecuación 2.69. Por último, el ripple en estas señales es mucho menor al que presenta V_0, ya que las corrientes RMS en sus bobinados son mucho menores.

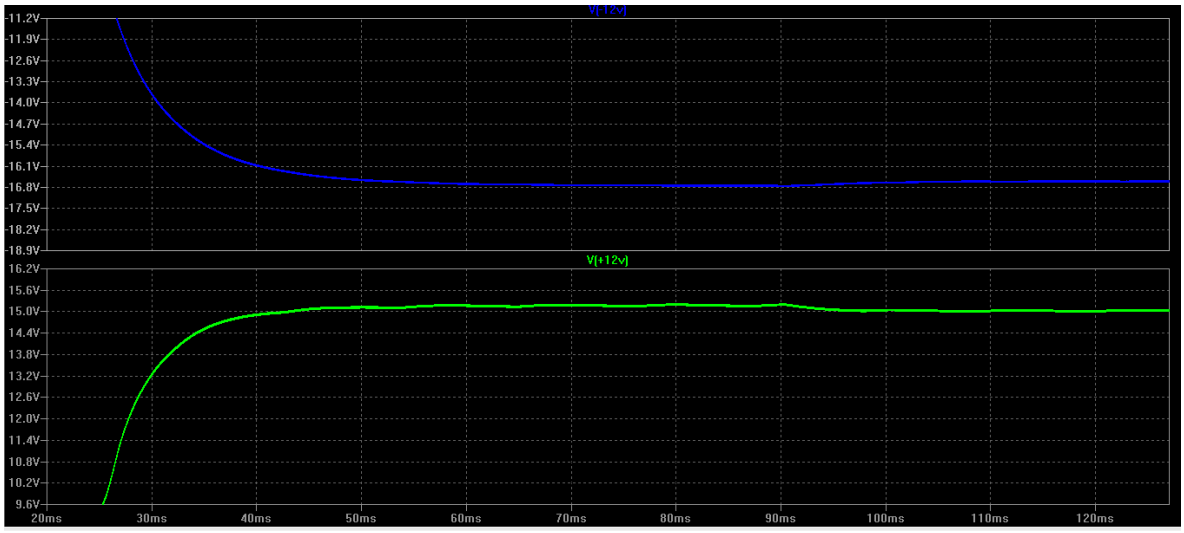


Figura 2.68: $+V_{CC}$ en verde y $-V_{CC}$ en azul.

Tensión de drain

V_D es mostrada en la figura 2.69. Cuando la llave está cerrada V_D es 0V, en cambio cuando la llave se abre actúa el circuito de clamp limitando la excursión a un valor V_{CLAMP} más V_{BULK}:

$$210 + 440V \cong 650 < V_{BULK} + V_{CLAMP} < 264V + 440V \cong 704V$$

El circuito de clamp limita la tensión hasta el momento que se termina la energía en la inductancia de pérdida del primario, luego empieza una oscilación de muy alta frecuencia producida por L_{LEAK} y por las capacidades parásitas de los componentes. Esta oscilación se produce entorno a la tensión reflejada desde el secundario más la tensión de entrada, calculada a continuación:

$$210V + (5V + 0.6V)38 \cong 423V < V_{BULK} + (V_0 + V_F)N_{PS} < 264V + (5V + 0.6V)38 \cong 477V$$

Pero cuando se termina el tiempo de desmagnetización y empieza el tiempo discontinuo se produce una nueva oscilación. Esta oscilación es ahora entorno a V_{BULK} y de menor frecuencia, ya que ahora resuenan la inductancia de primario con las capacidades parásitas. Pero como se está operando muy cerca del modo de transición, este tiempo es muy pequeño y apenas en el último ciclo que se ve en el gráfico se puede apreciar medio periodo de esta nueva oscilación.

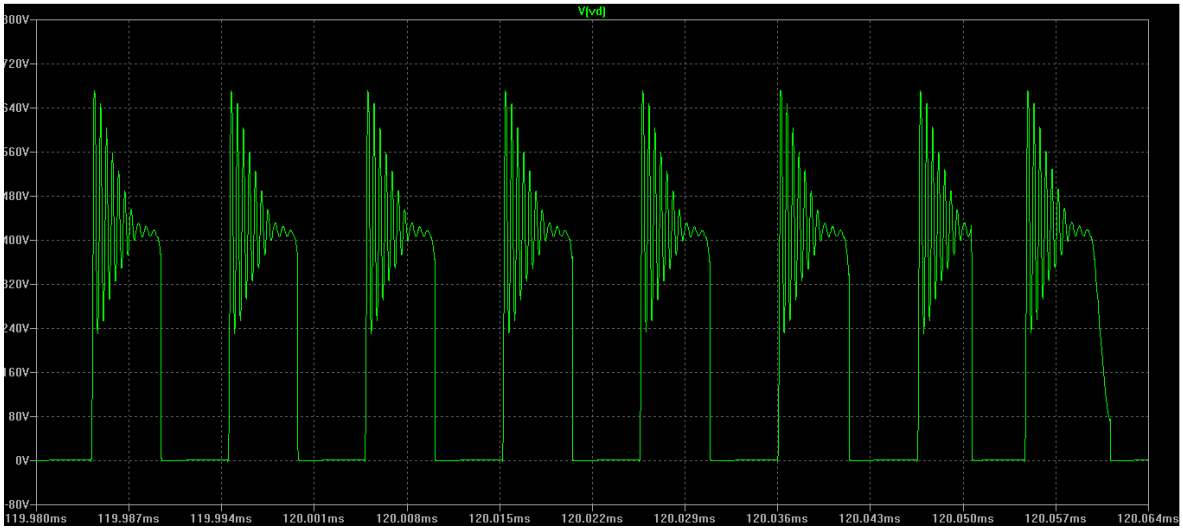


Figura 2.69: V_D.

Tensión de gate

Esta forma de onda es presentada en la figura 2.70. De verde esta la tensión de gate y de azul la tensión de entrada. Se ha incluido el valor mínimo de V_{BULK} para comparar el ciclo de trabajo en esta condición, con el calculado en la etapa de diseño. Se midió un ciclo de trabajo de aproximadamente 49.3% contra el 48.1% esperado, resultando una diferencia muy pequeña.

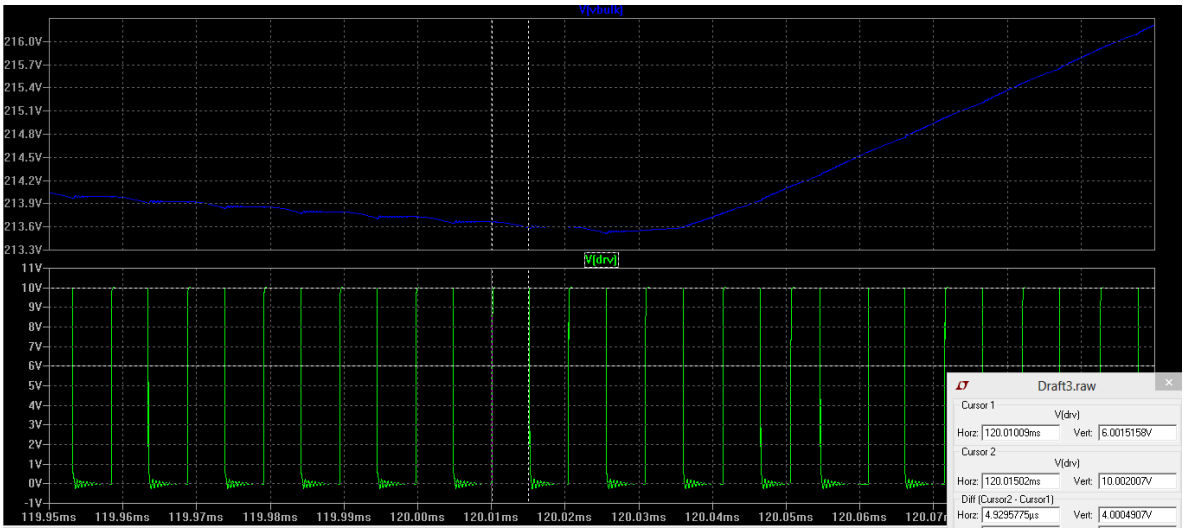


Figura 2.70: V_G de verde y V_{BULK} de azul.

Tensión de entrada

La figura 2.71 ahora muestra V_{BULK} y V_D. Con la tensión de línea mínima, esta imagen nos permite corroborar el valor de $V_{BULK(MIN)}$, estimada en más de 210V. Este valor es efectivamente

correcto como se muestra en la imagen. Además se agregó la tensión de drain para mostrar la dependencia entre $V_{D(MAX)}$ y V_{BULK}.

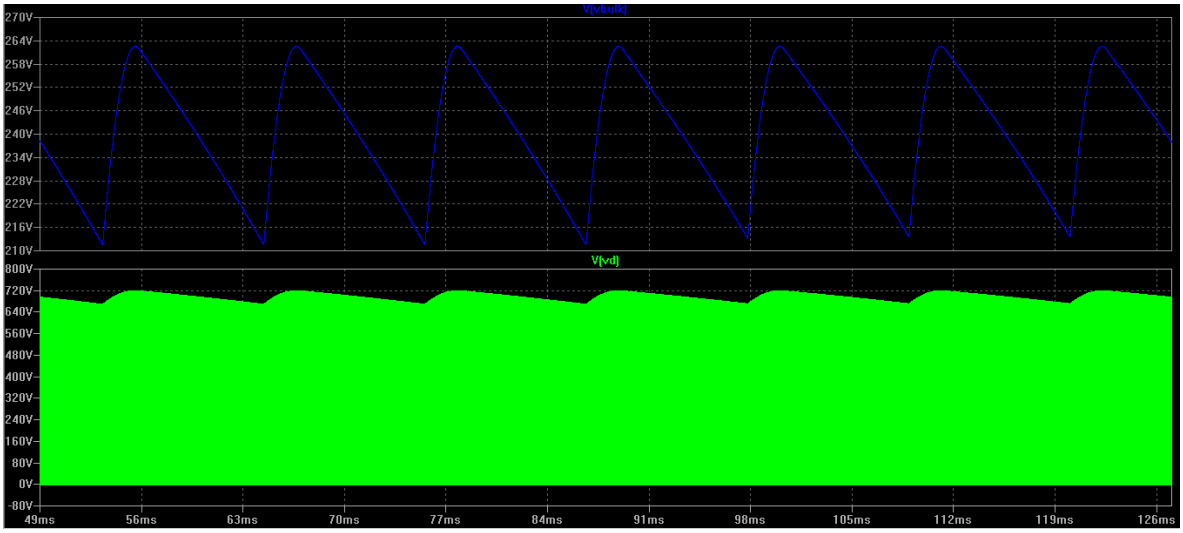


Figura 2.71: V_{BULK} en azul y V_D en verde.

Corriente en el primario

Ahora veremos la corriente en el bobinado primario y la tensión que entrara en el pin CS, mostradas en la figura 2.72. En cuanto a la corriente del primario, lo más importante es verificar el valor pico que alcanza. La imagen muestra una corriente pico de 0.258A en la simulación, contra los 0.2576A establecidos en la etapa de diseño. El otro punto a remarcar es el pico de tensión que aparece en V_{CS} en el momento en que conmuta la llave, razón por la cual se debe agregar un tiempo de blanking.

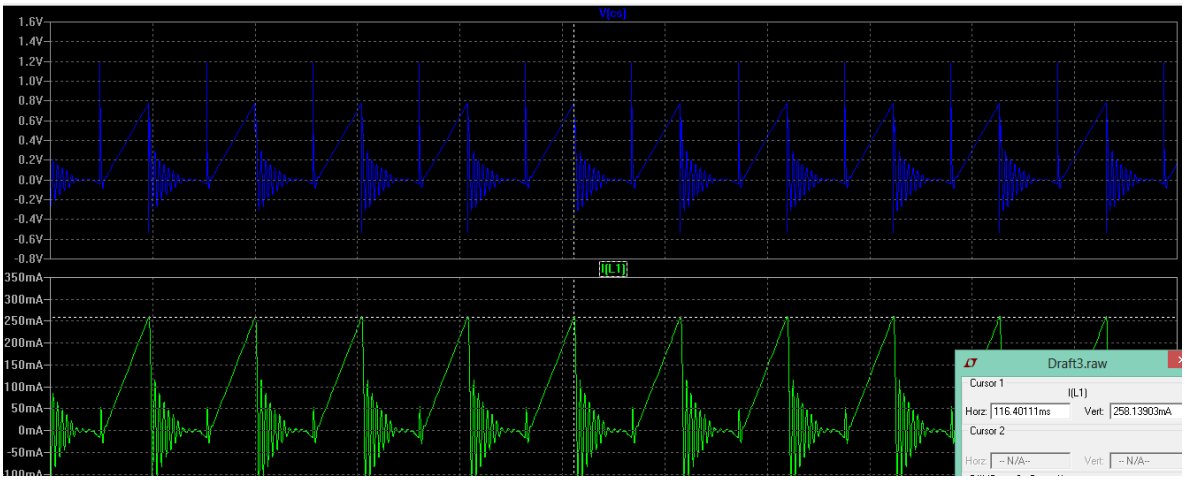


Figura 2.72: I_P en verde y V_{CS} en azul.

Corrientes en los bobinados

A continuación, se presentan las formas de onda de las corrientes en cada de uno de los bobinados, en la figura 2.73. La corriente pico en cada uno de los secundarios está dada por I_{PP}, la relación de vueltas entre el primario y cada uno de los secundario, y la fracción de la potencia en cada bobinado y la potencia total. En verde esta la corriente del primario, en azul la del bobinado principal, en rojo la del bobinado $+V_{CC}$ y la última es la corriente de $-V_{CC}$.

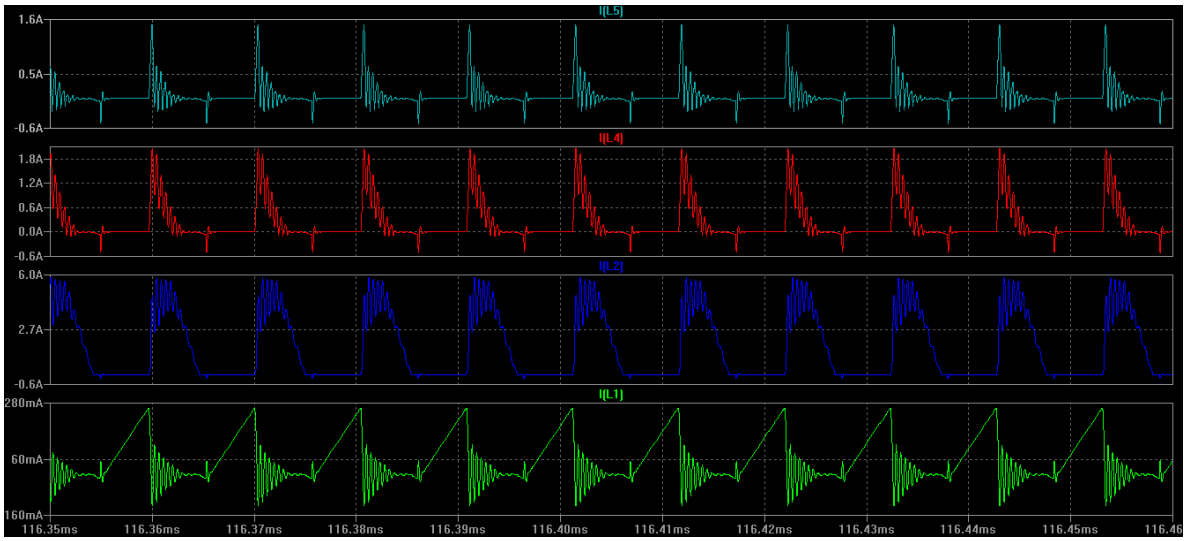


Figura 2.73: Corriente en el bobinado primario y en cada uno de los bobinados secundarios.

Tensión en el pin VS

Esta señal es mostrada en la figura 2.74 en dos escalas de tiempo diferente. La que se encuentra abajo permite apreciar como V_{VS} varía con la tensión de entrada, mientras que en la de arriba se ve mejor la señal que es muestreada por el controlador.

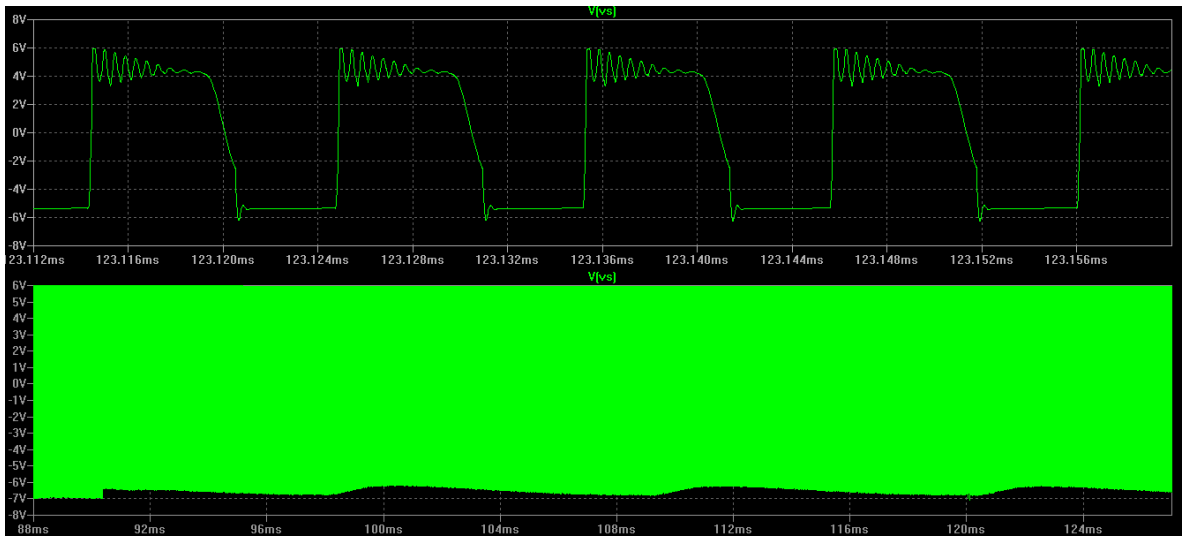


Figura 2.74: V_{VS} con dos escalas de tiempo diferente.

Formas de onda en el diodo de salida.

El análisis continúa con las formas de onda en el diodo de la salida principal, cuyas formas de onda se presenta en la figura 2.75. De verde aparece la tensión en el diodo, donde se marca la tensión en directa de aproximadamente 0.6V, y la tensión en inversa con un valor cercano a los 12V. En azul aparece la corriente en el diodo, que es la misma que la señal azul de la figura 2.73. Mientras que en rojo se presenta la potencia instantánea en el diodo, cuyo valor promedio resulto de 807mW. Este valor es prácticamente igual al indicado por la curva del fabricante, presentada en la figura 2.62.

Figura 2.75: Formas de onda en el diodo de la salida principal.

Luego se analizarán las pérdidas en el MOSFET y el circuito de clamp, pero esto se hará cuando se trabaje con la tensión de línea máxima, ya que esta es la peor condición en cuanto a la potencia disipada en el MOSFET.

Resultados sin carga externa

Hasta acá llega el análisis para la situación de carga máxima y mínima tensión de línea. Y ahora se plantea el caso contrario en cuanto a la carga, por lo que se deja al circuito secundario sin carga externa. A continuación se mostraran los resultados para esta situación, empezando por la tensión de salida y la corriente de realimentación, que se pueden observar en la figura 2.76.

Tensión de salida y corriente de realimentación.

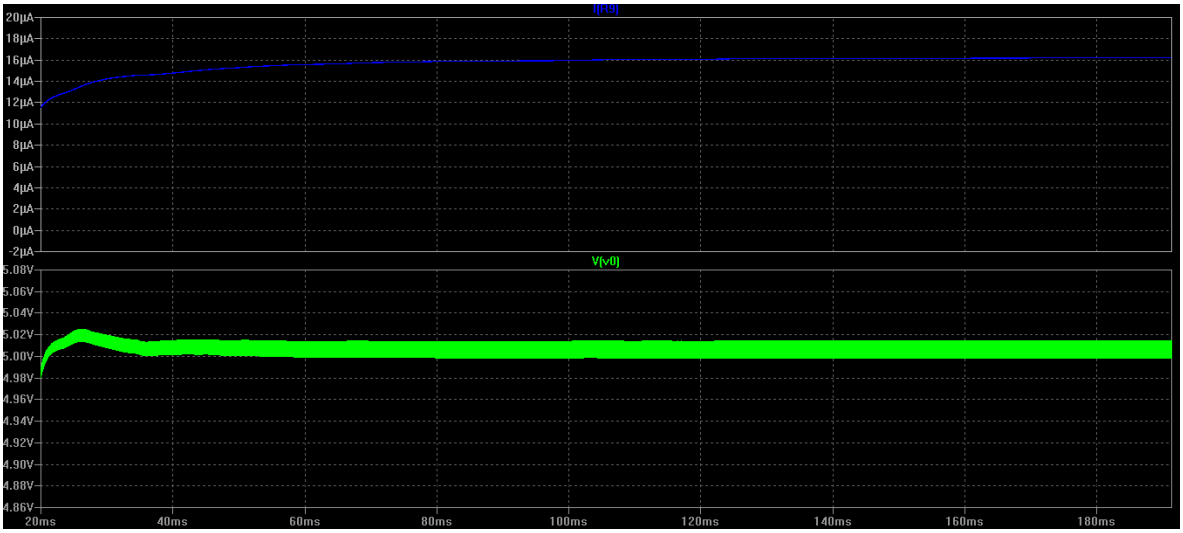


Figura 2.76: V_0 en verde y I_{FB}, sin carga externa.

En la tensión de salida aparece un pequeño sobrepico y tiene un ripple muy pequeño en comparación al caso anterior. Esto último se debe a dos factores, el primero es que ahora la corriente RMS en el secundario que debe filtrarse es mucho menor, y el segundo tiene que ver con que en la respuesta del lazo de control no aparecen oscilaciones. Con respecto a I_{FB}, ya no se presentan oscilaciones, como en el caso anterior, y por otro lado tiene un valor en continua superior a los 16µA.

Corriente en el primario

Seguimos con la corriente en el primario, que se muestra en la figura 2.77. La principal diferencia con el caso anterior es la corriente pico, que en este caso es casi cuatro veces más chica que cuando se estaba con carga máxima. También se resalta el cambio en la frecuencia de conmutación, ya que ahora el controlador está operando en una zona de modulación en frecuencia.

Tensión de drain

Por último, para este caso se muestra la tensión de drain en la figura 2.78. En esta imagen se puede apreciar de forma clara los tiempos de encendido, desmagnetización y discontinuo. El tiempo de conducción discontinua es el de mayor duración, y ahora si se puede ver como V_D oscila en torno a V_{BULK} durante este tiempo.

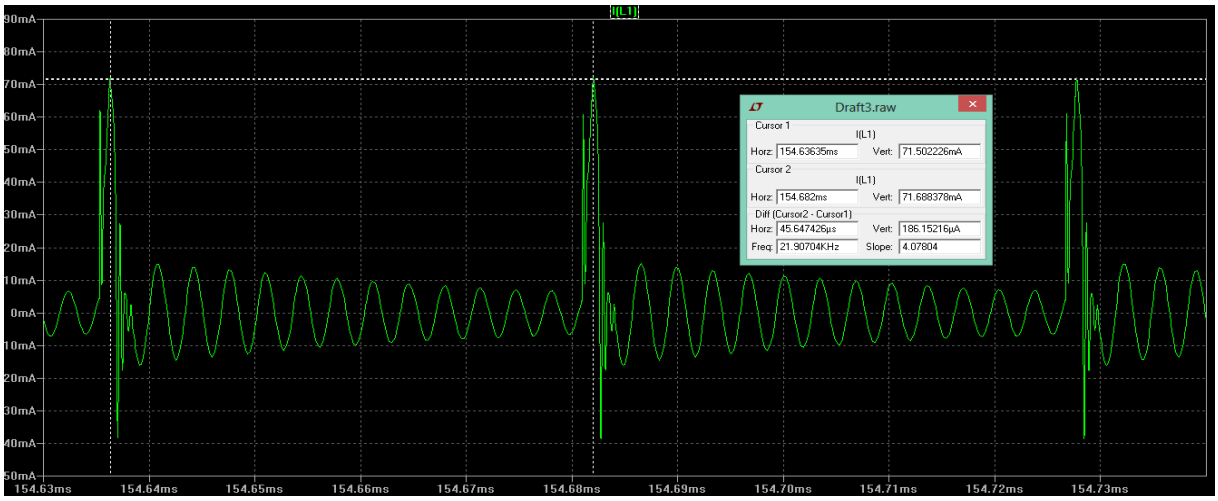


Figura 2.77: Corriente en el primario sin carga externa.

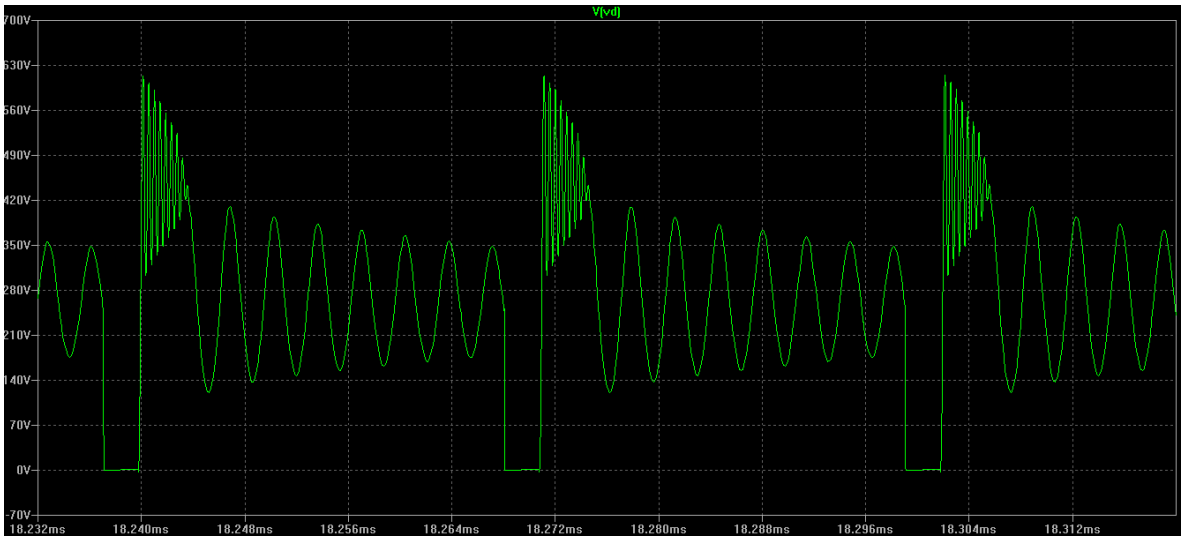


Figura 2.78: V_D sin carga externa.

Sin carga externa el controlador opera en la zona de FM con I_{PP} al mínimo. A partir de la forma de onda mostrada en la figura 2.77, podemos calcular cual es la potencia sin carga para esta fuente de alimentación. Este cálculo es expresado a continuación:

$$P_{SIN\ CARGA} = \frac{1}{2} * {I_{PP}}^2 * L_P * f_{SW} = \frac{1}{2} * 71.5mA^2 * 3.92mH * 22KHZ \cong 220mW$$

Resultados con carga máxima y tensión de entrada máxima

Finalmente, se plantea un último caso, que es con máxima carga y tensión de línea también máxima. Este caso nos permitirá analizar la potencia disipada en el MOSFET, el circuito de clamp y sobre todo hacer la simulación del filtro de modo común.

Comenzamos con la tensión de drain y la potencia disipada por el MOSFET, mostradas en la figura 2.79. Con respecto a V_D, se destaca que ahora tiene un valor máximo apenas por encima de 800V y que el convertidor ya no opera tan cerca del modo de transición. Ambas diferencias están relacionadas con que ahora se opera con $V_{BULK(MAX)}$. En cuanto a la potencia disipada en la llave, al valor medido en la simulación hay que sumarle las pérdidas capacitivas, ya que estas no son incluidas en el modelo SPICE. Por lo tanto a los 542mW medidos en la simulación hay que compararlos contra la suma de las pérdidas por conducción y conmutación, que según las ecuaciones 2.83 y 2.84 suman 530mW.

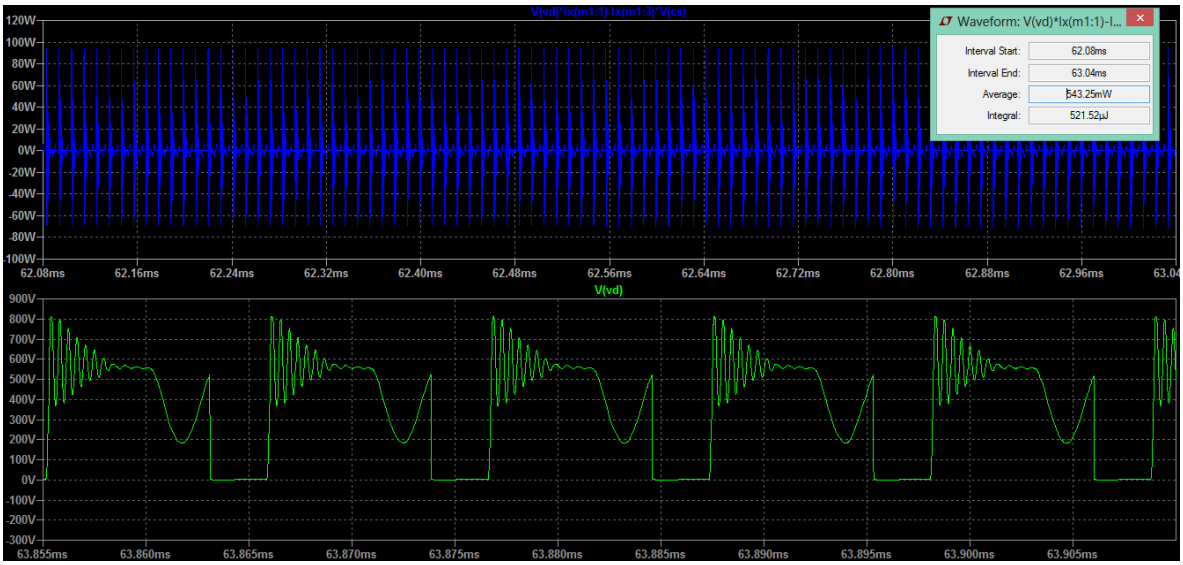


Figura 2.79: V_D en verde y la potencia disipada en el MOSFET de azul.

Ahora seguimos con la potencia disipada en el circuito de clamp, empezando con el TVS. La potencia instantánea se puede ver en la figura 2.80, y se mide una potencia promedio de 104mW.

Por otro lado, en la figura 2.81 se muestra la potencia instantánea en diodo SIC del circuito de clamp. Este dispositivo disipa simplemente 222µW.

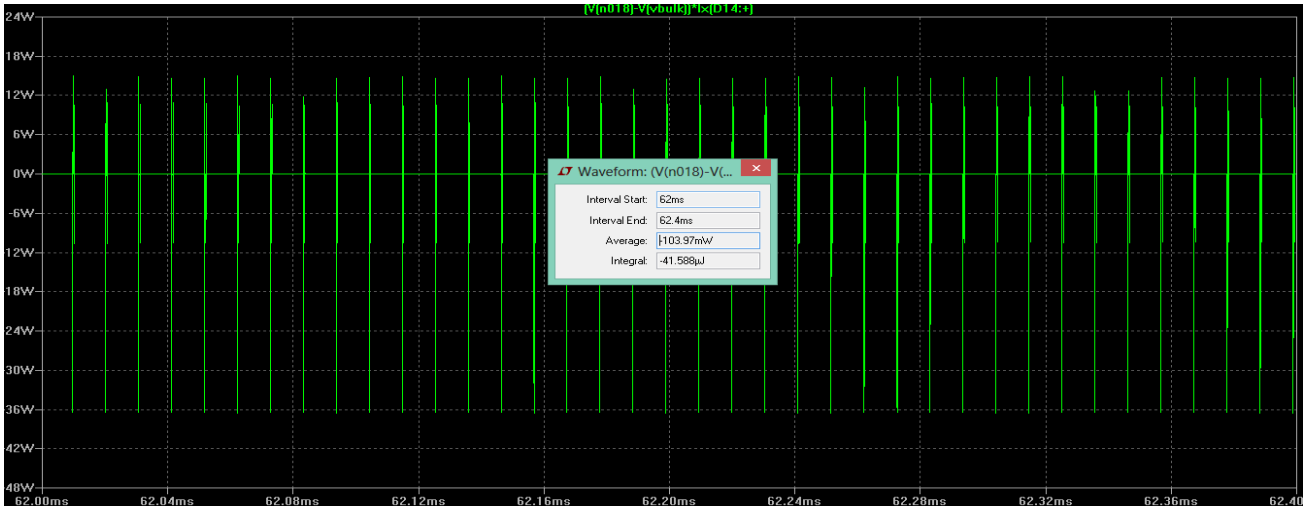


Figura 2.80: Potencia instantánea en el TVS.

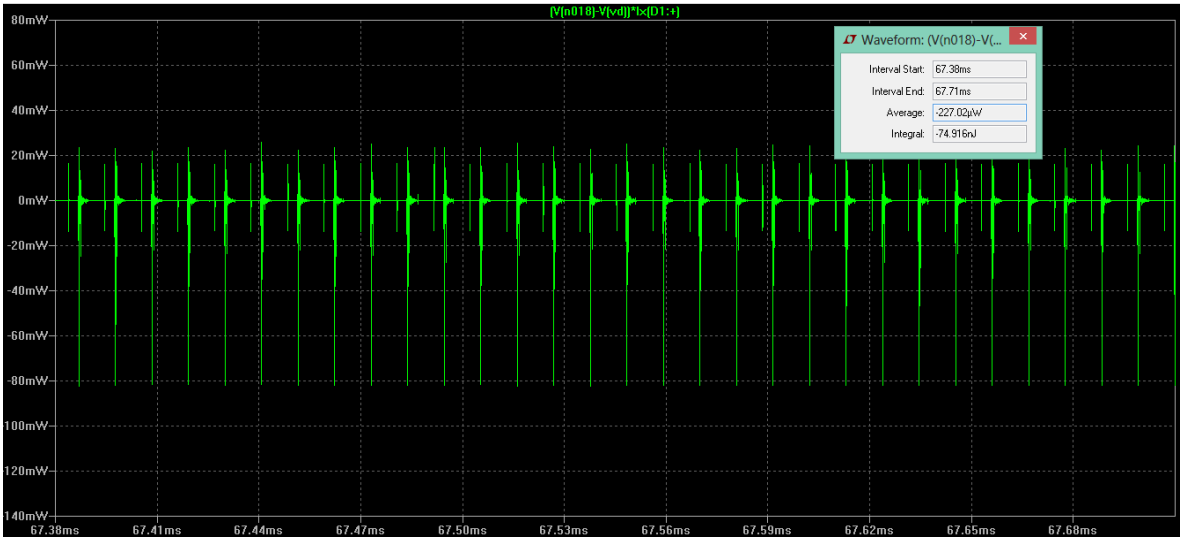


Figura 2.81: Potencia instantánea en el diodo SIC.

Para simular el filtro EMI se utilizó el circuito de la figura 2.82, mostrado a continuación.

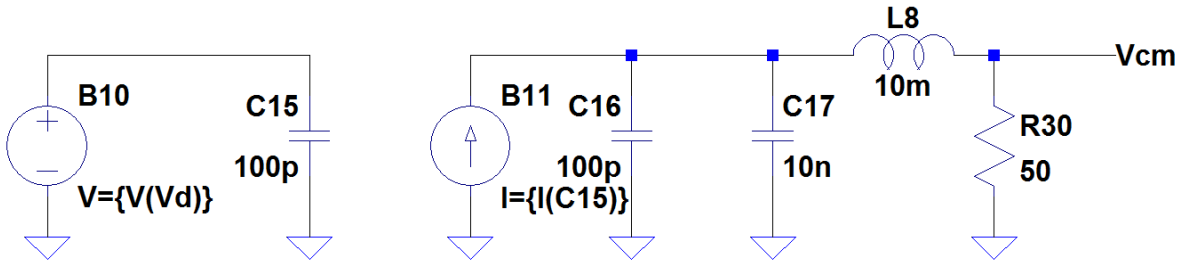


Figura 2.82: Circuito para simular el filtro EMI.

Se utilizó un generador de funciones que toma la señal de drain, que como se explicó anteriormente es el punto donde se generan las mayores interferencias, y una capacidad parásita de 100pF. Esta capacidad es fundamentalmente la que se presenta entre bobinado primario y secundario del transformador, que como se calculó con la ecuación 2.79 resulta de 80pF. Se toma 100pF para agregar un margen debido al cálculo y para tener en cuenta al resto de las capacidades que pueden generar corrientes de modo común. Luego se toma la corriente de este capacitor, que junto con el capacitor de modo común generan una tensión, que ingresa a un divisor formado por el inductor de modo común y la impedancia de la línea.

En la figura 2.83 se muestra la señal que ingresa al divisor de tensión y en la figura 2.84 se puede apreciar la tensión de modo común en la línea.

Figura 2.83: Señal de modo común que ingresa al divisor de tensión.

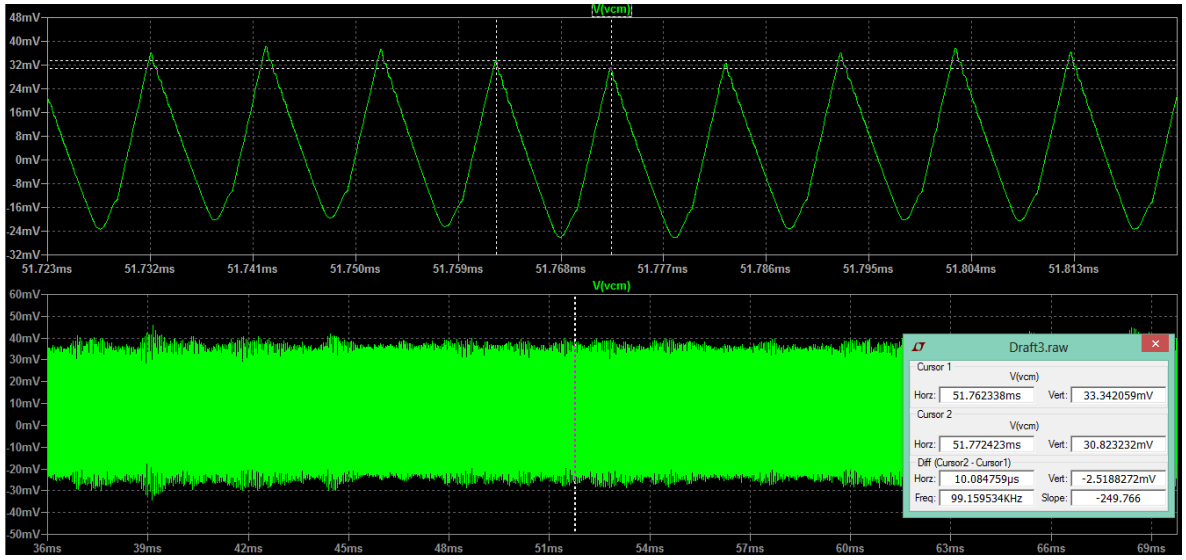


Figura 2.84: Tensión de modo común en la línea.

Para verificar el nivel de tensión de modo común impuesto por la curva de la figura 2.52, es necesario realizar la FFT de la forma de onda de la figura 2.94. La FFT resultante es la que se muestra en la figura 2.85.

En la curva de la figura 2.52 en líneas punteadas se muestran los niveles sugerido, desde 10KHZ hasta 150KHZ, y en línea continua los niveles obligatorios para cumplir con la norma.

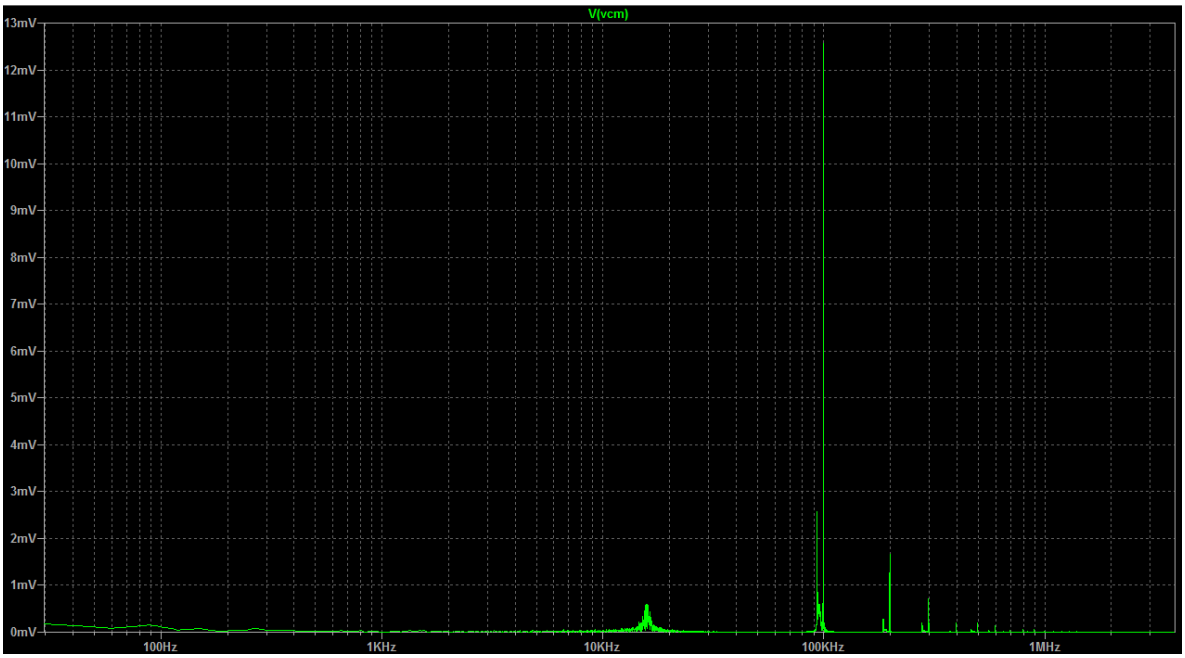


Figura 2.85: FFT de la tensión de modo común.

A continuación calcularemos los niveles para los armónicos de mayor amplitud:

- 10KHZ: $20\log\frac{0.5mV}{1\mu V} \cong 54dB$
- 100KHZ: $20\log\frac{12.5mV}{1\mu V} \cong 82dB$
- 200KHZ: $20\log\frac{1.7mV}{1\mu V} \cong 64.6dB$
- 300KHZ: $20\log\frac{0.8mV}{1\mu V} \cong 58dB$

Las componentes de mayor frecuencia son despreciables. A excepción de la componente de 100KHZ que está justo sobre la curva, el resto cumple con un margen considerable.

En conclusión, las simulaciones han mostrado resultados que se acercan mucho a lo esperado.

3

Amplificador Celda de carga

3.1 Galgas extensiométricas

Como se explicó al comienzo, el objetivo del ensayo consiste en medir el μ, el coeficiente de rozamiento del material. Por esta razón es que se aplica una carga normal al material y luego se mide el esfuerzo sobre el mismo. Por lo tanto, es necesario medir la fuerza de roce realizada sobre la pieza del ensayo. Para realizar esta medición se utiliza una celda de carga, un transductor que crea una señal eléctrica cuya magnitud es directamente proporcional a la fuerza aplicada.

A través de una estructura mecánica, la fuerza que está siendo sensada deforma a cuatro galgas extensiométricas en una configuración de puente de Wheatstone. Una galga extensiométrica es utilizada para medir esfuerzo mediante el cambio en su resistencia eléctrica. Este dispositivo hace uso de la dependencia de la resistencia eléctrica de un conductor con la geometría del mismo, como se expresa en la ecuación 3.1:

$$R = \rho \frac{L}{A} \qquad (3.1)$$

Donde ρ es la resistividad del material, L es el largo del conductor y A es la sección transversal del mismo. Por lo tanto, si un conductor es estirado dentro de los límites de su elasticidad, tal que no se rompa o se deforme permanentemente, se hará más estrecho y largo, produciéndose un aumento en la resistencia. En cambio, cuando el conductor se comprime se ensancha y se acorta, resultando una disminución de la resistencia.

Una típica galga extensiométrica es mostrada en la figura 3.1, donde se la puede apreciar en reposo, bajo tensión y en compresión.

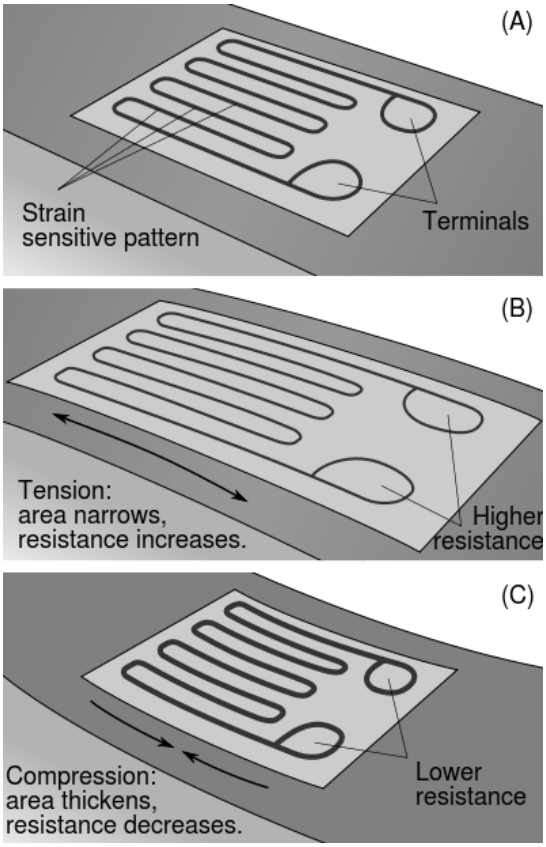


Figura 3.1: Galga extensiométrica.

Este dispositivo está formado por un soporte aislante y flexible, sobre el que se coloca la lámina metálica. Puede apreciarse como un conductor muy delgado se coloca en forma de zig-zag con líneas paralelas. Esto produce que ante un pequeño esfuerzo en la dirección de las líneas paralelas, se multiplica la medida del esfuerzo. Por esto motivo, es muy importante en qué dirección se aplica el esfuerzo. La galga debe adherirse a la estructura mecánica. Al ser exigida por una fuerza externa la lámina se deforma y cambia su resistencia. La sensibilidad de la galga respecto a los estiramientos está dada por el factor de galga, como se expresa en la ecuación 3.2:

$$GF = \frac{\Delta R/R}{\Delta L/L} = \frac{\Delta R/R}{\varepsilon} \qquad (3.2)$$

Donde se ve como el factor de galga es el cociente entre la variación relativa de la resistencia y la variación relativa de la longitud del conductor, que es igual al esfuerzo, ε. Debido a los límites impuestos por la elasticidad, se producen cambios muy pequeños en la resistencia, razón por la cual es necesario amplificar la señal entregada por estos dispositivos.

Efectos de la temperatura y el cableado

Las galgas no solo responden al esfuerzo aplicado, sino que también lo hacen a las variaciones de la temperatura. Este efecto produce lo que se llama esfuerzo aparente, que es producido por los siguientes efectos de la temperatura:

- Expansión térmica del material al cual está adherida la galga.
- Contracción térmica de la lámina metálica.
- Dependencia del factor de galga con la temperatura.
- Respuesta térmica de los cables de conexión

El esfuerzo aparente producido por estos fenómenos puede expresarse de forma simplificada en la ecuación 3.3:

$$\varepsilon_R = (\frac{\alpha_R}{GF} + \alpha_B - \alpha_M)\Delta T \qquad (3.3)$$

Donde α_R es el coeficiente térmico de la lámina metálica, α_B es el coeficiente de expansión térmica del material al cual está adherida la galga, α_M es el coeficiente de expansión térmica de la lámina y ΔT es el aumento de temperatura que provoca el esfuerzo aparente. Esta es una ecuación simplificada, ya que solo se tiene en cuenta los efectos lineales. Los fabricantes de galgas extensiométricas ofrecen una curva, aproximada con un polinomio de cuarto orden, del esfuerzo aparente en función de la temperatura.

Para mitigar este problema los fabricantes usan una técnica llamada Self-Temperature Compensation (STC), que consiste en adaptar el coeficiente térmico de la lámina mediante técnicas de producción para que cumpla con la ecuación 3.4:

$$\alpha_R = (\alpha_M - \alpha_B)GF \qquad (3.4)$$

De esta forma se cancela el esfuerzo aparente, aunque se requiere compatibilidad entre el material al cual está adherida la galga y la galga misma. Para ilustrar los efectos de aplicar STC, en la figura 3.2 se muestra el esfuerzo aparente en función de la temperatura sin STC, mientras que la figura 3.3 se muestra el resultado con STC.

Una última consideración con respecto a la temperatura es la corriente que circula por la galga, que depende tanto de la tensión de entrada como de la resistencia nominal de la celda. Cuanto menor sea la tensión de alimentación y mayor sea la resistencia nominal de la celda, menor será la corriente que circule por la misma, produciendo un menor aumento de la temperatura.

3.2 Celda de carga

Circuito puente

Como se mencionó en la sección anterior, la celda de carga es una estructura mecánica a la que se somete a un esfuerzo y produce una tensión proporcional al esfuerzo realizado sobre la misma. Está constituida por cuatro galgas extensiométricas en configuración de puente, como se muestra en la figura 3.4.

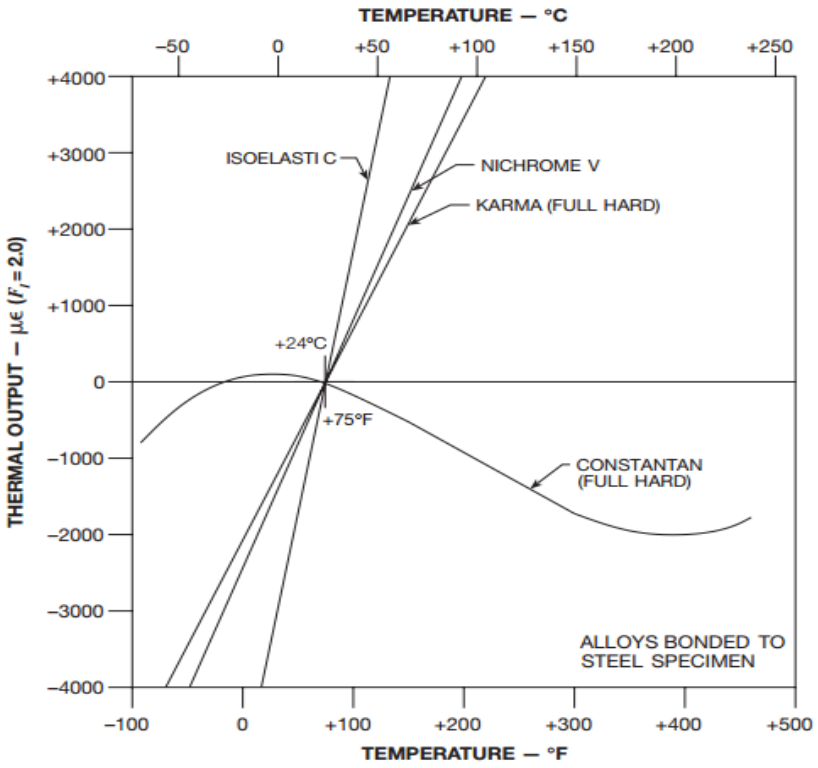


Figura 3.2: Esfuerzo aparente vs Temperatura sin STC.

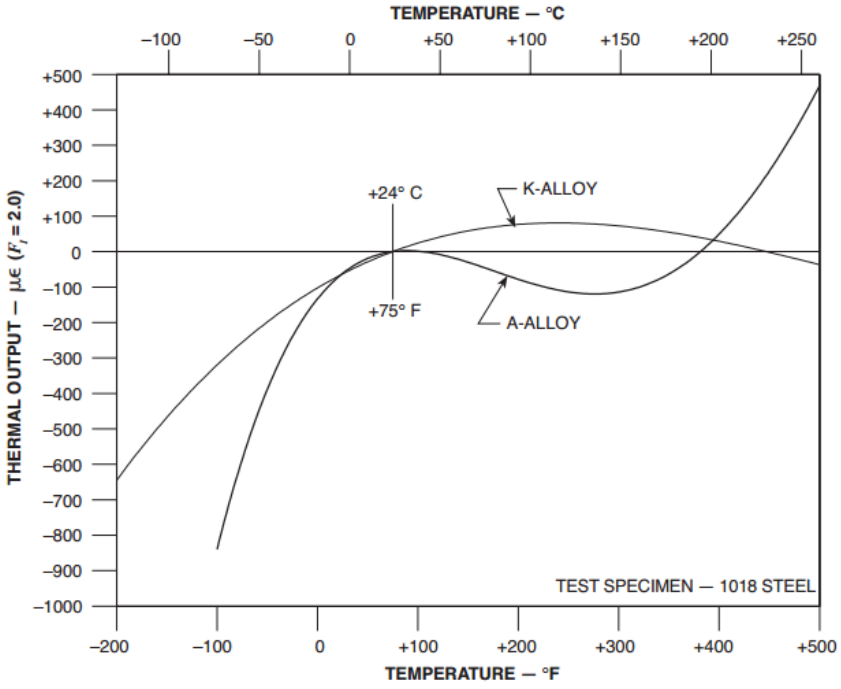


Figura 3.3: Esfuerzo aparente vs Temperatura con STC.

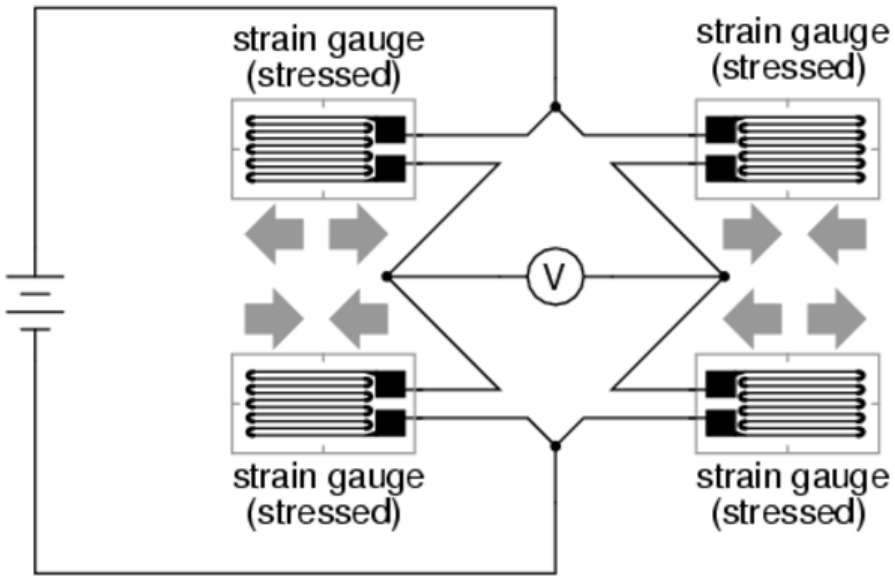


Figura 3.4: Puente completo.

A esta configuración se la llama de puente completo, ya que las cuatro resistencias que componen el circuito son galgas activas. Existen otras dos configuraciones, de cuarto de puente y de medio puente. La configuración de puente completo es la mejor por dos motivos. Uno es la linealidad, debido a que la tensión de salida es directamente proporcional al esfuerzo realizado. Y el otro es la mayor sensibilidad que se obtiene a medida que aumentan las galgas activas en el circuito, muy importante, ya que como se explicó anteriormente el esfuerzo realizado provoca muy pequeños cambios en la tensión de salida. Las otras dos alternativas son válidas para aquellas aplicaciones donde no sea posible la instalación de dos pares de galgas complementarias, como lo es el caso del puente completo.

El puente tiene dos puntos de entrada, donde se aplica la tensión de alimentación, y dos puntos de salida, que en forma diferencial entregan la tensión de salida con la información del esfuerzo. En la ecuación 3.5 se muestra la expresión para la tensión de salida:

$$V_0 = V_{IN}\frac{\Delta R}{R} \qquad (3.5)$$

Donde V_0 es la tensión diferencial entre los dos puntos medios del puente, V_{IN} es la tensión de alimentación, R es la resistencia que presenta en reposo la galga y ΔR es la variación de resistencia que se produce debido al esfuerzo aplicado a la celda de carga. Si introducimos la ecuación 3.2 en la 3.5, resulta la ecuación 3.6:

$$V_0 = V_{IN} * GF * \varepsilon \qquad (3.6)$$

En esta expresión queda clara la relación entre la tensión de salida y el esfuerzo aplicado sobre la celda. También se demuestra lo importante que es la tensión de entrada, ya que es necesario conocer su valor con gran precisión para obtener mediciones confiables del esfuerzo. Por esta razón es que se utilizó una referencia de precisión para alimentar la celda de carga.

Selección de la celda de carga

Dentro de los distintos tipos de celdas de carga disponibles, para facilitar su instalación, se optó por una celda tipo S, que puede utilizarse tanto para tensión como para compresión. En la figura 3.5 se muestra un diagrama donde se puede observar la posición de las cuatro galgas extensiométricas en la celda tipo S, mientras que en la figura 3.6 se muestra la imagen de la celda de carga utilizada.

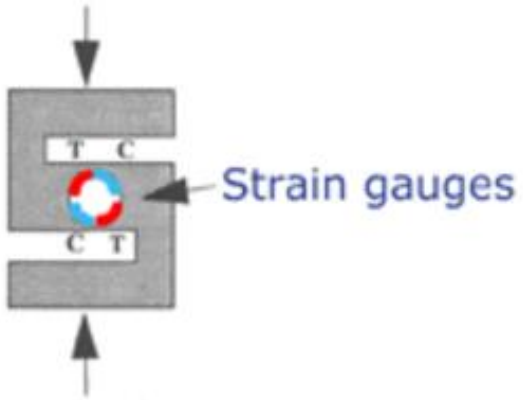


Figura 3.5: Diagrama de la posición de las galgas en una celda tipo S.

Figura 3.6: Celda de carga CZI-50.

El modelo seleccionado fue el CZI-50 de la empresa Argentina Flexar SRL, debido a que presentaba ventajas en cuanto al costo, con respecto a modelos importados, y las dimensiones, ideales para el espacio destinado en la máquina para la instalación de la celda. A continuación se

detallan las características más importantes del informe de ensayos, realizados según la norma OIMRL R-60, del fabricante de la celda CZI-50 utilizada:

- Capacidad máxima de 50Kg: Se requerían medir esfuerzo de hasta 30Kg, con lo cual los 50Kg son más que suficientes y además se trabajó en la zona más lineal de la celda.
- Sensibilidad de 2.88mV/V: La sensibilidad es relativamente alta, con respecto a los valores típicos de 2mV/V para otros modelos.
- Balance de cero de 0.057mV/V. Este es nivel de offset que presenta la salida de la celda cuando no hay carga.
- Resistencia nominal de 350Ω para las galgas extensiométricas que componen el puente.
- Tensión máxima de excitación de 15V.
- Desde -10°C hasta 40°C la celda de carga presenta compensación ante variaciones en la temperatura.
- La alinealidad mide la diferencia algebraica entre la salida para una carga específica y el punto correspondiente en una línea recta entre carga mínima y máxima. Se expresa en % de la carga nominal (% CN), y para esta celda es 0.03.
- Creep (% de CN en 20') de 0.03. Aquí se mide el cambio en la medición de salida en función del paso del tiempo, manteniéndose constantes el resto de las condiciones.
- Sobrecarga admisible de (% CN) de 150. Esta es la máxima carga que se puede aplicar sin cambiar las características de la celda.
- Carga limite (% CN) de 300. Una carga mayor a esta producirá fallas estructurales.
- Corrimiento del offset por temperatura (% CN/°C) de 0.003. Este es el nivel de desbalance que se produce ante variaciones de la temperatura, siempre que se esté dentro del rango de temperatura con compensación.
- Corrimiento de la sensibilidad por temperatura (% CN/°C) de 0.0015. Cambio en la sensibilidad debido a variaciones de la temperatura, siempre que se esté dentro del rango de temperatura con compensación.
- Resistencia de aislación de 63450MΩ. Esta es la resistencia de continua que hay entre el circuito de la celda y la estructura de la misma.
- Impedancia de entrada de 386.9Ω. Aquí se mide la resistencia entre los terminales de excitación, sin carga y con los terminales de salida a circuito abierto.
- Impedancia de salida de 350.2Ω. Esta es la resistencia entre los terminales de salida, sin carga y con los terminales de excitación en circuito abierto.

Idealmente, las resistencias de entrada y salida deberían ser iguales, pero esto no sucede ya que para compensar los cambios que se producen en las mediciones con la temperatura los fabricantes suelen colocar resistencias en serie con los terminales de excitación.

3.3 Tensión de alimentación

Como se mencionó anteriormente, la tensión de alimentación de la celda de carga es un muy parámetro muy importante, ya que la medición en la salida es directamente proporcional a

este parámetro. Como consecuencia es necesario definir a la tensión de entrada con precisión para obtener lecturas correctas del esfuerzo, motivo por el cual se utilizó una referencia de precisión.

Existe una relación de compromiso para la selección de este valor. Por un lado cuanto más alto sea, mayor será la tensión a fondo de escala y menor deberá ser la ganancia del amplificador. Aunque, también será mayor la potencia disipada en las galgas, produciendo mayores corrimientos por aumento de temperatura. Por esto último, se recomienda utilizar una tensión de excitación menor a la máxima, de 15V en este caso. En conclusión se optó por un valor de 10V, debido a que se tiene un margen con respecto al máximo, pero también se obtiene una tensión de fondo de escala más que aceptable. Otro factor importante para esta elección fue la gran cantidad de referencias de precisión disponibles en el mercado de este valor.

Ya con la tensión de alimentación adoptada es posible calcular la corriente que circulara por el puente, dada por la ecuación 3.7:

$$I_L = \frac{V}{R_N} = \frac{10V}{350\Omega} = 28.57mA \qquad (3.7)$$

Este parámetro será importante en el momento de definir el dispositivo que se usara como referencia de precisión.

Referencias de precisión: Serie vs Shunt

Ya con el valor definido para la tensión de entrada, el siguiente paso era determinar la referencia de precisión a utilizar. Existen dos tipos de referencias, series y shunt. Analizaremos las dos variantes para definir cuál es más conveniente para esta aplicación, comenzando la referencia tipo serie.

El funcionamiento de una referencia serie es muy similar al de un regulador lineal. Como se muestra en el diagrama de la figura 3.7, la referencia está en serie con la carga y puede modelarse como una resistencia controlada por la tensión.

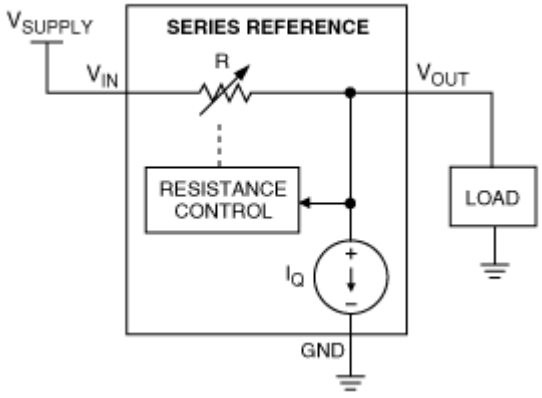


Figura 3.7: Diagrama de una referencia serie.

Se regula la caída de tensión en esta resistencia serie de forma tal que la tensión de alimentación menos dicha caída, sea igual a la tensión de salida. La tensión de entrada debe ser siempre superior a la tensión de salida más la mínima caída de tensión permitida por la referencia serie. La expresión para la tensión de salida se muestra en la ecuación 3.8:

$$V_0 = V_{IN} - R_{SERIE} * I_L \qquad (3.8)$$

Donde V_0 es la tensión de salida, V_{IN} es la tensión de alimentación, R_{SERIE} es la resistencia variable con la que se modela a la referencia serie e I_L es la corriente en la carga.

El diseño para una referencia serie es muy simple. Debe verificarse que la tensión de entrada y la potencia disipada en la referencia no superen el máximo especificado para el dispositivo. Para calcular la potencia disipada se utiliza la ecuación 3.9:

$$P_{MAX} = \left(V_{IN(MAX)} - V_0\right) * I_{L(MAX)} + I_Q * V_{IN(MAX)} \qquad (3.9)$$

Donde I_Q es la corriente de reposo que toma el dispositivo para asegurar la regulación en caso de que se quite la carga en la salida, este valor es muy pequeño. Esta es una ecuación de peor caso, donde debe tenerse en cuenta cualquier transitorio o pico en la tensión de entrada y la máxima carga en la salida.

La referencia serie tiene como gran ventaja la precisión, además de un coeficiente térmico muy bajo. Mientras que su principal desventaja es la poca corriente de carga que maneja el dispositivo, muchas veces obligando a añadir un buffer al circuito.

Por otro lado tenemos a la referencia shunt, cuyo funcionamiento es muy similar al de un diodo zener, solo que con características mucho mejores. Como un diodo zener, requiere de una resistencia externa que genere una corriente de alimentación y opera en paralelo a la carga, como puede observarse en la figura 3.8.

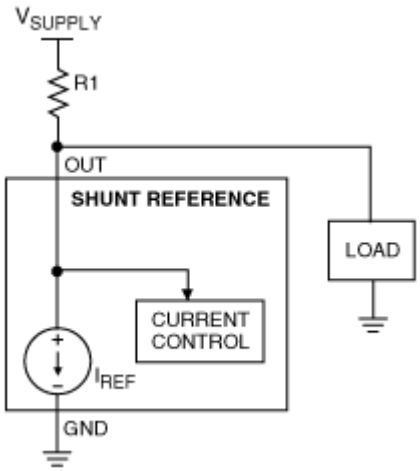


Figura 3.8: Diagrama de una referencia shunt.

Puede ser modelada como un generador de corriente controlado por tensión. Para regular ajusta la corriente de forma tal que la tensión de alimentación menos la caída en R_1, resulte igual a

la tensión de referencia. Otra forma de verlo es que, para mantener constante la tensión de salida, la referencia shunt obliga a que la suma de la corriente en la carga y en la misma referencia se mantenga constante.

El diseño es un poco más complicado, con respecto a una referencia serie, ya que debe dimensionarse correctamente la resistencia externa. Esta resistencia debe elegirse de modo tal que su caída de tensión, producida por la suma de la corriente en la carga y en la referencia, sea igual a la diferencia entre la tensión de alimentación y la tensión de salida. En la ecuación 3.10 se expresan estas relaciones:

$$V_0 = V_{IN} - R_1 * (I_L + I_{SHUNT}) \qquad (3.10)$$

Donde R_1 es la resistencia externa e I_{SHUNT} es la corriente la referencia. En cuanto a la disipación de potencia, se plantea el peor caso tanto para R_1 en la ecuación 3.11 como para la referencia shunt en la ecuación 3.12.

$$P_{R_1} = {(V_{IN(MAX)} - V_0)^2}\Big/{R_1} \qquad (3.11)$$

$$P_{SHUNT} = V_0 * \frac{(V_{IN} - V_0)}{R_1} \qquad (3.12)$$

Cabe destacar que la potencia disipada en la resistencia externa es independiente de la corriente en la carga. Mientras que para la potencia en la referencia shunt, el peor caso se da cuando no hay carga en la salida, con lo cual toda la corriente circula por la referencia.

La referencia shunt tiene como ventaja el costo y que se puede adaptar a cualquier tensión de entrada y cualquier corriente de carga, siempre que se dimensione correctamente la resistencia externa. Esto último se cumple si tanto la tensión de alimentación como la corriente en la carga son constantes, como lo es el caso de esta aplicación, en cambio cuando presentan grandes cambios es posible que se encuentre alguna limitación. Su desventaja es la menor precisión, si se la compara con una referencia serie.

Finalmente se decidió usar una referencia shunt. Primero por una cuestión de costos, ya que una referencia serie de la corriente de carga calculada en la ecuación 3.7 cuesta el doble, con respecto al caso de una referencia shunt de la misma precisión. El dispositivo seleccionado fue el LM4040AIM3-10.0 de Texas Instruments, que cuenta con las características detalladas a continuación:

- Tensión de salida de 10V.
- Tolerancia de ±0.1%.
- Coeficiente térmico de 100ppm/°C.
- Máxima corriente de cátodo de 15mA.
- Mínima corriente de cátodo de 103µA.

Para este caso como tensión de alimentación se utilizó un valor ligeramente por debajo de los 15V, entregados por la fuente. La corriente de carga que debe manejar la referencia shunt es algo menor a 30mA. Mientras que para la corriente por la misma referencia se adoptó un valor

cercano a la mitad de su máxima corriente de cátodo, 7.5mA. Por lo tanto, la resistencia a colocar tiene un valor dado por la ecuación 3.13:

$$R_1 = \frac{(V_{IN}-V_0)}{(I_L+I_{SHUNT})} = \frac{(15V-10V)}{(30mA+7.5mA)} = 133.33\Omega \qquad (3.13)$$

Finalmente se adopta el valor comercial de 137Ω para esta resistencia.

3.4 El amplificador operacional real

Se modelará el comportamiento en continua del AO real, con el fin de comprender las limitaciones del mismo y realizar la elección del dispositivo a utilizar. Los parámetros a tener en cuenta en el diseño incluyen las corrientes de polarización, el offset de tensión en la entrada, el rechazo a señales de modo común y el rechazo a la fuente de alimentación. Estas imperfecciones en el AO surgen fundamentalmente de las diferencias que existen entre los transistores de cada uno de los pares en su etapa de entrada.

Por otro lado, hay que sumarle al análisis los efectos de la temperatura y el envejecimiento, que producen corrimientos adicionales. Si a todo esto le agregamos la impedancia de entrada finita, modelada como una resistencia y un capacitor, y la impedancia de entrada no nula, tenemos el modelo real para el AO en continua. Este modelo se puede observar en la figura 3.9.

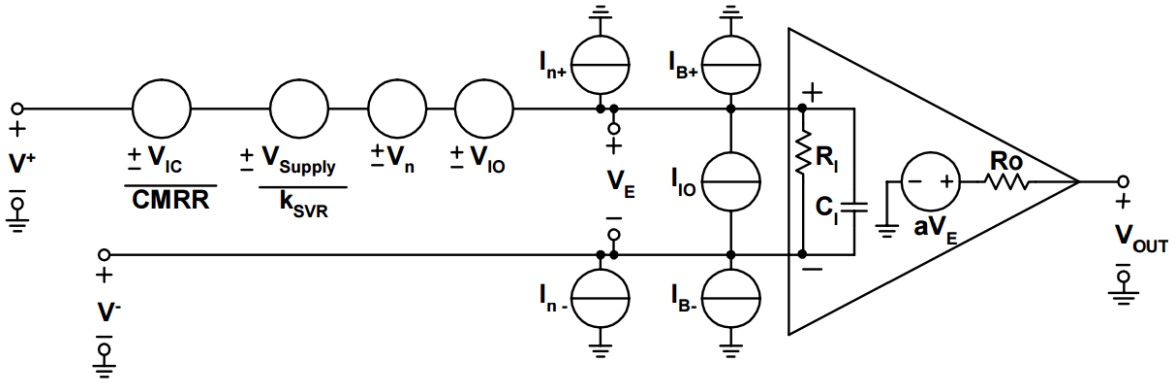


Figura 3.9: Modelo real del AO en continua.

Donde se incluyen los siguientes generadores para modelar cada uno de los parámetros del AO:

- El primero desde izquierda a derecha tiene un valor igual a la tensión de modo común sobre el rechazo de modo común.
- El siguiente modela el PSRR y tiene un valor igual a la tensión de alimentación sobre el rechazo a la fuente de alimentación.
- Luego aparecen dos generadores que modelan el offset en la tensión de entrada y su dependencia con la temperatura.

- Finalmente aparecen los generadores de corriente que modelan las corrientes de polarización en cada entrada, la corriente de offset entre los pines de entrada y la variación de estos parámetros con la temperatura.

Por último, aparecen la impedancia de entrada finita y la impedancia de salida no nula. Para minimizar estos efectos existen diversos métodos que se verán a continuación, comenzando por la compensación interna de las corrientes de polarización, ilustrado en la figura 3.10.

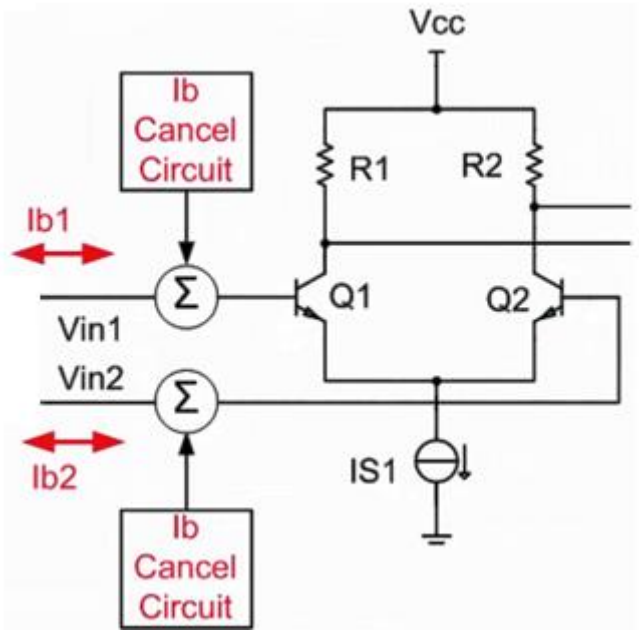


Figura 3.10: Compensación interna de las corrientes de polarización.

Si se utiliza un generador de corriente interno para suministrar la corriente de base en los transistores, es posible cancelar la corriente en los pines de entrada. De esta forma la única corriente que circulará por los terminales de entrada es la diferencia entre la corriente de base y la corriente en el generador interno de compensación, resultando muy pequeña. Esta técnica presenta varias ventajas:

- Tensión de offset muy pequeña
- Bajo corrimiento térmico
- Poca tensión de ruido
- Baja corriente de polarización
- Corriente de polarización estable con la temperatura

Aunque este método de compensación también tiene algunas desventajas, que surgen del hecho de que la corriente de polarización externa proviene de la diferencia entre la corriente de base y la corriente interna de compensación. Razón por la cual la corriente de polarización externa presenta mayor ruido y puede generar que estas corrientes circulen en direcciones contrarias.

Las hojas de datos no suelen especificar si se utiliza o no esta técnica de compensación interna y tampoco suelen ofrecer un circuito interno. Pero mirando los datos ofrecidos por el

fabricante es posible determinar si se utiliza este método o no. Uno de los indicadores es si la corriente de polarización se especifica junto con el símbolo "±", esto nos dice que se está usando compensación interna. El otro indicador es la corriente de offset, si esta tiene una magnitud similar a la corriente de polarización también se está usando compensación interna.

En cuanto al offset en la tensión de entrada, hay disponibles amplificadores operacionales llamados de auto-cero, que presentan una tensión de offset bajísima (menores a 5µV) y con un coeficiente térmico despreciable. El circuito de este AO se ilustra en la figura 3.11.

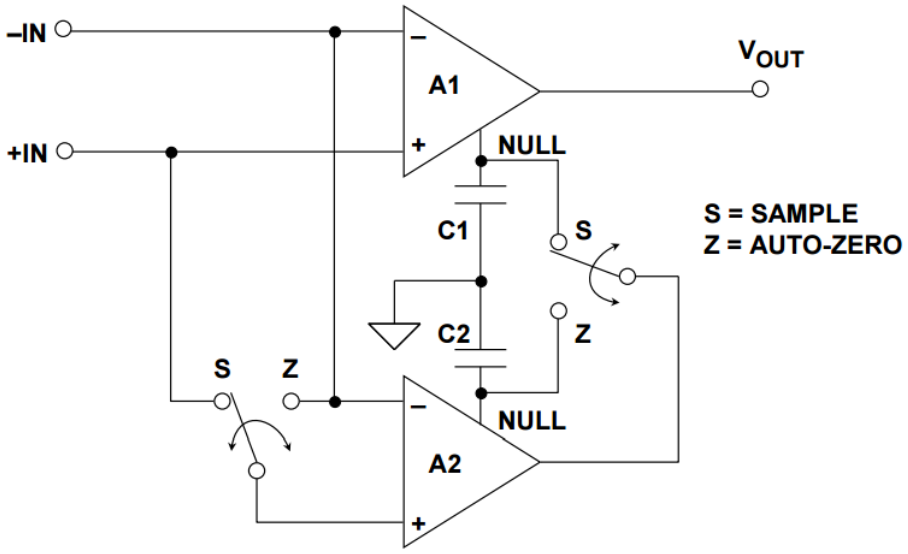


Figura 3.11: Amplificador operacional auto-cero.

En este circuito A_1 es el amplificador principal y A_2 es el amplificador de cancelación. En el modo de muestreo (con la llave en posición "S"), el amplificador de cancelación muestrea el offset en la tensión de entrada de A_1 e impone su tensión de salida en cero, aplicando la tensión adecuada en pin de NULL del amplificador principal. Aunque A_2 tiene su propia tensión de offset en la entrada que debe cancelarse, antes de corregir el offset en A_1. Esto se logra cuando la llave está en la posición "Z" (modo de auto-cero), donde se desconecta A_2 de A_1, se cortocircuito sus terminales de entrada y se conecta su salida a su propio pin NULL. Los capacitores C_1 y C_2 se utilizan para mantener la tensión de cancelación en cada uno de los amplificadores.

Otros amplificadores ofrecen pines de "nulling", donde se conecta un potenciómetro para cancelar este efecto. Se muestra el circuito interno de un AO que ofrece esta posibilidad en la figura 3.12.

El potenciómetro colocado externamente queda en paralelo con R_1 y R_2. Moviendo el potenciómetro se modifican las corrientes de colector en Q_5 y Q_6, que forman el amplificador diferencial en la etapa de entrada. Ajustando estas corrientes es posible cancelar la tensión de salida a una temperatura dada.

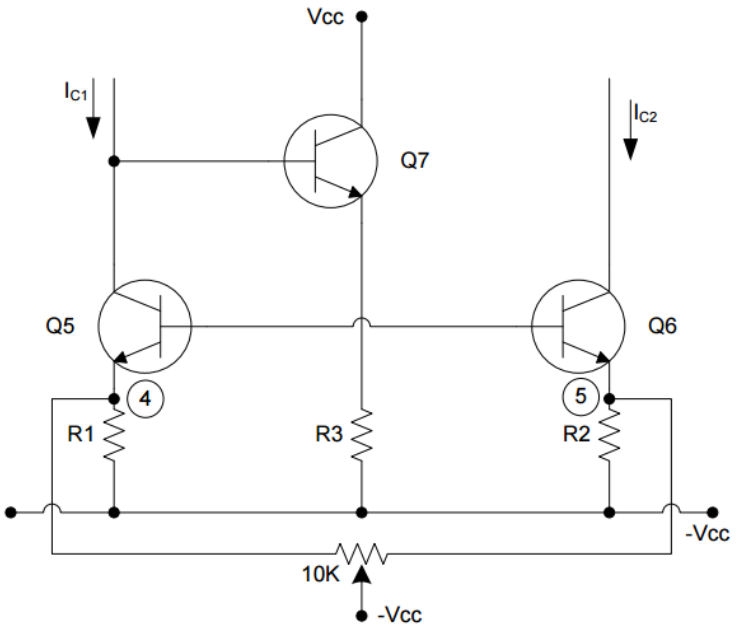


Figura 3.12: Entradas de “nullling”.

Aunque, para este amplificador se optó por otra alternativa, que consiste en ingresar una señal de compensación en uno de los terminales de entrada. En la figura 3.13 se muestran las opciones para la implementación de este método. Mediante un potenciómetro conectado a $\pm V_{CC}$ se puede inyectar una señal tanto positiva como negativa, dependiendo de la polaridad de la tensión de offset en la salida. En el circuito (A) se inyecta corriente en el pin inversor para cancelar el offset en la salida, mientras que el circuito (B) se introduce una tensión en el pin no inversor para producir el mismo efecto. La segunda variante es superior a la primera con respecto a la ganancia de ruido, ya que el primer circuito aumenta la ganancia de ruido para la señal introducida, mientras que el segundo circuito la mantiene constante. Por esto último, se eligió el circuito (B) para cancelar el offset en la tensión de salida.

Se eligió esta alternativa para cancelar el offset en la tensión de salida por que no solo compensa el offset en la tensión de entrada, sino que también se pueden compensar todos los desajustes producidos en continua.

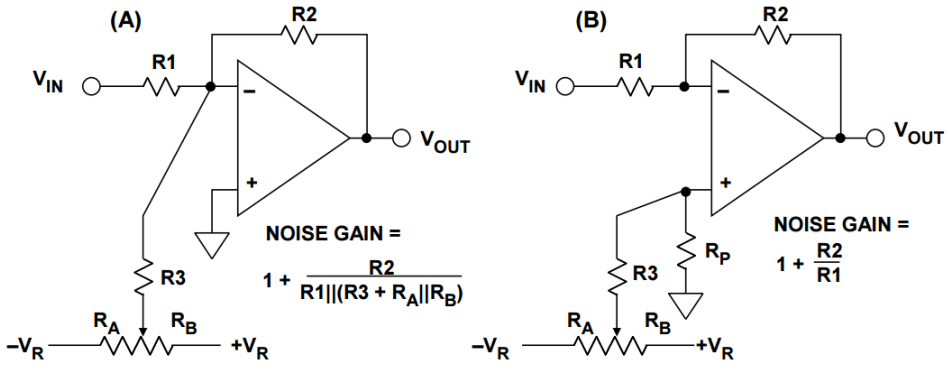


Figura 3.13: Compensación externa del offset en V_0.

3.5 Amplificador de instrumentación

Para amplificar la pequeña señal entregada por la celda de carga se requiere de un amplificador con gran ganancia diferencial y que su vez rechace cualquier señal de modo común. El tipo de amplificador que mejor se adapta a las características requeridas es el de instrumentación. El amplificador de instrumentación es un bloque con una ganancia de lazo cerrado que tiene una entrada diferencial y una sola salida con respecto al pin de referencia. La impedancia de las entradas es muy alta y balanceada, las corrientes de polarización son muy bajas y tiene una impedancia de salida muy baja.

A diferencia de un amplificador operacional, cuya ganancia de lazo cerrado está determinada por resistencias externas, el amplificador de instrumentación consta de una red interna de realimentación de resistencias que se encuentran aisladas de los terminales de entrada. La ganancia puede estar determinada internamente o de forma externa mediante un resistor.

De todas formas, existen distintos circuitos que pueden cumplir la función que cumple el amplificador de instrumentación y que se analizarán a continuación para ver qué ventajas y que desventajas presenta cada uno. Empezamos con el amplificador diferencial, cuyo esquema puede observarse en la figura 3.14.

Si R1 es igual a R3 y R2 es igual a R4, la tensión de salida está dada por la ecuación 3.14:

$$V_0 = (V_2 - V_1)\frac{R_2}{R_1} \qquad (3.14)$$

Se ve como idealmente se amplifica únicamente la diferencia entre las señales de entrada. Aunque en realidad este circuito presenta dos desventajas con respecto al amplificador de instrumentación. Una es que la impedancia de entrada de cada una de las entradas es relativamente baja, dependiendo del valor de las resistencias y más aún cuando se requieren valores de ganancia elevados, y también son desbalanceadas. El desbalance de impedancia impacta de forma negativa en el CMRR. La otra desventaja es que el rechazo de modo común es dependiente de la tolerancia de las resistencias, donde sí se calculan las ganancias de modo diferencial y modo común con resistencias de tolerancia del 0.1%, se obtiene un CMRR de 66dB.

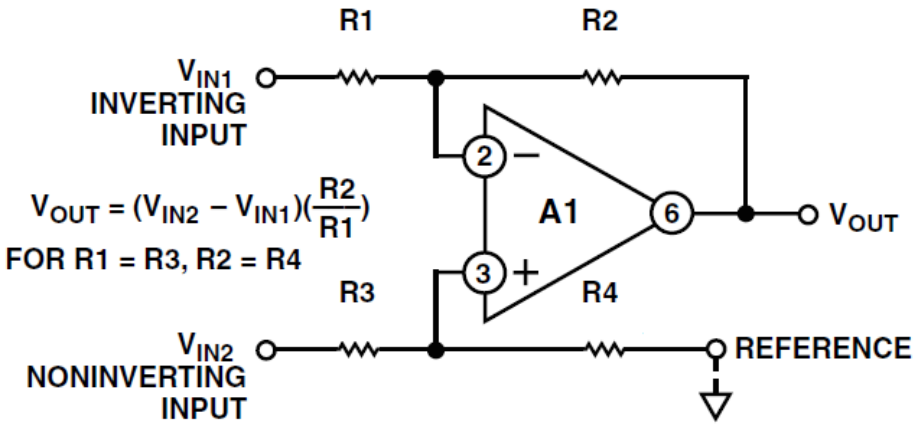


Figura 3.14: Amplificador diferencial.

Aunque es posible mejorar este circuito mediante el agregado buffers en las entradas, como se muestra en el circuito de la figura 3.15.

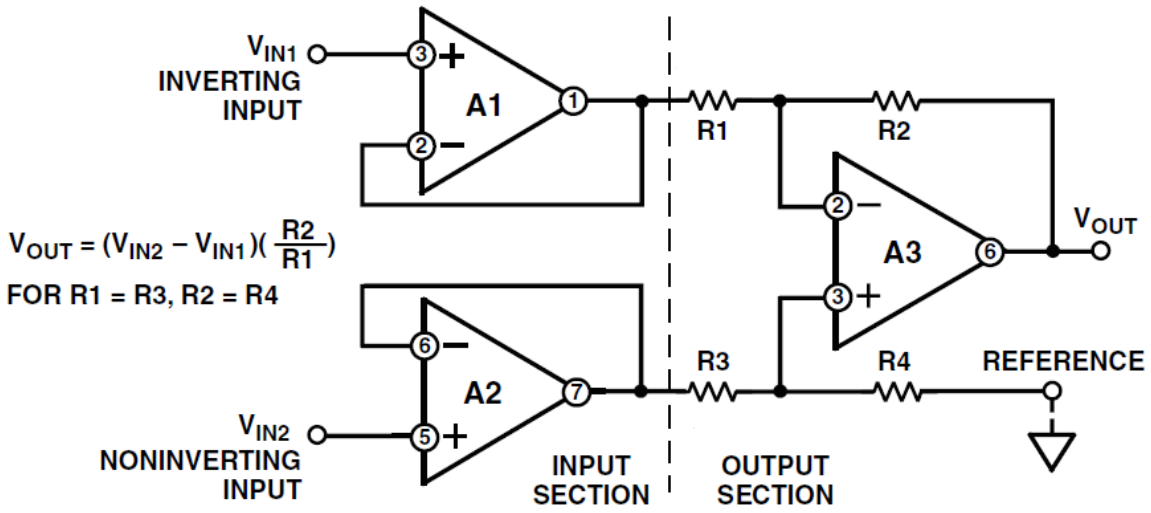


Figura 3.15: Amplificador diferencial con buffers en la entrada.

De esta forma se obtiene entradas con una impedancia de entrada muy alta y balanceada. Pero a este circuito se le puede hacer una mejora más, que se muestra en el circuito de la figura 3.16.

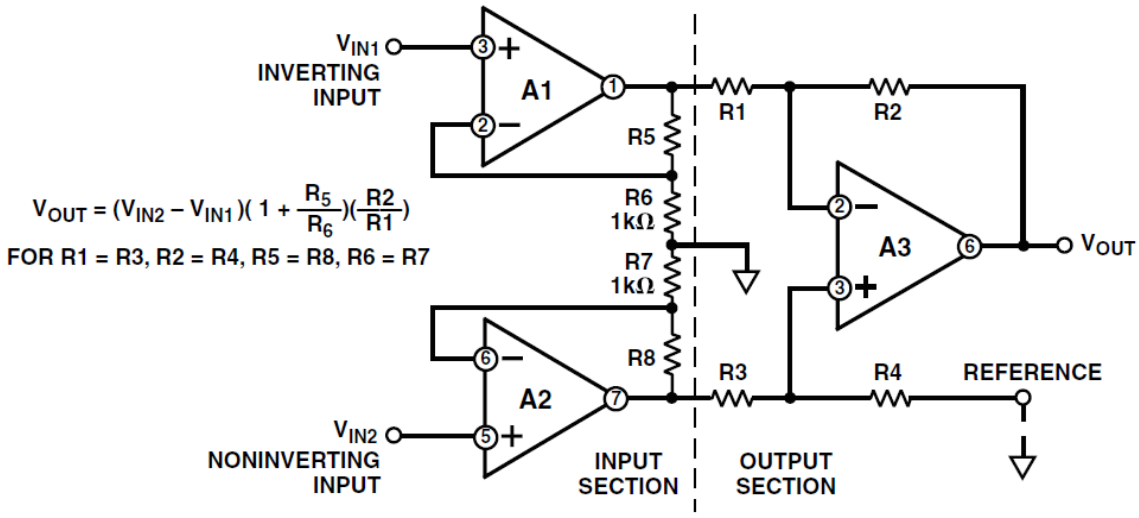


Figura 3.16: Amplificador diferencial con buffers en la entrada con ganancia.

Añadiendo R_6 y R_7 se logran que la etapa de entrada ahora tenga una ganancia adicional, de esta forma se tiene mayor flexibilidad en el diseño. Aunque, esta ganancia adicional aumenta la

ganancia de modo diferencial como la de modo común. Por este motivo, finalmente surge la configuración clásica del amplificador de instrumentación, ilustrado en la figura 3.17.

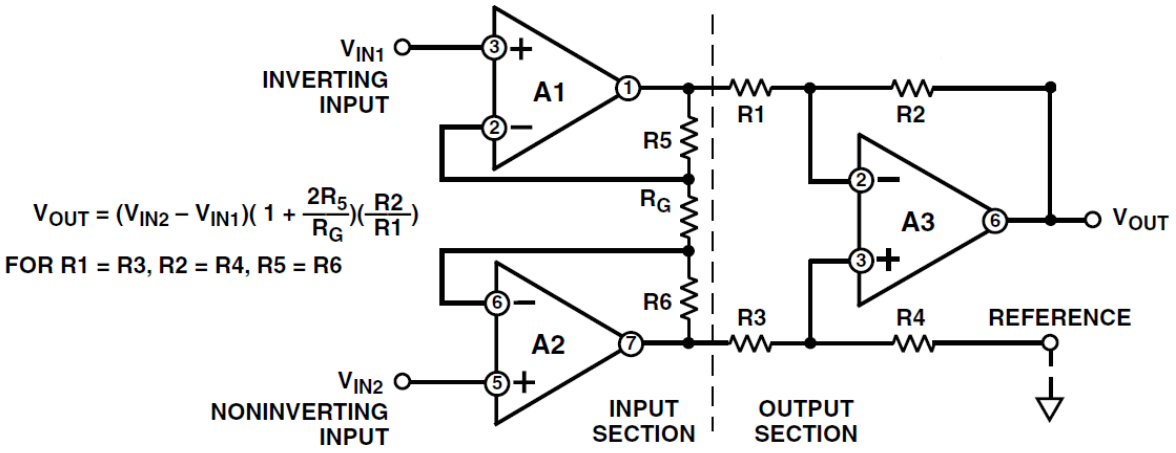


Figura 3.17: Amplificador de instrumentación.

Se reemplazan las resistencias 6 y 7 por R_G y ahora la tensión diferencial esta aplicada sobre la misma. Y como la tensión de entrada amplificada, en la salidas de los AOs 1 y 2, aparece en forma diferencial entre R_5, R_G y R_6. Con esto se consigue que se pueda variar la ganancia diferencial simplemente modificando R_G. La ganancia para este circuito está dada por la ecuación 3.15, con la codicion de que R_5 sea igual a R_6:

$$V_0 = (V_{IN2} - V_{IN1})\left(1 + \frac{2R_5}{R_G}\right) * \frac{R_2}{R_1} \qquad (3.15)$$

Como la tensión de modo diferencial es aplicada sobre R_G, si aplicamos únicamente una señal de modo común no circulara corriente en esta resistencia y los amplificadores A1 y A2 se comportaran únicamente como simples seguidores. Es decir, que la ganancia de modo de común es unitaria, mientras que la ganancia de modo diferencial en esta etapa está dada por la ecuación 3.16:

$$G_{D(IN)} = (1 + \frac{2R_5}{R_G}) \qquad (3.16)$$

Teóricamente, se puede aumentar CMRR todo lo que se desee. Solo debe usarse valores de R_G que permitan ganancias diferenciales cada vez más grandes y manteniéndose constante la ganancia de común.

También cabe destacar que debido a la simetría de esta configuración, los errores de modo común en los amplificadores de entrada, tienden a ser cancelados en el restador de salida. Esto incluye los errores producidos por la dependencia de CMRR con la frecuencia.

Existe una versión del amplificador de instrumentación con dos AOs, cuyo circuito se muestra en la figura 3.18:

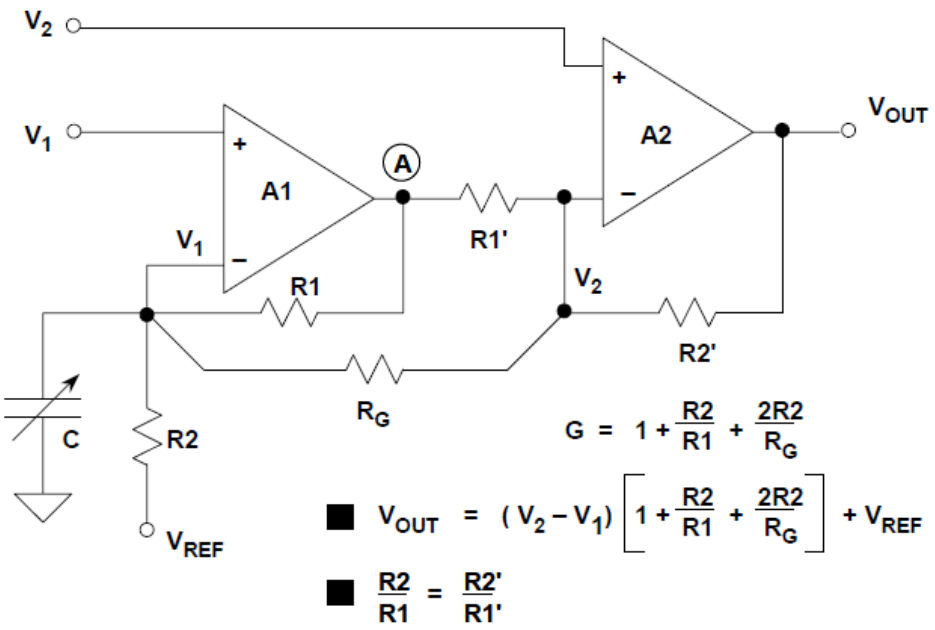


Figura 3.18: Amplificador de instrumentación con dos AOs.

Se expresa la tensión de salida, donde se ve como con la resistencia R_G se puede utilizar para ajustar la ganancia. Al igual que en el caso de la configuración con tres AO, este circuito también presenta una impedancia de entrada muy elevada y balanceada. Este circuito tiene la obvia ventaja de utilizar un AO menos, lo que se traduce en un menor costo y consumo. Aunque existe una limitación para la ganancia impuesta por el rango de tensión de modo común en la entrada. Para ganancias bajas, $R_1 >> R_2$, A1 podría saturar si la tensión de modo común es muy elevada, quitando margen para amplificar la señal diferencial. En el caso de ganancias elevadas este problema desaparece. Siguiendo con los puntos negativos de este circuito, el rechazo a señales de modo común en alterna es inferior, ya que en la señal que ingresa en A1 tiene un desfasaje adicional con respecto a la señal que ingresa en A2. Además los dos amplificadores presentan diferentes ganancias de lazo cerrado y diferente ancho de banda en consecuencia. Para aumentar el CMRR en AC se agrega un pequeño capacitor variable como se muestra en la figura.

Los amplificadores de instrumentación, en cualquiera de sus dos versiones, pueden implementarse tanto de forma discreta como de forma integrada. Esta última cuenta con todas las ventajas de los circuitos integrados, mayor facilidad en el diseño y fundamentalmente un mejor apareamiento entre los resistores y componentes que componen el amplificador. Esta última característica es la que permite obtener los rechazos de modo común más elevados. Sin embargo, implementar este circuito de forma discreta cuenta con una mayor flexibilidad.

Se hará una última consideración con respecto a los circuitos vistos hasta ahora y esta relacionando con el pin REF. La tensión ingresada a este pin será la referencia de tensión para la salida. Puede estar a masa, a la mitad de la tensión de alimentación en casos de utilizar fuente simple o usarse para cancelación de offset. Pero no se puede ingresar una tension de cualquier forma, para ilustrar esto nos referiremos a las figuras 3.19 y 3.20.

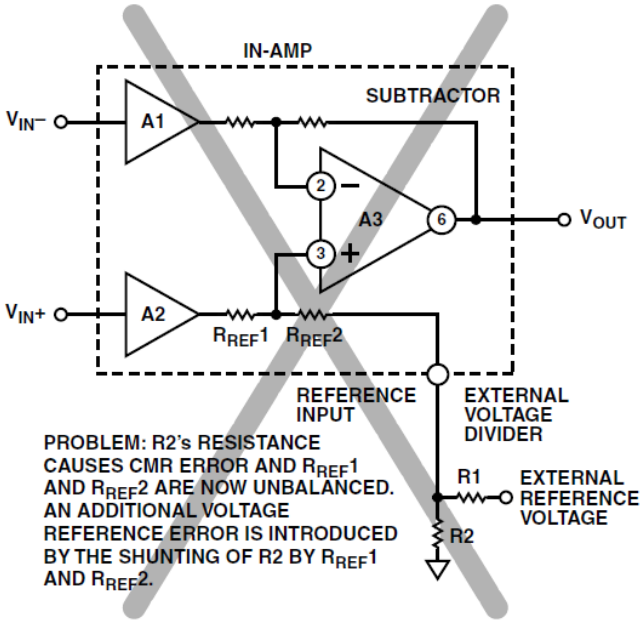


Figura 3.19: Forma incorrecta de utilizar el pin REF.

En el primer circuito se utiliza un divisor de tensión compuesto por una tensión de referencia externa, R_1 y R_2. R_2 queda en serie con una de las resistencias interna del restador, resultando un desbalance entre R_{REF1} y R_{REF2} y la disminución de CMRR. También se genera un error adicional en la tensión de referencia.

Por otro lado, en el circuito de la figura 3.20 se utiliza un AO como buffer para introducir el nivel de referencia. Como el AO presenta una impedancia de salida muy baja, no se influye en el restador de salida y por lo tanto no se generar errores como en el caso anterior.

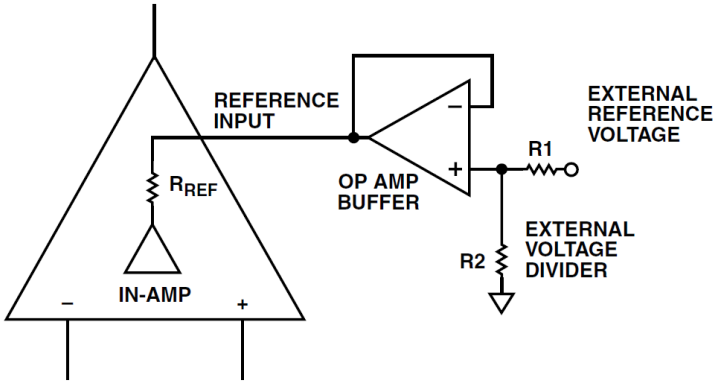


Figura 3.20: Forma correcta de utilizar el pin REF.

Finalmente, se decidió utilizar el circuito de la figura 3.21.

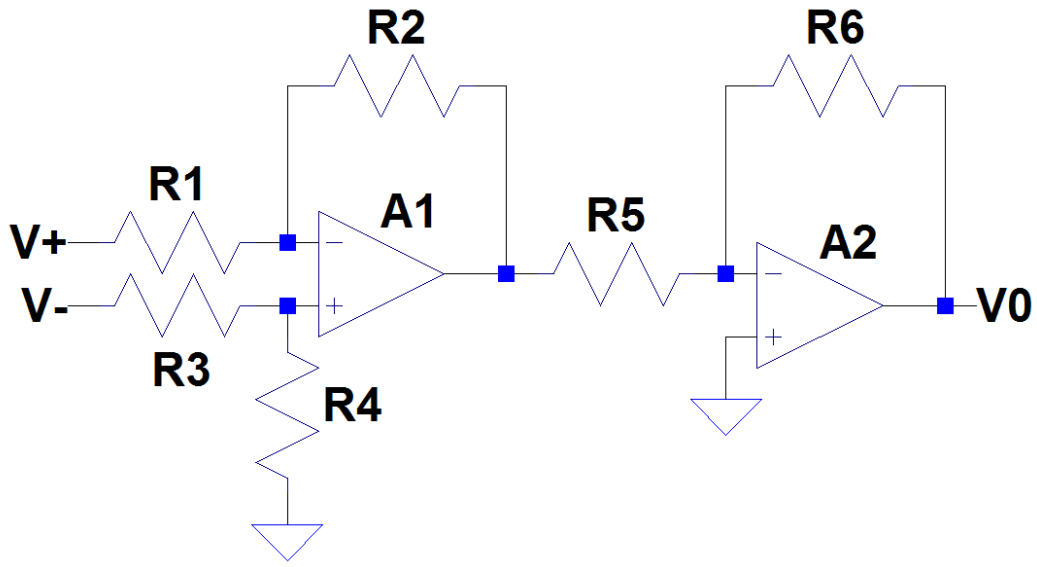


Figura 3.21: Configuración elegida para el amplificador.

Este circuito cuenta con dos etapas bien diferenciadas. La primera está compuesta por un restador puro, es decir sin amplificación, donde todas las resistencias tienen el mismo valor. Mientras que la segunda es una etapa de amplificación, donde la ganancia es determinada por la relación entre R_6 y R_5.

La etapa de entrada es como el amplificador diferencial, pero como aquí no se busca amplificación, las resistencias pueden tomar un valor elevado para mejorar la impedancia de entrada. Aunque no se llegara a los valores tan altos que se obtienen con un buffer en la entrada, además tampoco hay balance. Esta etapa es la que define el rechazo a la tensión de modo común y presenta las mismas características que el amplificador diferencial. La etapa de salida amplifica a la tensión diferencial entregada por la primera etapa, pero también amplifica a las señales de modo común. Por esto último, este circuito es inferior al amplificador de instrumentación en cuanto al CMRR, ya que en el de instrumentación la ganancia de modo diferencial no amplifica la ganancia de modo común. Sin embargo, esta configuración ofrece una gran flexibilidad en el diseño, dos etapas bien diferenciadas y una ventaja en cuanto al costo.

3.6 Diseño

Ya con la configuración del amplificador seleccionada, se procede con el diseño del mismo. Empezamos describiendo el amplificador operacional utilizado, el OP07D de Analog Devices. Esta versión del AO estándar OP07 es de precisión, offset muy bajo, bajo consumo, corrientes de polarización pequeñas, alto CMRR y alto PSRR. En la figura 3.22 se muestran las especificaciones dadas por el fabricante a temperatura ambiente con una tensión de alimentación de ±15V.

Parameter	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
INPUT CHARACTERISTICS						
Offset Voltage	V_{OS}			45	150	μV
		$0°C \le T_A \le 70°C$			250	μV
		$-40°C \le T_A \le +125°C$			350	μV
Input Bias Current	I_B			0.2	1	nA
		$-40°C \le T_A \le +125°C$			1	nA
Input Offset Current	I_{OS}			0.2	1	nA
		$-40°C \le T_A \le +125°C$			1	nA
Input Voltage Range			−13.5		+13.5	V
Common-Mode Rejection Ratio	CMRR	$V_{CM} = \pm 13.0$ V	120	140		dB
		$-40°C \le T_A \le +125°C$	120			dB
Open-Loop Gain	A_{VO}	$R_L = 2$ kΩ to ground, $V_O = \pm 11$ V	1000	10,000		V/mV
		$-40°C \le T_A \le +125°C$	1000			V/mV
Offset Voltage Drift	$\Delta V_{OS}/\Delta T$	$0°C \le T_A \le 70°C$		0.5	2.5	μV/°C
		$-40°C \le T_A \le +125°C$		0.5	1.5	μV/°C
OUTPUT CHARACTERISTICS						
Output Voltage Swing	V_{OUT}	$R_L = 10$ kΩ to ground	±13.95	±14		V
		$-40°C \le T_A \le +125°C$	±13.9			V
		$R_L = 2$ kΩ to ground	±13.75	±13.8		V
		$-40°C \le T_A \le +125°C$	±13.7			V
Short-Circuit Current	I_{SC}			30		mA
Output Current	I_O	$V_O = 13.5$ V		15		mA
POWER SUPPLY						
Power Supply Rejection Ratio	PSRR	$V_S = \pm 4.0$ V to ±18.0 V	115	130		dB
		$0°C \le T_A \le 70°C$	115			dB
		$-40°C \le T_A \le +125°C$	110			dB
Supply Current/Amplifier	I_{SY}	$V_O = 0$ V		1.1	1.3	mA
		$0°C \le T_A \le 70°C$			1.55	mA
		$-40°C \le T_A \le +125°C$			1.85	mA
DYNAMIC PERFORMANCE						
Slew Rate	SR	$R_L = 10$ kΩ		0.2		V/μs
Gain Bandwidth Product	GBP			0.6		MHz
Phase Margin				80		Degrees
NOISE PERFORMANCE						
Voltage Noise	$e_{n\,p\text{-}p}$	0.1 Hz to 10 Hz		0.25		μV p-p
Voltage Noise Density	e_n	f = 1 kHz		10		nV/√Hz
Current Noise Density	i_n	f = 1 kHz		0.074		pA/√Hz

Figura 3.22: Especificaciones del 0P07D.

Seguimos con la etapa de entrada, donde se requiere determinar el valor para las resistencias que componen el restador. Para obtener un valor alto en la impedancia de entrada se requiere un valor elevado en estas resistencias. Sin embargo, la impedancia de entrada del AO, del orden los MΩ, impone un límite para este valor. Con lo cual se elige utilizar resistencias de 100KΩ.

Resta definir la ganancia en la etapa de salida. Este parámetro está dado por el nivel máximo de la señal a amplificar y los 3.3V de referencia que utiliza el ADC. Si bien la celda de carga permite una carga de hasta 50Kg, solo se medirá hasta 30Kg. Por ende, se calcula la señal entregada por la celda para 30Kg mediante la ecuación 3.17:

$$V_{CELDA(MAX)} = Sensibilidad * V_{IN} * \frac{30Kg}{50Kg} = 2.88\frac{mV}{V} * 10V * \frac{30Kg}{50Kg} = 17.28mV \qquad (3.17)$$

Aunque este nivel de tensión no se amplificó hasta 3.3V, se dejó un margen. De esta forma se podrá compensar el offset a la salida del amplificador, ya sea producto de la celda o el

amplificador, por software. Antes de aplicar la carga durante el ensayo se registrará la salida del amplificador, este valor idealmente es cero. Sin embargo, en la práctica esto no es así, se registrará un valor distinto de cero. El nivel registrado puede ser debido al desbalance en el circuito puente de la celda, que además es amplificado, o puede ser producto de los errores de continua analizados previamente en el amplificador o ambos. Por lo tanto, se adopta una ganancia de 180, resultando la máxima tensión a la salida del amplificador como se expresa en la ecuación 3.18:

$$V_{0(MAX)} = V_{CELDA(MAX)} * G = 17.28mV * 180 = 3.1104V \qquad (3.18)$$

Para obtener este valor de ganancia, la relación entre las resistencias en la etapa de salida debe ser de 180. Como R_5 está a la salida de un AO no puede tomar valores pequeños, ya que se podría sobrepasar el nivel de corriente máxima en la salida. Pero a su vez R_6 no puede tomar valores muy grandes, por la misma razón expuesta para la selección de las resistencias del restador. Con esto en consideración se adoptó un valor de 180KΩ para R_6 y 1KΩ para R_5.

Todas las resistencias analizadas anteriormente se seleccionaron de 0.1% de tolerancia, para tener un mejor rechazo a las señales de modo común.

En la etapa de salida se agregó un capacitor en paralelo a R_6 para filtrar ruido. Añadiendo este capacitor se genera un polo en la transferencia de lazo cerrado que atenúa las componentes de alta frecuencia. Se define el ancho de banda de interés, rango de frecuencia donde la ganancia permanece constante e igual a 180, a un 1Hz. Este valor pequeño permite filtrar ruido, fundamentalmente el ruido de línea, y también la adquisición de una gran cantidad de datos. El capacitor comercial resultante es de 820nF, que recalculando da una frecuencia de corte de 1.078Hz.

Por otra lado, también debe agregarse capacitores de desacople en la alimentación. Ya que el rechazo a la fuente de alimentación es función de la frecuencia, como se muestra en el gráfico de PSRR vs frecuencia para el OP07D en la figura 3.23.

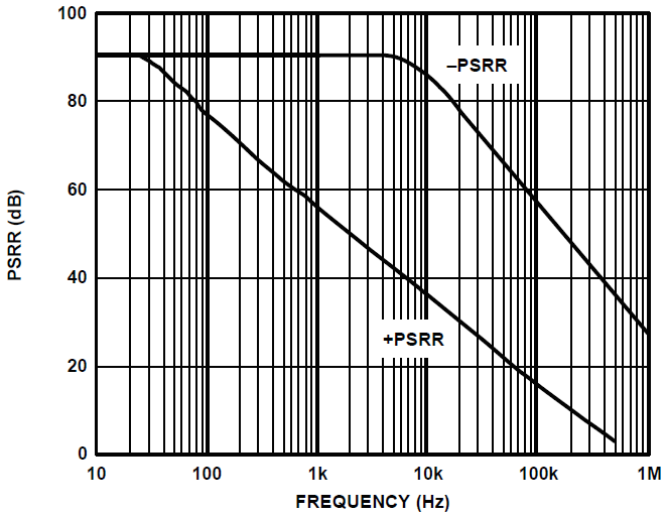


Figura 3.23: PSRR vs frecuencia para el OP07D.

Si bien para el ancho de banda dado para este circuito es muy bajo, la frecuencia de línea y sus armónicos también producen errores en la tensión de alimentación, por lo que es necesario añadir capacitores de desacople para mejorar el rechazo a frecuencias superiores.

Para bajas frecuencias se utiliza un capacitor de gran valor, generalmente electrolítico, mientras que para altas frecuencias se usa un capacitor más chico con una buena respuesta en frecuencia, generalmente cerámico. Para este amplificador se optó por capacitores cerámicos de 10µF para rechazar las bajas frecuencias, ya que estos tienen una mejor respuesta en frecuencia que los electrolíticos. Para altas frecuencias se utilizaron capacitores de 0.1µF cerámicos. También hay que considerar el diseño de la PCB, ya que para desacople correcto los caminos entre los capacitores y el pin de alimentación del integrado debe ser lo más corto posible para reducir las inductancias serie. Además se deben conectar los capacitores a un plano de masa de baja impedancia.

Finalmente, en la entrada no inversora del amplificador en la etapa de salida se incorporó un circuito que permita el ajuste del offset. Dicho circuito se puede observar en la figura 3.24, donde aparece el amplificador completo.

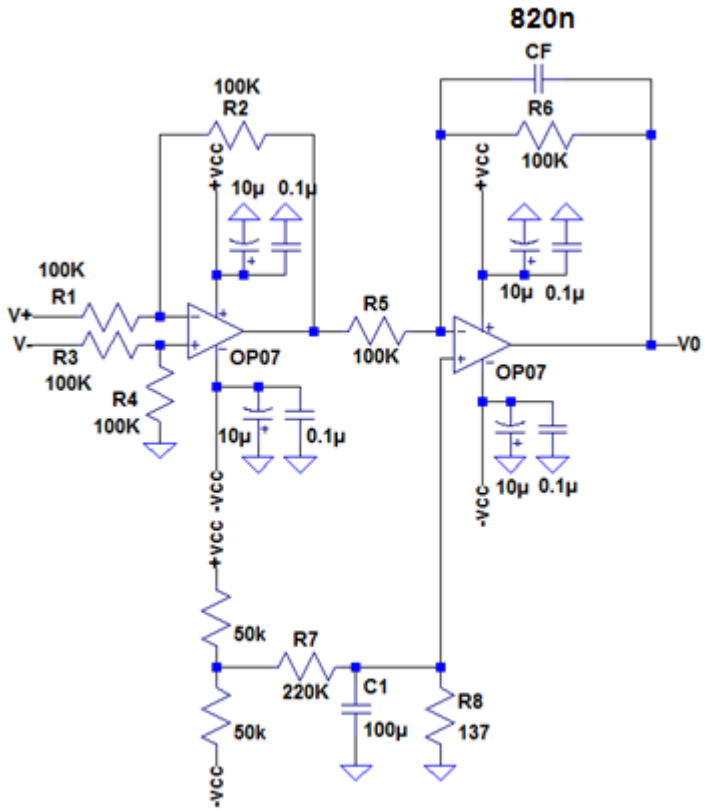


Figura 3.24: Amplificador completo.

El circuito está compuesto por un potenciómetro de 100KΩ a $\pm V_{CC}$, para poder cancelar salidas tanto positivas como negativas, y un filtro. Este filtro sirve para eliminar el ruido granular del potenciómetro. La función de R_8 es reducir la ganancia, de lo contrario ante una pequeña variación

en el potenciómetro aparecería un nivel de tensión muy alto en la salida, debido a la alta ganancia del amplificador. Como esta tensión es un valor de continua puro, se toma un capacitor de valor elevado, de 100µF.

Si bien se compensará el offset en la salida por software, si el nivel de tensión sin carga resulta muy alto se podría perder rango dinámico en el ADC. Por ende, en un primer momento con la celda de carga alimentada pero sin carga en la misma se realizará una medición de la tensión de salida. Se tratará llevar a cero la salida ajustando el potenciómetro. Una vez realizado este ajuste, la compensación se hará únicamente por software. Donde se registrará la salida sin carga, luego a las mediciones con carga se le restará el valor de la primera medición.

3.7 Limitador de salida

Para proteger la entrada del ADC se incorporó un limitador para la tensión de salida del amplificador. El circuito utilizado para este propósito es mostrado en la figura 3.25.

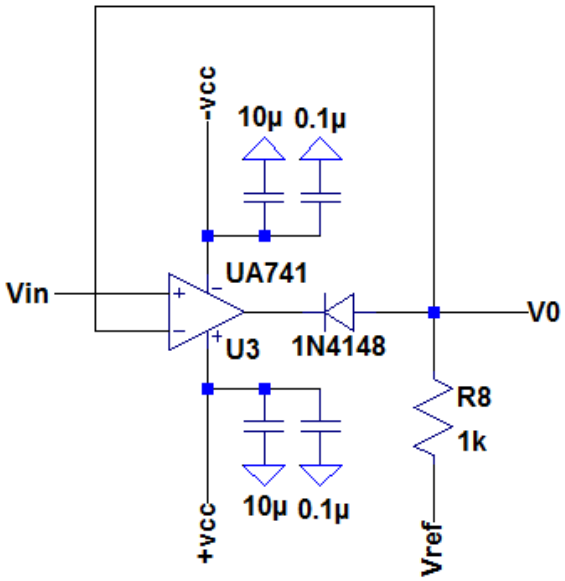


Figura 3.25: Limitador.

Este circuito está compuesto por un amplificador operacional, un diodo y una resistencia puesta a la tensión de limitación. La ventaja que presenta este limitador con respecto a otros que utilizan diodos es que no se tiene en cuenta la caída de tensión de directa del mismo, ya que esta tensión está dividida por la elevada ganancia del amplificador operacional. De esta manera se logra limitar de forma muy precisa a la tensión que ingresa en el pin no inversor del AO a un nivel igual a

la tensión de referencia de 3.3V. No se requieren características especiales para el AO, por lo que se optó por un amplificador operacional de propósitos generales, el µA741.

3.8 Simulación

A continuación se presentará el resultado de las simulaciones realizadas en el programa LTSPICE. El circuito completo de la simulación se puede apreciar en la figura 3.26.

Se incluyó la referencia de precisión shunt LM4040 en su versión de 10V para generar la tensión de excitación para la celda de carga. También aparece el circuito puente, formado por cuatro resistencias de valor nominal de 350Ω. En estas resistencias se incluyó una expresión en función de la fuerza aplicada a la carga que determina el cambio en el valor de la resistencia. Luego se toma la salida diferencial del circuito puente y se ingresa en el amplificador de instrumentación. Finalmente seguido del amplificador aparece el limitador, cuya salida es la que lee el ADC del microcontrolador.

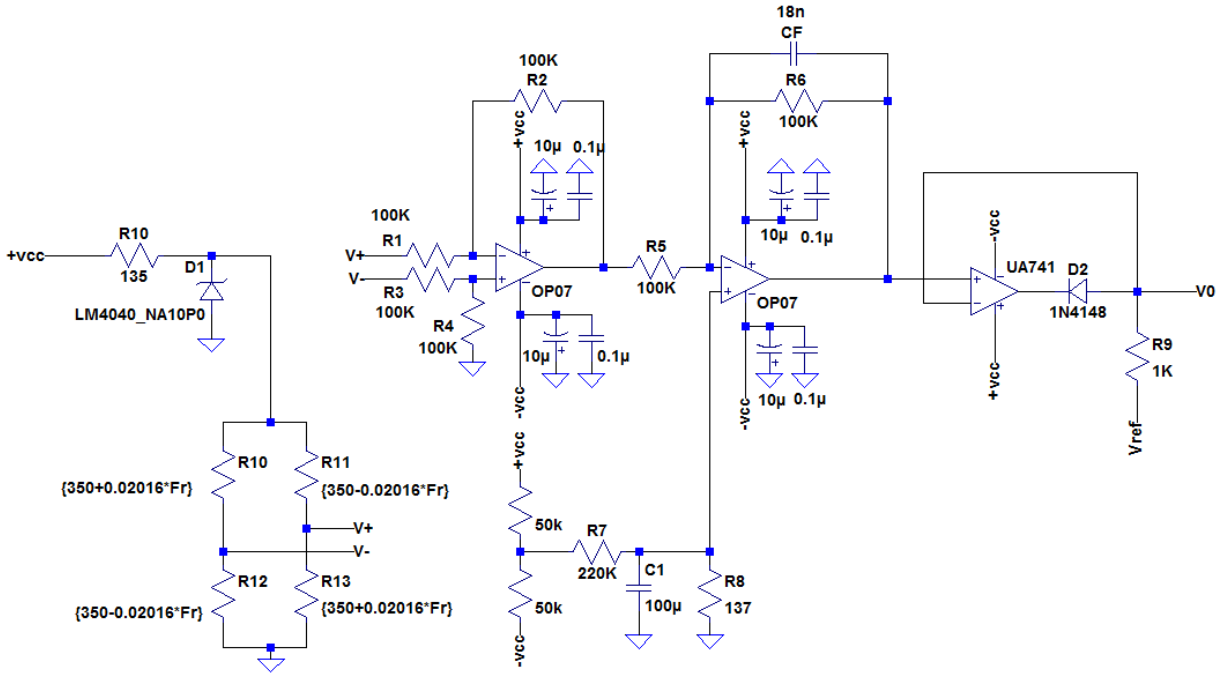


Figura 3.26: Circuito utilizado para la simulación.

Empezaremos con el análisis del circuito que genera la tensión de referencia de 10V. La tensión de entrada de 15V es suministrada por un generador de tensión en la simulación. En la figura 3.27 se muestra la salida del regulador shunt en verde, mientras que en el otro grafico se muestran las corrientes por la resistencia, el regulador shunt y la celda de carga.

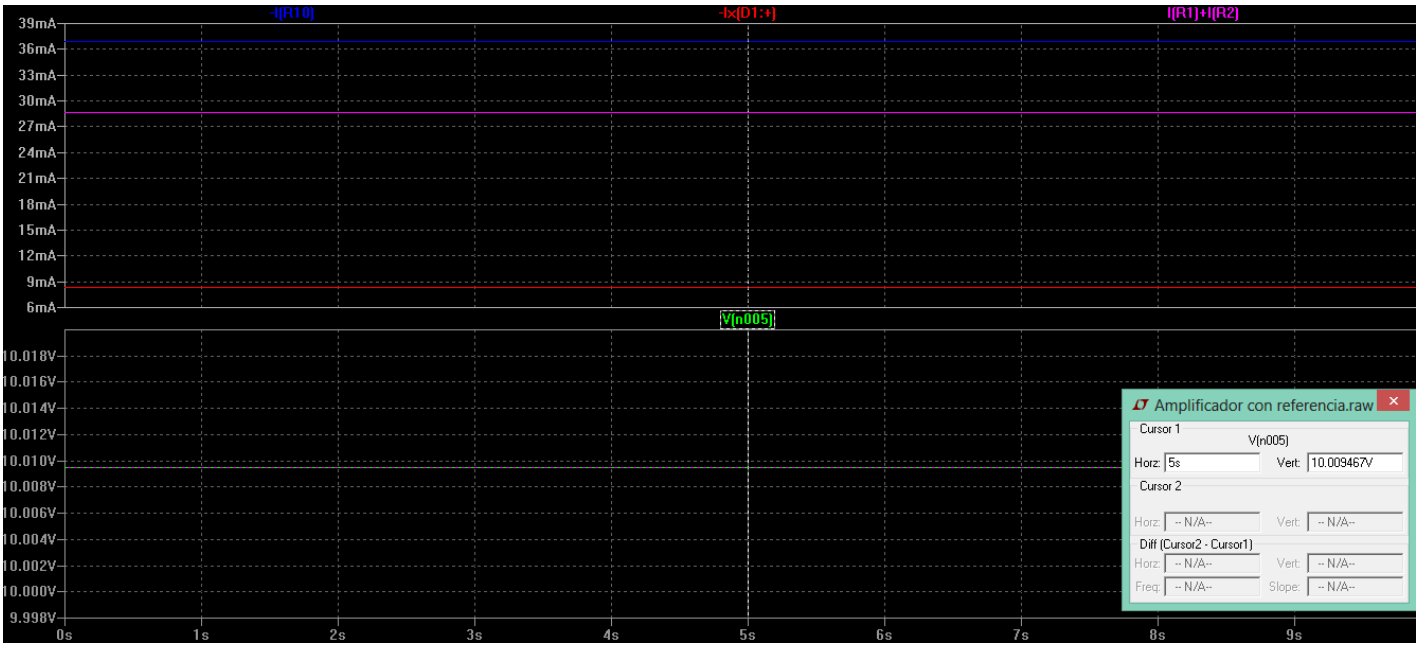


Figura 3.27: Formas de onda del regulador shunt.

La mayor de las corrientes es la que circula por la resistencia, es igual a la suma de las otras dos y se mide un valor de 37mA. Mientras que la menor es la que circula por el regulador con nivel de 8.33mA, restando la corriente de consumo en el puente que es 28.63mA.

Seguimos el análisis en el amplificador, cuyas formas de onda se puede apreciar en la figura 3.28 para una fuerza aplicada a la celda de 15Kg. En el gráfico de abajo aparecen las tensiones de salida del puente. Estas señales presentan un pequeño corrimiento, positivo en una de las entradas y negativo en la otra, con respecto a su tensión de modo común de 5V. En el medio se muestra la salida de la primera etapa, que elimina la tensión de modo común dejando solo la señal diferencial, con un valor medido de 8.63mV. Finalmente la tensión de salida se muestra de rosa, resultando con un nivel de 1.55V. Recordemos que se instrumentó la celda para medir cargas de hasta 30Kg, produciendo una salida máxima de 3.11V. En esta simulación se ingresó con la mitad de la carga máxima y se obtuvo la mitad de la tensión de salida máxima, con lo cual se comprueba que la salida varía en función de la carga aplicada según lo diseñado.

Por último, se muestra cómo actúa el circuito limitador. En la simulación anterior, donde la tensión que ingresa al limitador es menor a la tensión de referencia, este circuito no influye. Por otro lado, ahora se presentan las formas de onda del limitador cuando se ingresa con una fuerza de 50Kg, superior a los 30Kg máximos permitidos, en la figura 3.29. En verde aparece la entrada del circuito limitador, salida del amplificador de instrumentación, con un nivel superior a los 5V. En cambio la tensión de salida, la señal de azul en el gráfico, está limitada a los 3.3V que se aplican en la tensión de referencia. De esta manera se demuestra el correcto funcionamiento de este circuito utilizado para limitar la tensión que ingresa en el ADC del microcontrolador.

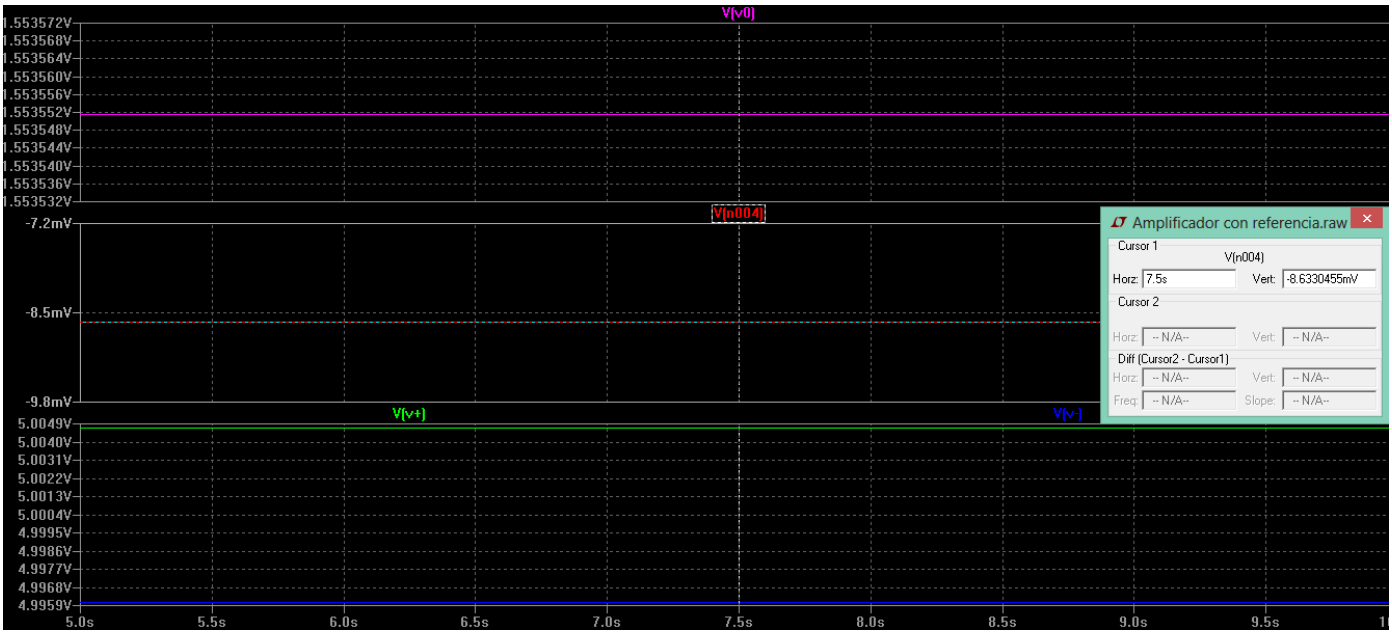


Figura 3.28: Formas de onda en el amplificador de instrumentación.

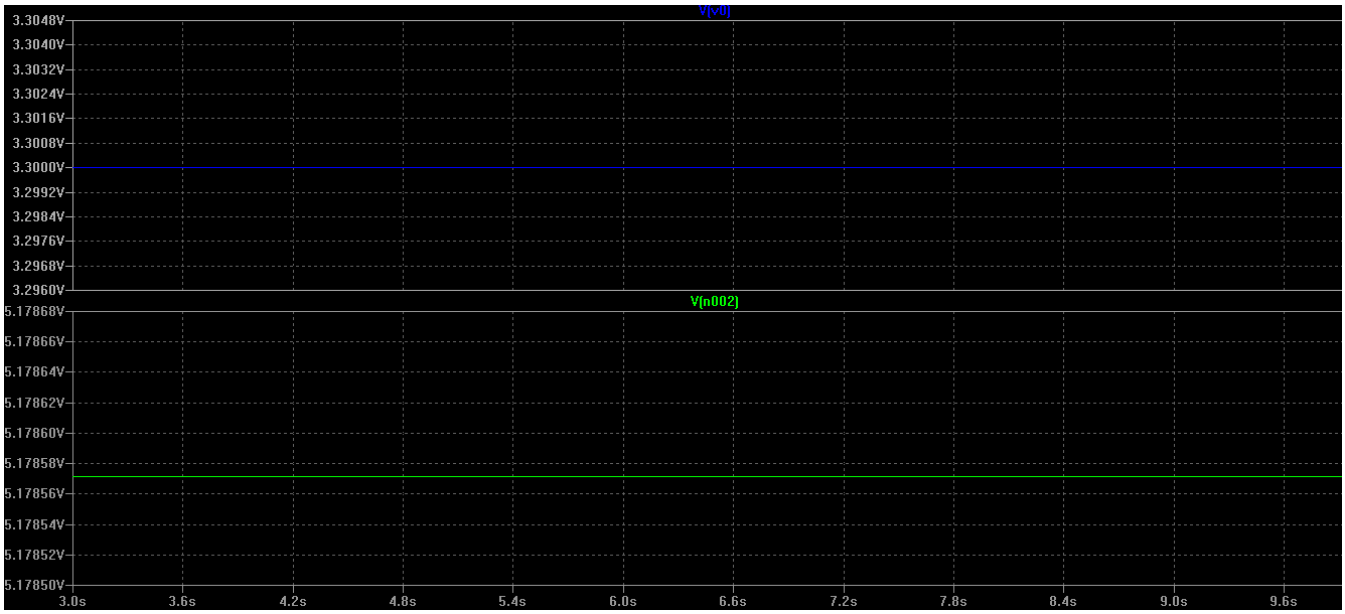


Figura 3.29: Formas de onda en el limitador.

En conclusión, el resultado de las simulaciones respalda el diseño.

4

Variador de velocidad

4.1 Motor Asincrónico

Principio de operación y construcción

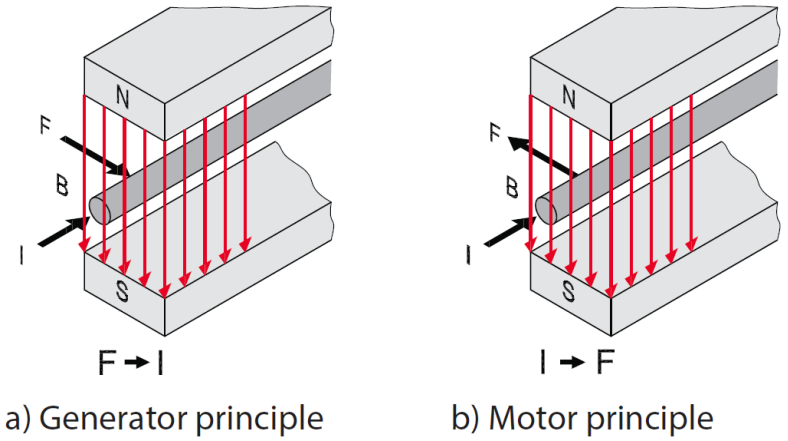
Antes de comenzar con la descripción y selección del variador de velocidad, se hará un breve análisis del motor asincrónico para comprender como es el control del mismo.

Existe una gran cantidad de motores eléctricos que se dividen entre los motores de continua y de alterna. A su vez, los de alterna se dividen entre sincrónicos y asincrónicos. A esta última categoría pertenece el motor instalado en la máquina, también llamado motor de inducción.

Los motores de alterna pueden convertir energía eléctrica en mecánica, operando como motor, o viceversa, operando como generador, por medio de inducción electromagnética. El principio de inducción electromagnética nos dice que si un conductor se mueve, producto de la aplicación de una fuerza, a través de un campo de inducción magnético, se induce una tensión en el conductor. Si el conductor es parte de un circuito cerrado, una corriente fluirá a través de él.

El principio del generador se aplica cuando por medio del movimiento se induce tensión en el conductor, como se muestra en la figura 4.1.a. Mientras que el principio del motor se da cuando se coloca un conductor, por el cual circula una corriente, en un campo magnético. En esta situación, el conductor se mueve hacia afuera del campo magnético producto de una fuerza que actúa sobre él, como se ilustra en la figura 4.1.b.

El campo magnético en el motor es generado por la parte estacionaria, el estator, y el conductor, en el que se inducen las fuerzas electromagnéticas, se encuentra en la parte giratoria, el rotor.



a) Generator principle b) Motor principle

Figura 4.1: Principio de inducción electromagnética.

Para generar el campo magnético el estator utiliza bobinas, que se conectan a la fuente de alimentación trifásica de alterna. El núcleo del estator está compuesto por láminas de acero-silicio de elevada permeabilidad, de reducido espesor y apiladas hasta obtener la longitud necesaria, con el objeto de reducir las corrientes de Foucault. Luego, con conductores aislados, se bobina en las ranuras del núcleo. Las laminaciones que forman el núcleo y el estator ya bobinado se muestran en la figura 4.2.

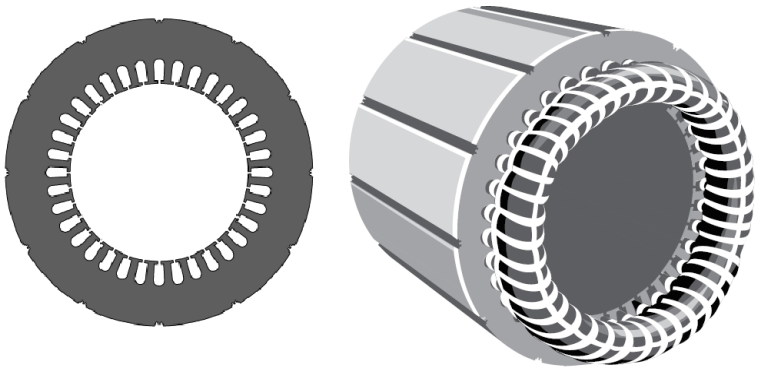


Figura 4.2: Laminaciones para el núcleo y el estator.

La conexión de las bobinas y la red trifásica, para el caso de un motor de dos polos, se indica en la figura 4.3. Los bobinados A1 y A2 se conectan a la fase A de la red, B1 y B2 a la fase B y C1 y C2 a la fase C. Se ve como los bobinados A1, B1 y C1 de encuentran 120° separados uno del otro, lo mismo sucede con A2, B2 y C2, esto se debe al desfasaje eléctrico que existe entre las fases de la red. Es un motor de dos polos debido a que la bobina de cada fase tiene dos polos.

Para mostrar cómo se genera el campo magnético giratorio se recurre a la figura 4.4, donde se muestra la dirección resultante del campo magnético a lo largo de un periodo.

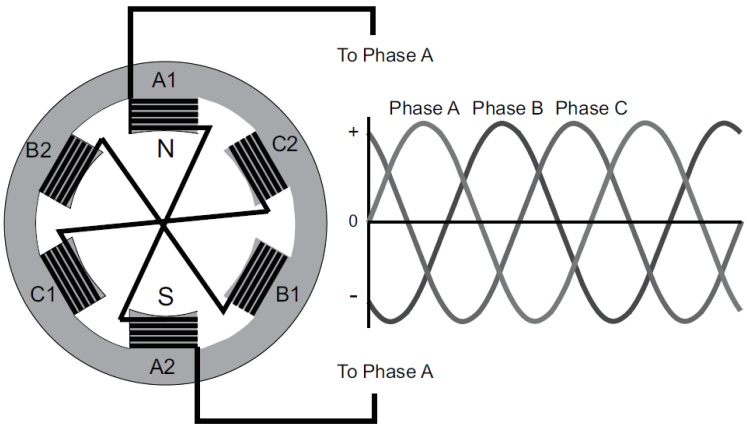


Figura 4.3: Conexión del estator a la red.

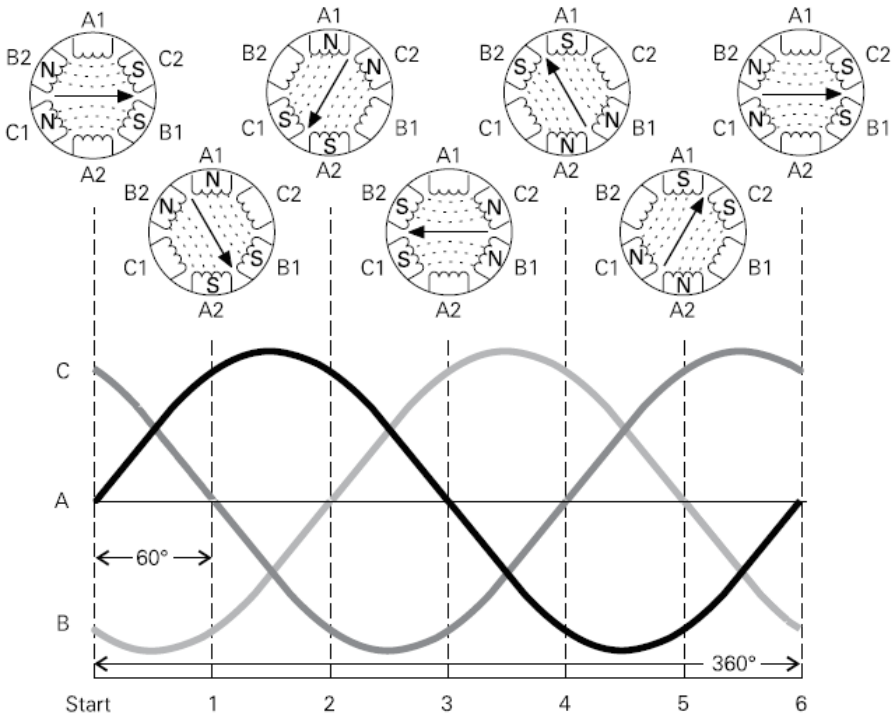


Figura 4.4: Campo magnético a lo largo de un periodo.

El resultado de esta disposición de arrollamientos en el estator conectados a una terna simétrica de tensiones es un campo denominado rotante, de amplitud esencialmente constante y dirección variable, que gira a la frecuencia de línea. A la velocidad del campo magnético giratorio se la llama velocidad sincrónica. Esta velocidad depende de la frecuencia de la línea y el número de pares de polos del motor, esta relación se muestra en la ecuación 4.1:

$$N_S = \frac{60*f}{P} = \frac{60*50HZ}{1} = 3000RPM \qquad (4.1)$$

Aquí se expresa la velocidad sincrónica en RPM y se evalúa para un motor de dos polos, un par de polos.

Como se mencionó anteriormente los motores de alterna se dividen en sincrónicos y asincrónicos. El estator funciona de la misma manera en ambos tipos, pero en cuanto al diseño y el movimiento del rotor en relación con el campo magnético difieren. En el caso de motores sincrónicos la velocidad del rotor coincide con la del campo magnético, por otro lado, en un motor asincrónico esta relación no se cumple.

En cuanto al rotor, el tipo más común en motores de inducción es el de jaula de ardilla. De la misma forma que el estator, el núcleo del rotor también está construido con laminaciones. En las ranuras de las laminaciones se colocan barras conductoras, que son cortocircuitadas mediante un aro de terminación. El eje del motor se encastra en rotor. Las laminaciones del núcleo y el rotor se muestran en la figura 4.5.

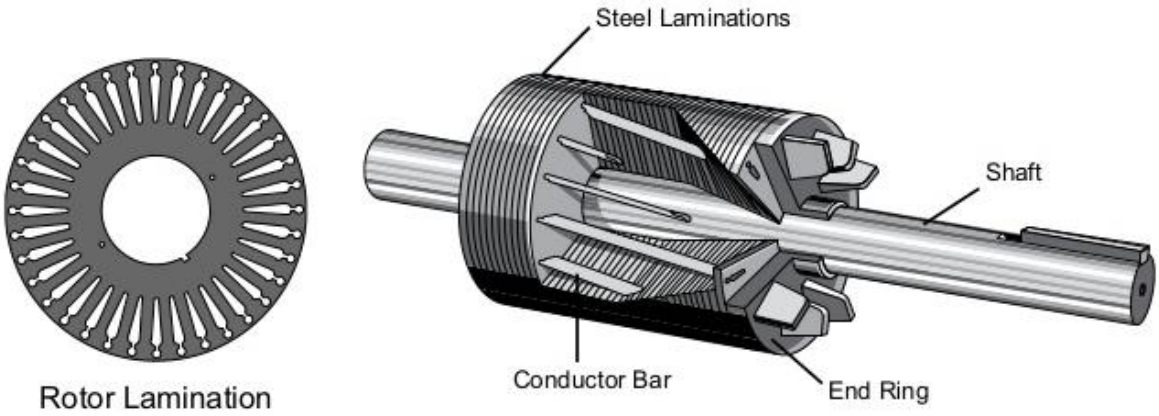


Figura 4.5: Laminaciones para el núcleo y el rotor.

Para demostrar cómo se produce el movimiento en el rotor se recurre a la figura 4.6.

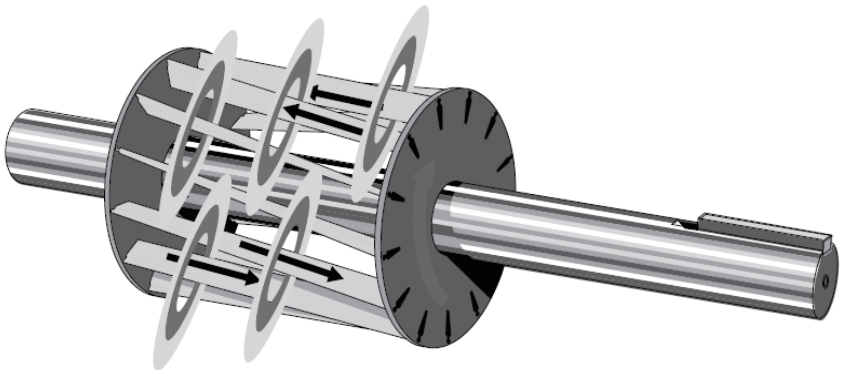


Figura 4.6: Rotor en movimiento.

Cuando el campo magnético generado por el rotor atraviesa las barras conductoras del rotor, se induce una tensión en las mismas. El circuito se cierra mediante el aro de terminación y

fluye una corriente a través de las barras conductoras, generando que el rotor se comporte como un electroimán. El campo magnético en las bobinas del estator ejerce una fuerza de atracción con algunas barras y al mismo tiempo fuerza de repulsión con otras. Esta es la causa del movimiento giratorio en el rotor, aunque no a la velocidad sincrónica. Por ley de Faraday es necesario que el flujo que atraviesa las espiras del rotor varíe en el tiempo y por lo tanto se pueda inducir corriente en el mismo. A esta diferencia de velocidad de la denomina deslizamiento y se expresa en la ecuación 4.2:

$S = N_0 - N_S$ (4.2)

Este parámetro es variable con el punto de operación del motor. Cuando el motor opera a su velocidad nominal el deslizamiento relativo es mínimo, mientras que cuando se reduce la velocidad del rotor el deslizamiento relativo aumenta.

Torque

La fuerza que actúa sobre las barras del roto está dada por la ecuación 4.3:

$F = B \times I_W * L$ (4.3)

Donde B es la densidad de flujo magnético, I_W es la corriente inducida en las barras y L es la longitud de las barras. Esta fuerza genera un torque en el eje del motor, como se puede apreciar en la figura 4.7.

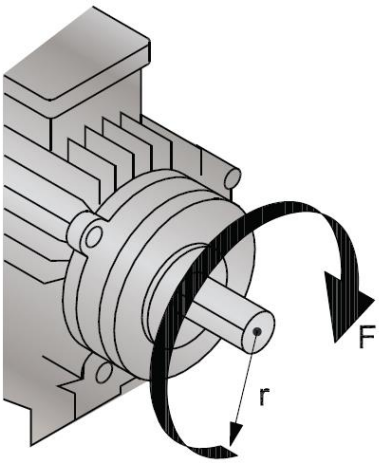


Figura 4.7: Torque en eje del motor.

El torque está dado por la ecuación 4.4:

$T = F \times r$ (4.4)

El toque que puede producir un motor está relacionado con la potencia del mismo. Para deducir esta relación, primero planteamos el trabajo realizado por el motor en la ecuación 4.5:

$W = F \times d = F \times (2\pi * r * n)$ (4.5)

Donde d es la distancia que tira el motor para una carga dada, que se relaciona con el radio y la cantidad de vueltas, n. También planteamos la ecuación 4.6, la relación entre la potencia y el trabajo:

$$W = P * t \qquad (4.6)$$

De la ecuación 4.5 se despeja la fuerza y se la introduce en la ecuación 4.4, a su vez se reemplaza al trabajo por la ecuación 4.6, resultando la ecuación 4.7:

$$T = \frac{W}{d} \times r = \frac{P*t*r}{2\pi*r*n} \qquad (4.7)$$

Si ahora se evalúa para t igual a 60 segundos, n es ahora la velocidad en revoluciones por minuto y el torque resulta como se muestra en la ecuación 4.8:

$$T = \frac{P*9550}{N} \qquad (4.8)$$

Donde N es la velocidad a la que gira el eje del motor en RPM y P la potencia expresada en KW.

Por otro lado, en la figura 4.8 aparece el grafico del torque y de la corriente en el motor en función de la velocidad del motor.

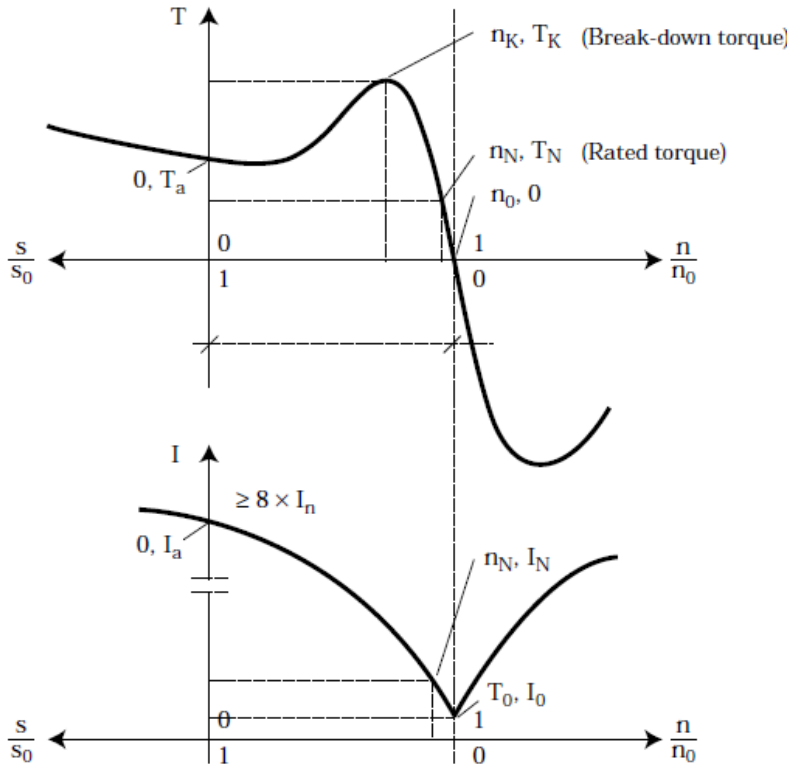


Figura 4.8: Torque y corriente en función de la velocidad del motor.

Existe una zona de operación normal y otras dos de frenado. En el rango de $n/n_0>1$, donde el motor gira por encima de la velocidad sincrónica el motor opera como generador. Creando un

torque opuesto y devolviendo su salida a la red eléctrica. En la zona de $n/n_0<0$, al frenado se lo llama regenerativo.

La zona de operación normal, que es con $0<n/n_0<1$, se divide en dos. Si $0<n/n_0<n_K/n_0$, se está en la zona de arranque, mientras que $n_K/n_0<n/n_0<1$ es la zona de operación del motor. Destacaremos algunos puntos importantes del gráfico, comenzado con T_A, que es el torque de arranque cuando con el motor en reposo se alimenta con tensión nominal y frecuencia nominal. En estas mismas condiciones en el gráfico de la corriente se observa corriente de arranque, I_A, muy por arriba de la nominal. T_K es el máximo torque que puede generar el motor cuando es alimentado con tensión y frecuencia nominal. T_N es el torque nominal, este es el punto de operación óptimo del motor cuando se lo conecta a la red eléctrica.

Eficiencia

La potencia que el motor toma de la red es mayor a la que el motor es capaz de entregar a la salida. Esto está relacionado con las pérdidas que se producen en el mismo, y se lo cuantifica mediante la eficiencia, dada por la expresión de la ecuación 4.9:

$$\eta_M = \frac{P_0}{P_{IN}} \qquad (4.9)$$

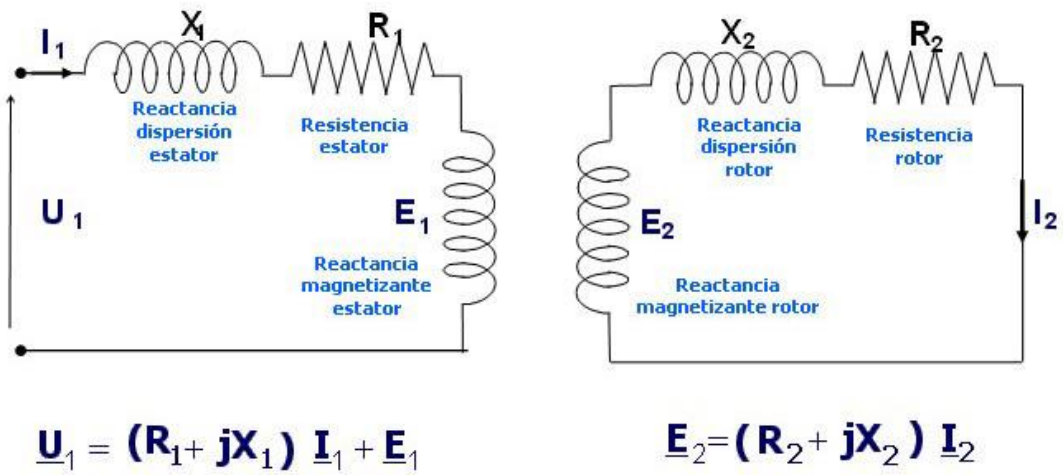
Las principales causas de las pérdidas en el motor son las siguientes:

- Pérdidas en el cobre: Debido a la resistencia de los bobinados del estator.
- Pérdidas en el núcleo: Se dividen en dos, por histéresis y por las corrientes de Eddy. Las pérdidas por histéresis se generan por someter al núcleo a un flujo de alterna, donde se magnetiza y desmagnetiza constantemente al núcleo. La magnetización y des magnetización requiere de energía, que es tomada de la red y aumenta a medida que también lo hace la frecuencia. Las corrientes de Eddy son generados por la tensión inducida en el núcleo. La circulación de estas corrientes produce disipación de calor, pérdidas.
- Pérdidas en el ventilador: Ocurren por la viscosidad del aire en el ventilador.
- Pérdidas por fricción en el rotor.

Circuito equivalente

La transferencia de energía en un motor asíncrono se produce de estator a rotor por inducción electromagnética de forma análoga a la transferencia en el transformador, pero ha de tenerse en cuenta que cuando el motor gira las frecuencias en el estator y rotor son diferentes.

El circuito equivalente para cada fase de un motor asíncrono trifásico con el rotor detenido, se obtiene de forma análoga al transformador. Designando con los subíndices 1 y 2 las magnitudes del estator y rotor respectivamente, el esquema equivalente es el mostrado en la figura 4.9.



$$\underline{U}_1 = (R_1 + jX_1)\,\underline{I}_1 + \underline{E}_1 \qquad \underline{E}_2 = (R_2 + jX_2)\,\underline{I}_2$$

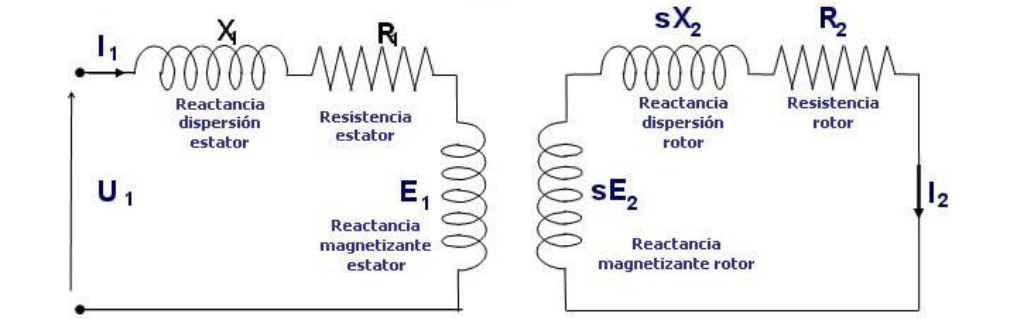
Figura 4.9: Circuito equivalente con el rotor detenido.

En el modelo con el rotor en movimiento aparece una dependencia con el deslizamiento, debido a la diferencia de frecuencia entre estator y rotor, resultando el circuito de la figura 4.10.

Con el rotor parado: $E_2 = K\,N\,f_2\,\Phi_m$ como $f_2 = f_1 \Rightarrow E_2 = K\,N\,f_1\,\Phi_m$

Con el rotor girando: $E_{2s} = K\,N\,f_2\,\Phi_m$ como $f_2 = s\,f_1 \Rightarrow E_{2s} = K\,N\,s\,f_1\,\Phi_m = s\,E_2$

Como la reactancia es función de la frecuencia $\Rightarrow X_{2s} = s\,X_2$



$$\underline{U}_1 = (R_1 + jX_1\underline{I}_1) + \underline{E}_1 \qquad s\,\underline{E}_2 = (R_2 + sjX_2)\underline{I}_2$$

Figura 4.10: Circuito equivalente con el rotor en movimiento.

Sin embargo, acomodando la expresión de la corriente en el rotor se obtiene el modelo mostrado en la figura 4.11. Donde la resistencia que aparece modela tanto la resistencia del bobinado del rotor como la resistencia de la carga mecánica. Por lo tanto, operando algebraicamente como se muestra en la figura se obtiene el segundo circuito mostrado en la figura 4.11.

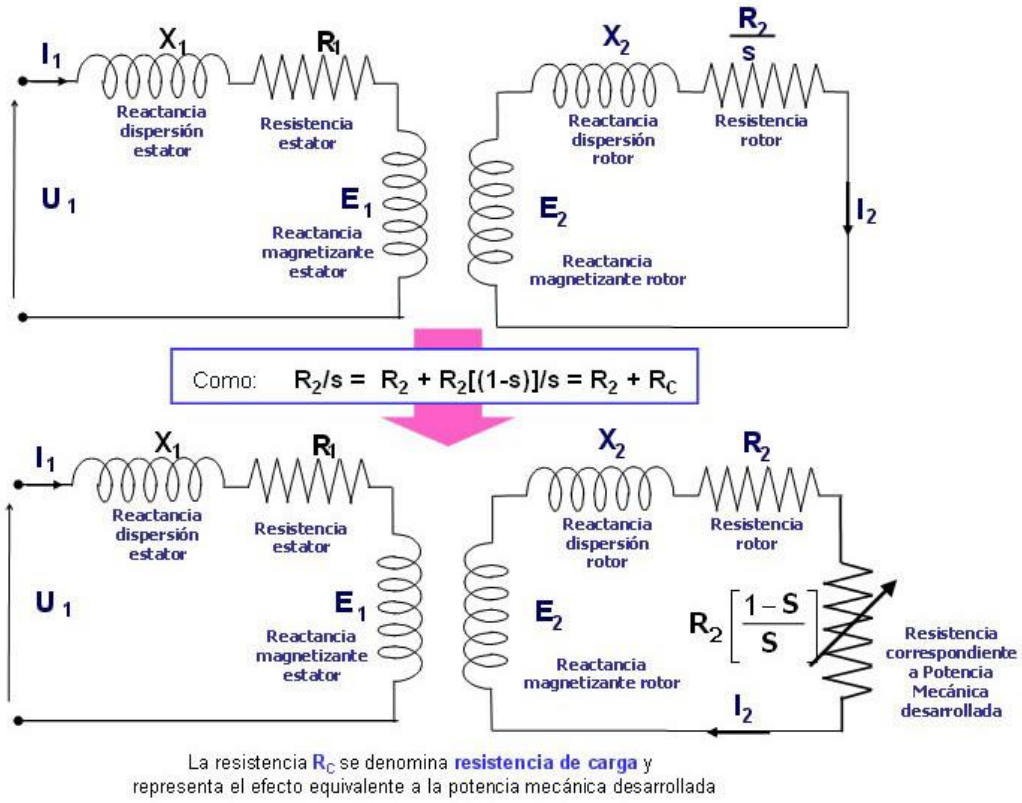


Figura 4.11: Modelo del motor de inducción.

Por último, si a este circuito lo referimos al primario se obtiene el modelo que aparece en la figura 4.12.

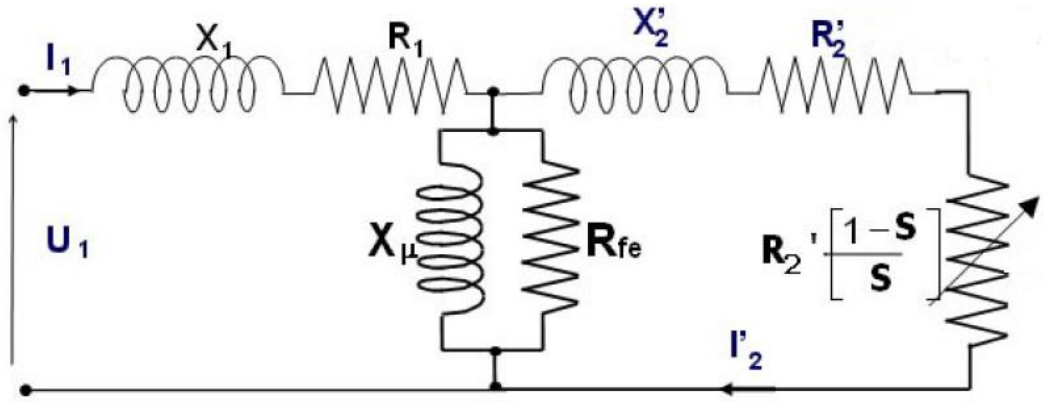


Figura 4.12: Circuito equivalente del motor asincrónico.

Donde R_{FE} representa las pérdidas en el núcleo y X_{μ} representa al flujo de magnetización confinado en el núcleo y común a los dos bobinados del transformador.

Especificaciones

Para terminar con todo lo relacionado al motor, veremos las especificaciones del mismo, que se encuentran en la chapa del motor:

- Conexión trifásica a 50 Hz.
- Potencia de 1.1KW = 1.5 HP.
- Tensión nominal de 220V en delta y 380V en estrella.
- Corriente nominal de 4.51A en delta y 2.61A en estrella.
- Velocidad nominal de 2840RPM.
- Cos φ = 0.84.

En la figura 4.13 se muestra como es la conexión de las bobinas del estator tanto en estrella como en delta.

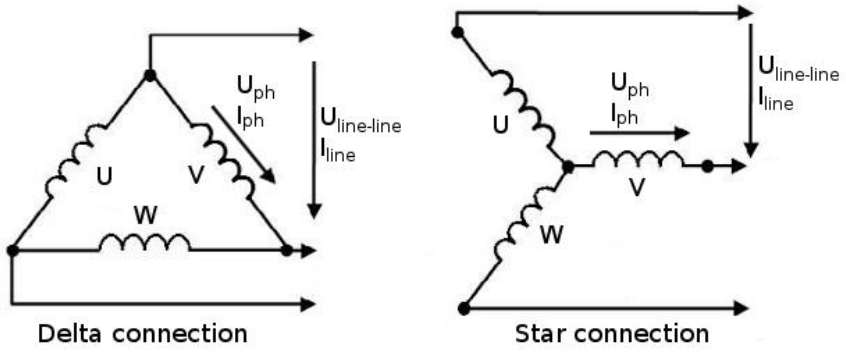


Figura 4.13: Conexión delta y estrella.

De acuerdo a la tensión de la red que se disponga se utilizara una o la otra. Se puede cambiar de una conexión a otra simplemente modificando los jumpers en los terminales del motor, como se muestra en la figura 4.14. La configuración a) corresponde a la conexión estrella y la b) a delta.

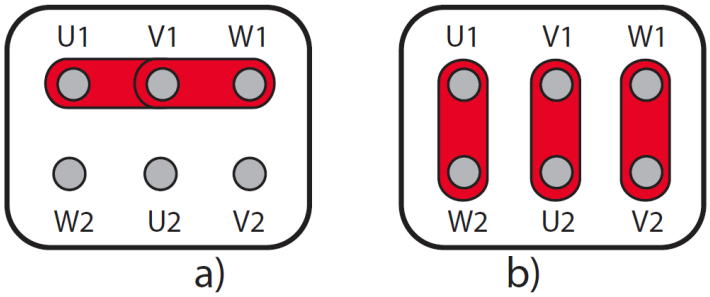


Figura 4.14: Terminales del motor.

La corriente nominal del motor, llamada corriente aparente, es dividida en dos componentes, la corriente activa y la corriente reactiva. La corriente activa se transforma en

potencia en el eje, mientras que la corriente reactiva es la potencia necesaria para obtener el campo magnético del motor. La corriente aparente no es la simple suma entre sus dos componentes, ya que se encuentran desplazadas en el tiempo una respecto de la otra. Se requiere de una suma cuadrática, como se expresa en la ecuación 4.10:

$$I_S = \sqrt{{I_W}^2 + {I_B}^2} \qquad (4.10)$$

Donde I_S es la corriente aparente, I_W es la corriente activa e I_B es la corriente reactiva. En la figura 4.15 se muestra un diagrama de estas corrientes.

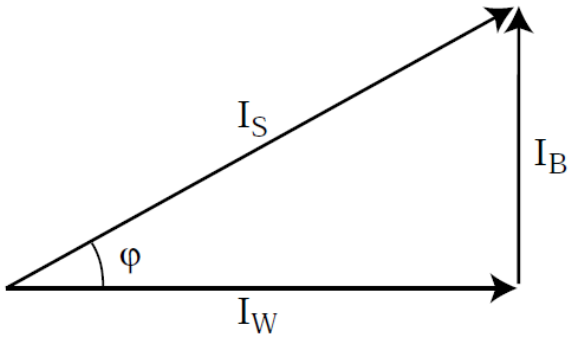


Figura 4.15: Relación entre las corrientes del motor.

Se ve como φ es el ángulo entre las corrientes y el coseno de este ángulo es como se expresa en la ecuación 4.11:

$$\cos\varphi = \frac{I_W}{I_B} = \frac{P}{S} \qquad (4.11)$$

El coseno del ángulo también es la relación entre la potencia aparente en la salida, S, y la potencia real P.

Con los datos en la chapa del motor también es posible determinar el torque nominal mediante la ecuación 4.12:

$$T = \frac{P*9550}{N} \cong 3.7Nm \qquad (4.12)$$

La eficiencia del motor es determinada como el cociente entre la potencia nominal y la potencia tomada de la red, como se expresa en la ecuación 4.13:

$$\eta = \frac{P}{\sqrt{3}*V*I*\cos\varphi} = \frac{1.1KW}{\sqrt{3}*220V*4.51*0.84} \cong 0.753 \qquad (4.13)$$

La velocidad de deslizamiento se puede calcular mediante la ecuación 4.2, recordando que la velocidad sincrónica ya fue calculada en la ecuación 4.1:

$$S = N_S - N = 3000RPM - 2840RPM = 160RPM$$

El valor relativo del deslizamiento resulta como se muestra en la ecuación 4.14:

$$s = \frac{S}{N_S} = \frac{160RPM}{3000RPM} = 5.33\% \qquad (4.14)$$

4.2 Selección del convertidor de frecuencia

Lo primero a tener en cuenta son las características de la red con la cual se quiere utilizar el variador de velocidad y el motor. En este caso se utiliza una sola fase de 220V y 50Hz. Para esta alimentación el motor debe ser conectado en la configuración delta y presenta una corriente nominal de 4.51A. Por ende, se eligió un convertidor de frecuencia que cumpla con los siguientes requisitos:

- Tensión de entrada monofásica de 220V y 50Hz.
- Debe proveer una corriente mayor a 4.51A.
- Potencia nominal mayor a 1.1KW = 1.5HP.

Otro de los factores determinantes es el método de control que utiliza el equipo. El clásico algoritmo V/f constante con compensación, también llamado escalar, típicamente ofrece:

- Regulación del orden del 2%/3% a lazo abierto.
- Velocidad de respuesta entre 2Hz y 5Hz.
- Rango de velocidad de 1:25, es decir que se puede operar a una velocidad mínima 25 veces más pequeña que la nominal.

En cambio, un método de control vectorial a lazo abierto tiene:

- Regulación del orden del 0.5%.
- Rango de velocidad de 1:100.
- Velocidad de respuesta entre 3Hz y 20Hz.

Debido a las claras ventajas que ofrece el control vectorial sobre el escalar, se eligió un convertidor que utilice control vectorial.

Por último, también se tuvo en cuenta el costo y la disponibilidad en la ciudad. Del análisis de todos estos aspectos, se determinó el uso del convertidor de frecuencia VLT Micro Drive FC 51, en su versión de 2HP de potencia de la compañía Danfoss. La imagen del mismo se puede observar en la figura 4.16. Este equipo es capaz de entregar una corriente ininterrumpida de 6.8A, superior a la nominal del motor en su conexión delta, y una corriente intermitente de 10.2A. Por otro lado, cuenta con las siguientes protecciones:

- Protección térmico-electrónica del motor contra sobrecarga.
- Control de la temperatura del disipador, que garantiza la desconexión del variador de velocidad en caso de sobretemperatura.
- Protección contra cortocircuito entre los terminales del motor.
- En caso de la falta de una fase del motor o la red, el convertidor de frecuencia se desconecta y genera una alarma.

- Control de la tensión en el circuito intermedio, que asegura la desconexión del equipo ante un nivel demasiado bajo o alto.

Figura 4.16: El convertidor de frecuencia VLT Micro Drive FC51.

Una imagen con un diagrama en bloques simplificado del convertidor con todas sus conexiones es mostrada en la figura 4.17. Aparte de los terminales para la tensión de red y para el motor, VLT Micro Drive FC51 cuenta con los siguientes terminales:

- Entrada analógica de corriente de 4mA-20mA para utilizarse como señal de realimentación o referencia.
- Entrada analógica programable como de corriente de 4mA-20mA o de tensión de 0V a 10V para usarse como señal de realimentación o referencia.
- Salida programable como analógica de corriente de 4mA-20mA, de corriente de 0mA a 20mA o digital. Las señales en esta salida son de información acerca del estado del convertidor.
- Cinco entradas digitales programables para cumplir distintas funciones de control, incluyendo una de pulsos.
- Comunicación serie RS-485 con protocolo Modbus.
- Dos relés de salida programable, utilizadas para generar señales de advertencias.
- Dos terminales destinados a la conexión de una resistencia de frenado externa.
- Dos terminales con el acceso a la tensión en el circuito intermedio.
- Un terminal de 10V y otro para su referencia, destinados para las entradas analógicas.
- Un terminal de 24V y otro para su referencia, destinados para las entradas digitales.

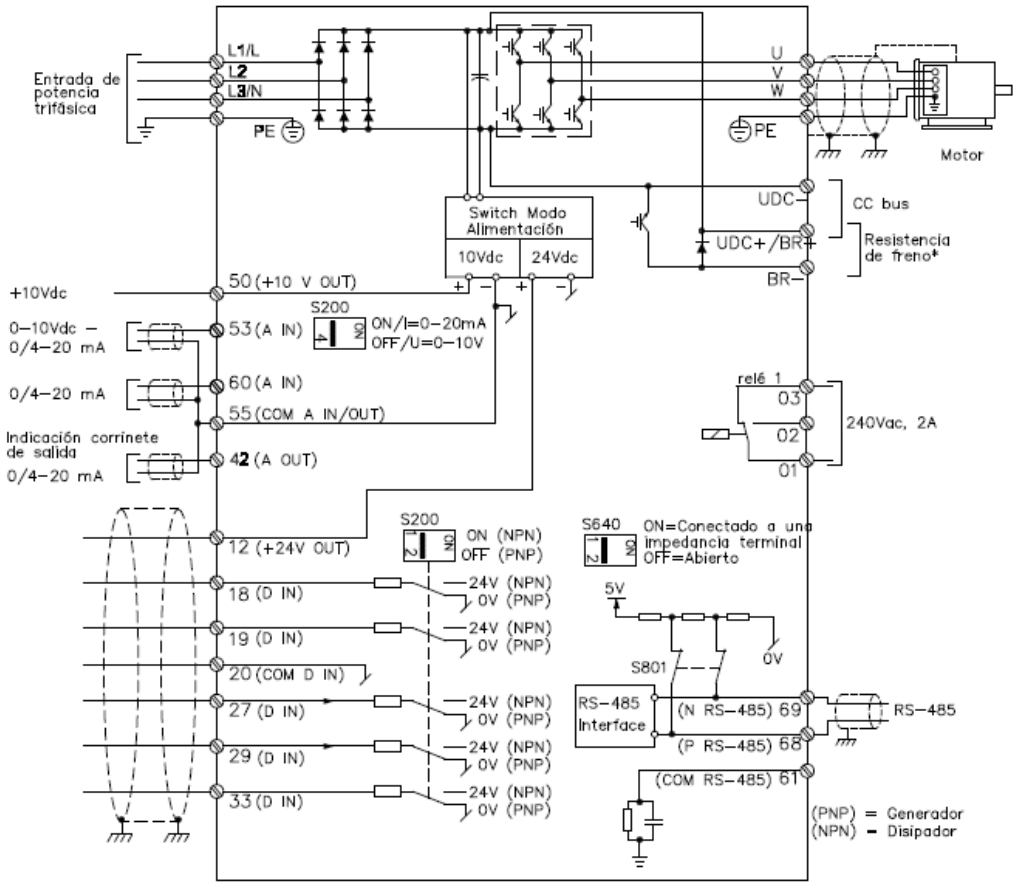


Figura 4.17: Conexiones en el convertidor de frecuencia.

En las siguientes secciones analizaremos el funcionamiento de cada una de las partes que componen un convertidor de frecuencia, haciendo énfasis en la configuración utilizada en el variador de velocidad VLT Micro Drive FC51, y los métodos utilizados por este equipo para el control de la velocidad del motor. Por último, se detallará el control remoto del convertidor de frecuencia desde el microcontrolador.

4.3 Componentes de un variador de velocidad

Para controlar la velocidad del motor se utiliza un convertidor de frecuencia. Este equipo toma la tensión de red y le proporciona una señal trifásica con tensión y frecuencia variables al motor. En esta sección analizaremos las partes que componen a un convertidor de frecuencia. El diagrama en bloques básico del convertidor es mostrado en la figura 4.18.

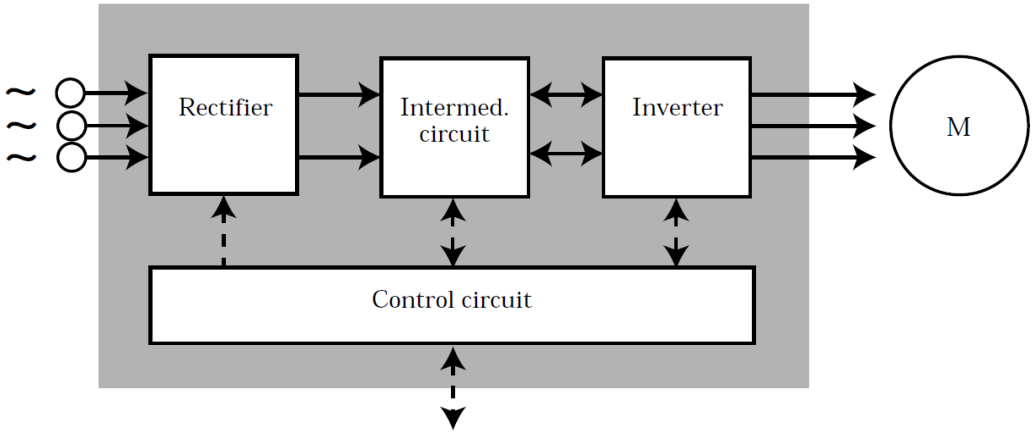


Figura 4.18: Diagrama en bloques del convertidor de frecuencias.

En esencia, VLT Micro Drive FC51 está compuesto por un rectificador monofásico, un capacitor de almacenamiento de energía como circuito intermedio, un inversor y circuito de control.

El inversor permite generar una terna trifásica de tensión, con amplitud y frecuencia variables a partir de una tensión de continua. Está compuesto por semiconductores controlados, IGBTs (Insulated Gate Bipolar Transistor) en este caso, y tiene una estructura y formas de onda como las que se observan en la figura 4.19.

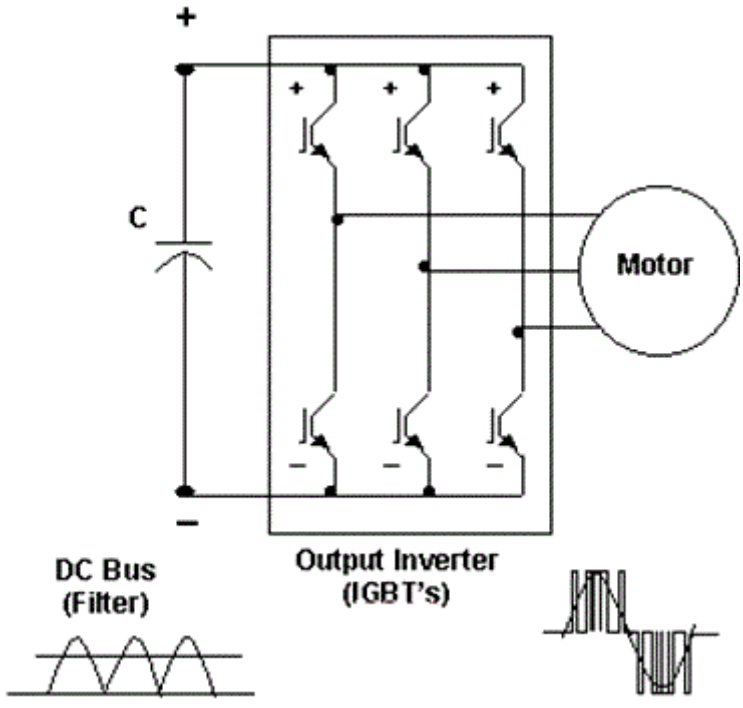


Figura 4.19: Inversor con sus formas de onda.

Por último, el circuito de control cumple cuatro funciones fundamentales:

- Controlar a los semiconductores del convertidor de frecuencia.
- Intercambio de datos entre el convertidor de frecuencia y los periféricos.
- Control y reporte de fallas.
- Implementar funciones de protección para el motor y el variador de velocidad.

4.4 Métodos de modulación

En un convertidor PWM se genera una corriente senoidal en el motor enviando una determinada secuencia de pulsos a las llaves. La velocidad del motor se controla mediante la frecuencia de repetición de la secuencia de pulsos, mientras que la amplitud la impone la relación entre el tiempo de encendido y de apagado de las llaves. Para generar la secuencia de pulsos necesaria en los transistores del inversor existen fundamentalmente dos tipos de modulación, una es la modulación senoidal y la otra es la modulación de vectores espaciales.

En la modulación senoidal se compara una señal de referencia, de forma de onda senoidal de frecuencia igual a la velocidad en el motor, con una señal triangular de frecuencia mucho mayor a la del motor. En la figura 4.20 se muestra un ejemplo de esta modulación con dos fases.

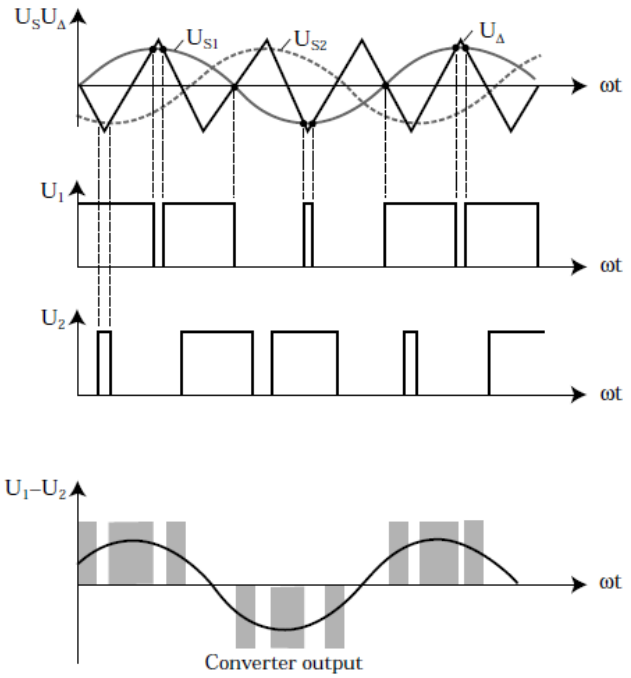


Figura 4.20: Modulación senoidal.

La forma de onda resultante de esta modulación es muy dependiente de la frecuencia de la onda triangular, también llamada frecuencia portadora. Se muestran los resultados para dos frecuencias diferentes en la figura 4.21.

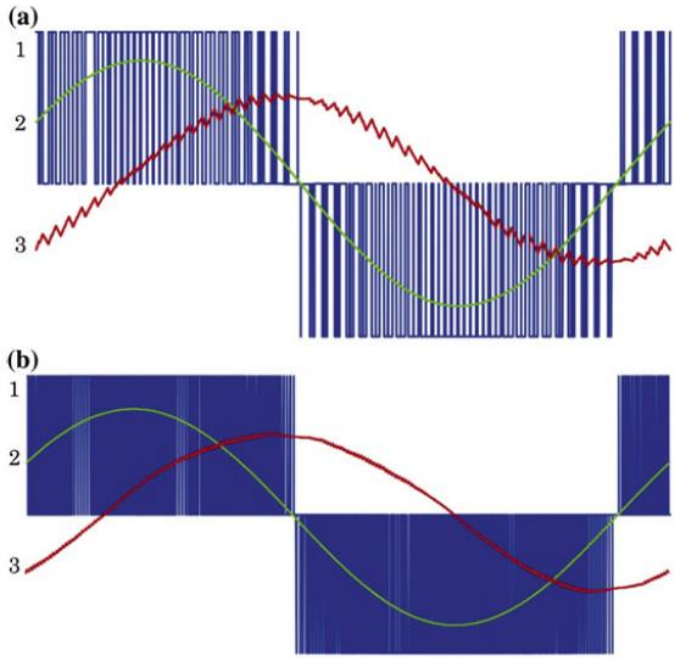


Figura 4.21: Formas de onda en la modulación senoidal para distintas frecuencias.

En el caso b) la frecuencia de portadora es mayor que en a), esto tiene como consecuencia una menor distorsión en b). Además este método tiene como desventajas, que no hace un buen uso de la tensión en el circuito intermedio y presenta una distorsión por armónicos elevada (THD).

Por otro lado, aparece modulación de vector espacial (cuya sigla en inglés es SVM). Que hace uso de las distintas combinaciones posibles para las dos llaves de cada una de las tres ramas del inversor. Estas combinaciones son presentadas en la figura 4.22.

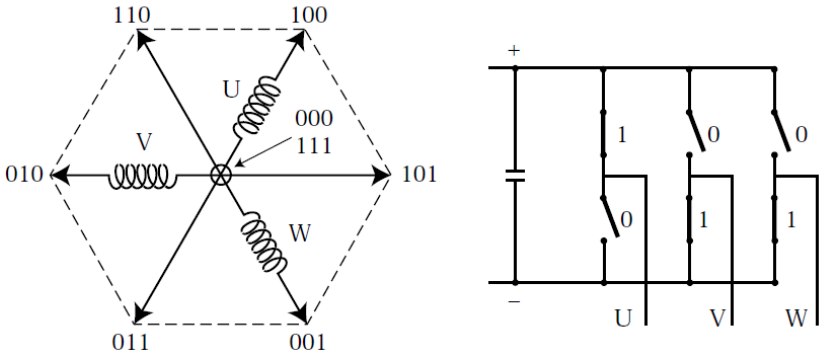


Figura 4.22: Combinaciones posibles para las llaves del inversor.

En cada rama hay dos llaves complementarias, es decir que cuando una está abierta la otra debe estar cerrada y viceversa. Esto deja ocho niveles discretos de tensión que se pueden aplicar a las bobinas del estator. De las ocho combinaciones, en dos se aplica la misma tensión en cada una de las fases, la tensión positiva del circuito intermedio para la todas las llaves en "1" y la tensión negativa del circuito intermedio cuando todas las llaves están en "0". De esta forma queda un hexágono con los seis niveles restantes, 000 y 111 aparecen en el centro del mismo.

Para determinar la secuencia adecuada para lograr la tensión y frecuencia deseada en el inversor, se hace uso de la trasformación de Clarke para un sistema trifásico, también llamada transformación αβγ, mostrada en la ecuación 4.15:

$$\begin{bmatrix} i_\alpha(t) \\ i_\beta(t) \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix}\begin{bmatrix} i_a(t) \\ i_b(t) \\ i_c(t) \end{bmatrix} \qquad (4.15)$$

Donde las corrientes a, b y c son las de cada una de las fases del motor. Esta es una versión simplificada, ya que se hace uso del balance del sistema expresada en la ecuación 4.16:

$$i_a(t) + i_b(t) + i_c(t) = 0 \qquad (4.16)$$

La transformación de Clarke puede pensarse como la proyección de tres cantidades trifásicas, corrientes o tensiones, sobre dos ejes complejos y estacionarios α y β. Este concepto se grafica en la figura 4.23.

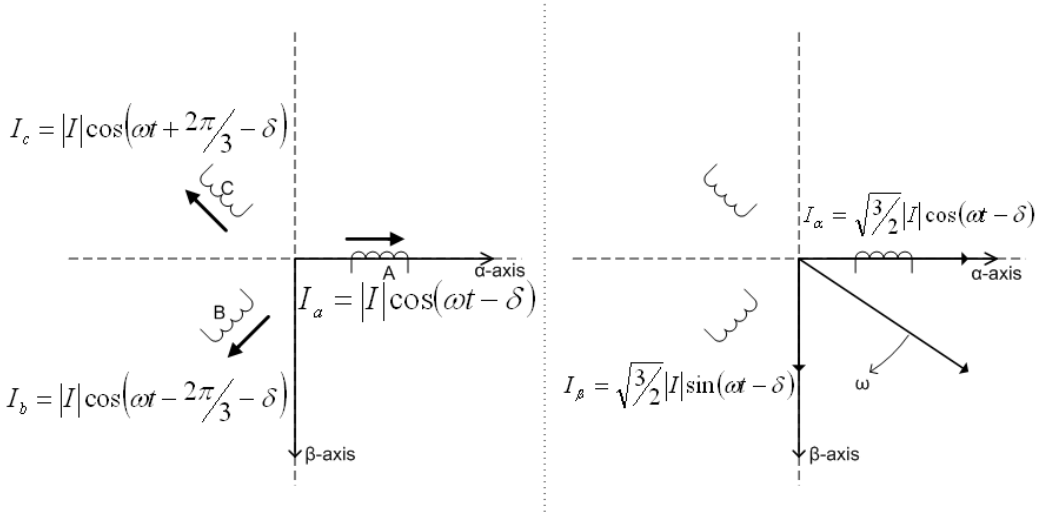


Figura 4.23: Transformación de Clarke.

El sistema de coordenadas αβ es estacionario, por ende, para tener una representación completa del motor se introduce un marco de referencia que se mueva a la velocidad sincrónica y que será muy importante en el algoritmo de control. Con este fin se recurre a la transformación de Park, expresada en la ecuación 4.17:

$$\begin{bmatrix} i_d(t) \\ i_q(t) \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i_\alpha(t) \\ i_\beta(t) \end{bmatrix} \qquad (4.17)$$

Esta transformación representa el pasaje de un sistema de referencia a otro, como se ilustra en la figura 4.24, donde entre ambos existe un desfasaje.

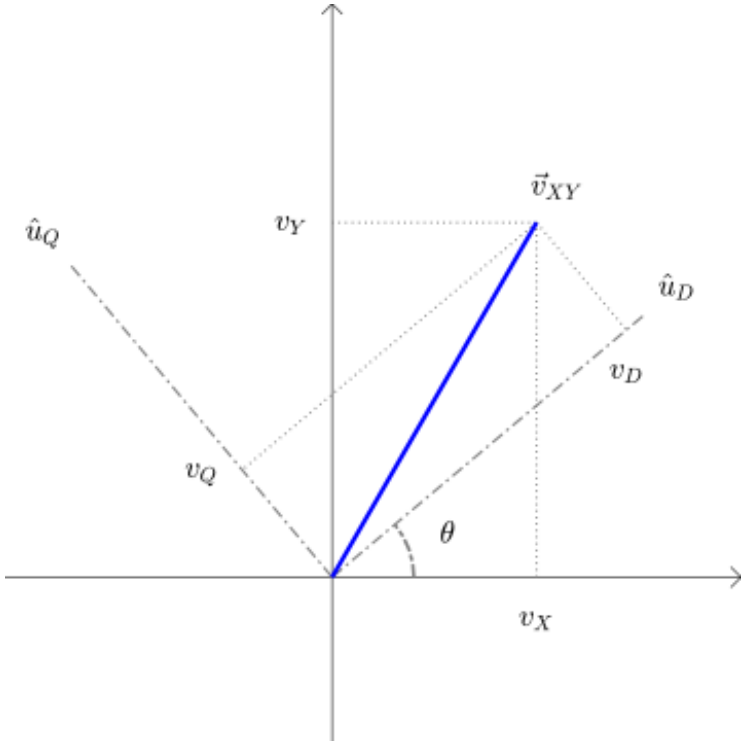


Figura 4.24: Transformación de Park.

El desfasaje entre un sistema y otro está dado por la ecuación 4.18:

$$\theta(t) = \int_0^t \omega(t)dt + \theta_0 \qquad (4.18)$$

Donde ω es la velocidad angular sincrónica. En definitiva, en el sistema de coordenadas αβ se representan los vectores complejos de cualquier cantidad trifásica, mientras que en el sistema de coordenadas dq se representan las mismas cantidades pero desde un sistema que gira sincrónicamente a su misma velocidad angular. En la figura 4.25 se muestra una representación de este concepto para la corriente en el estator.

Donde ω es la velocidad en el eje del motor, ω_S es la velocidad sincrónica, i_S es la corriente en el estator y ψ_r es el flujo en el rotor.

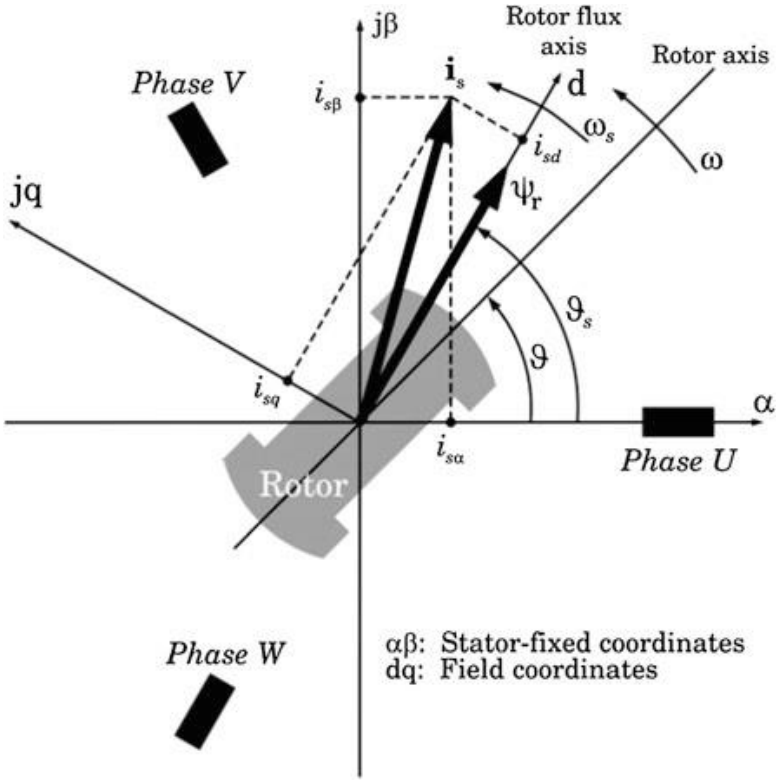


Figura 4.25: Corriente en el estator en los sistemas de ejes αβ y dq.

Se representan los ocho vectores posibles en los ejes αβ en la figura 4.26. Entre un vector y otro se definen secciones, siendo seis en total. Se puede obtener cualquier vector, al que llamaremos vector de referencia, simplemente con sus dos vectores adyacentes. En la figura 4.27 se muestra un vector arbitrario u_S, que representa la tensión en el estator, representado en los dos sistemas de ejes. Se puede ver que u_S es la suma de dos vectores en la dirección de u_1 y u_2, u_r y u_l. Estos dos vectores se obtienen promediando el tiempo en que u_1 y u_2 se aplican al motor, en un tiempo disponible para la realización del vector de referencia, T_P. Esto se expresa en las ecuaciones 4.19 y 4.20:

$$|u_r| = |u_s|\frac{T_R}{T_P} \qquad (4.19)$$

$$|u_l| = |u_s|\frac{T_L}{T_P} \qquad (4.20)$$

En caso de que la suma de los tiempos T_R y T_L sea menor que T_P, se debe completar con uno de los vectores u_0 o u_7. Este tiempo está dado por la ecuación 4.21, en caso de completar con u_0:

$$u_S = u_1\frac{T_R}{T_P} + u_2\frac{T_L}{T_P} + u_0\frac{T_P-(T_R+T_L)}{T_P} \qquad (4.21)$$

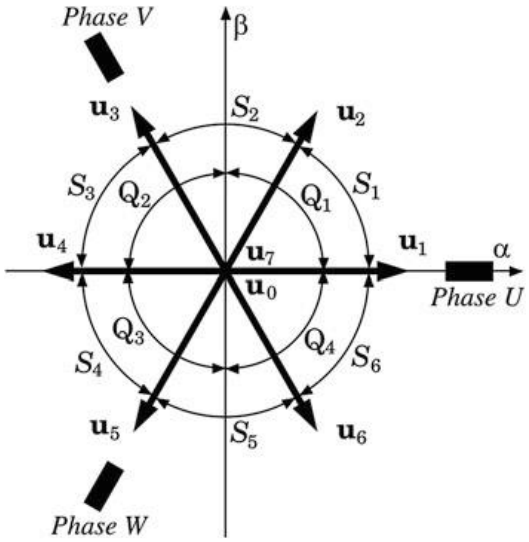


Figura 4.26: Representación de los ocho vectores posibles en el sistema αβ.

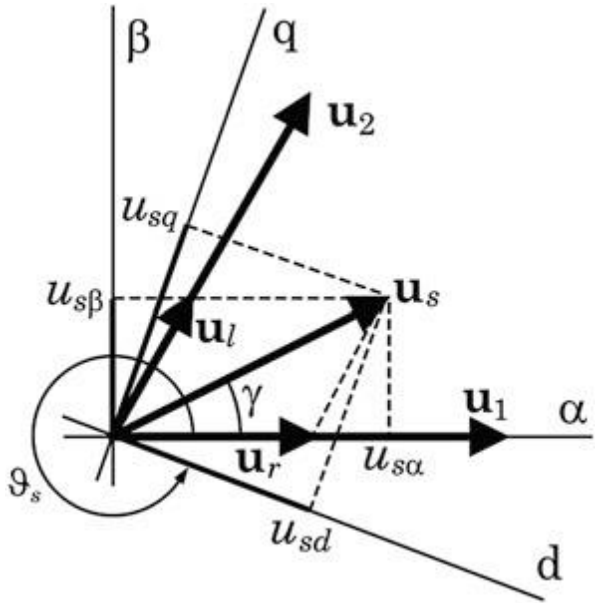


Figura 4.27: Tensión en el estator en los sistemas αβ y dq.

A su vez, para compensar la inclusión de uno de estos dos vectores, se repite la secuencia, pero se completa con el otro vector. Para reducir las pérdidas por conmutación en los transistores se utiliza una secuencia tal que las transiciones sean mínimas. En la figura 4.28 se muestra la secuencia resultante para el ejemplo planteado.

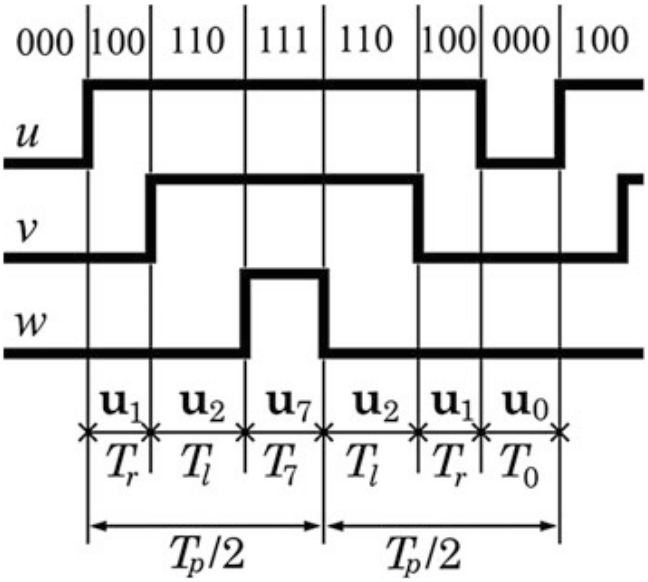


Figura 4.28: Secuencia resultante para un vector de referencia en S_1.

Para cualquier vector de referencia en la sección 1 la secuencia es la misma, pero con una distribución de tiempos diferentes. Lo mismo sucede con el resto de las secciones.

Este es el procedimiento básico de SVM, existen distintas variantes que ofrecen algunas ventajas. En el caso del variador de velocidad utilizado en el proyecto, implementa dos tipos de modulación llamadas SFAVM (Stator Flow-oriented Asynchronous Vector Modulation) y 60° AVM (Asynchronous Vector Modulation). De acuerdo a una estimación de la temperatura en su disipador, utiliza un método de modulación u otro.

En una comparación entre la modulación senoidal y vectorial, la segunda ofrece mejor dinámica, performance, menor distorsión armónica y hace un mejor uso de la tensión disponible en el circuito intermedio.

4.5 Algoritmos de control

V/f

Este es el clásico algoritmo de control en convertidores de frecuencia. Actualmente ha sido reemplazado por otros métodos de mejor performance, aunque es muy común que los convertidores puedan operar con más de un método de control, seleccionando un determinado algoritmo según las condiciones de operación.

Como se explicó anteriormente, para cambiar la velocidad en el eje del motor es necesario variar la frecuencia de la tensión que se aplica a las bobinas del estator, resultando en un cambio de la velocidad del campo magnético rotante. Pero, para mantener el torque del motor es necesario

modificar también la tensión en el mismo. Si en la ecuación 4.3 reemplazamos la potencia se obtiene la ecuación 4.22:

$$T = \frac{9550*P}{N} = \frac{9550*\sqrt{3}*U*I*\eta*\cos\varphi}{f*60/p} = K*\frac{U}{f} \qquad (4.22)$$

Esta expresión deja en claro que el torque en el motor es directamente proporcional al cociente entre la tensión y la frecuencia en el motor. De este modo, ante un cambio en la frecuencia del motor, también se modifica la tensión de forma tal que el cociente se mantenga constante, el torque se mantendrá constante en todo el rango de operación. También la magnetización se mantiene en su nivel óptimo. Un ejemplo de un motor con tensión nominal de 400V y frecuencia nominal de 50Hz es mostrado en la figura 4.29, donde se muestra la curva del torque en función de la velocidad aplicando el método de V/f constante.

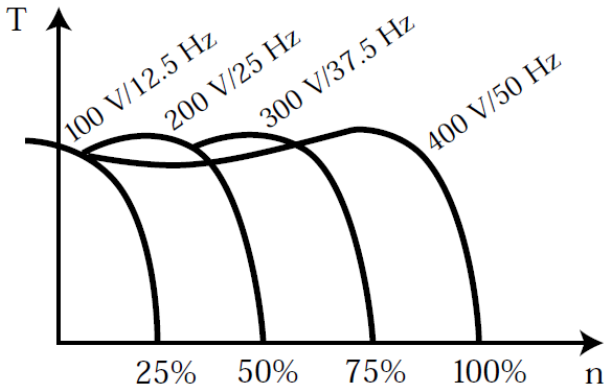


Figura 4.29: Curva de T vs N aplicando V/f.

No obstante, existen ciertas condiciones donde la magnetización no es la ideal. Planteando la ecuación de malla para el estator en el circuito equivalente del motor de la figura 4.12, se obtiene la ecuación 4.23:

$$V_1 = V_{R1} + V_{X1} + V_{MAG} \qquad (4.23)$$

Donde V_1 es la tensión en el motor, V_{R1} es la caída de tensión en la resistencia del estator, V_{X1} es la caída en la bobina del estator y V_{MAG} es la tensión de magnetización. Si se reduce la velocidad del motor y se aplica este algoritmo, la tensión también se reduce. Pero como la corriente se mantiene, V_{R1} no se modifica, resultando en una reducción de la tensión de magnetización. En consecuencia, el motor no se encuentra correctamente magnetizado y no es capaz generar el torque nominal. Para combatir este efecto se aplica una tensión de compensación adicional para bajas velocidades.

El VLT Micro Drive FC51 utiliza este método con motores conectados en paralelo o aplicaciones especiales, aunque no provee compensaciones de carga y deslizamiento con este tipo de control.

VVC+

El algoritmo de control que implementa este convertidor de frecuencia es el Voltage Vector Control Plus (VVC+), llamado así por el fabricante, y está basado en Field Oriented Control (FOC). FOC plantea las transformaciones vistas en SVM, las de Clarke y Park, para obtener el sistema de coordenadas dq que gira con la velocidad sincrónica. El flujo en el rotor se alinea con el eje d. De este modo, en la representación de la corriente en el estator se obtienen dos componentes perpendiculares, una en el eje d y la otra en el eje q. La componente directa, eje d, es la corriente que impone el flujo, mientras que la componente en cuadratura, eje q, es la corriente que produce el torque. La representación de la corriente en el estator fue mostrada en la figura 4.25, donde se ve que la velocidad a la gira el campo magnético en el rotor, velocidad sincrónica, es diferente a la que gira el rotor debido al deslizamiento. Al ser perpendiculares estas dos corrientes pueden manejarse de forma independiente a la otra. Es decir, que se puede controlar de forma independiente el flujo en el rotor y el torque, de forma análoga al control de motores de continua.

VVC+ utiliza los datos nominales del motor para obtener el circuito equivalente de motor. A partir del circuito equivalente, mostrado en la figura 4.30, se calcula la tensión sin carga en el estator. Teniendo en cuenta que no circula corriente en el rotor, se obtiene la expresión de la ecuación 4.24:

$$U_L = (R_S + s * L_S) * i_S \qquad (4.24)$$

Siendo R_S la resistencia en el estator, i_S la corriente en el estator y L_S es la bobina en el estator, que se compone como se muestra en la ecuación 4.25:

$$L_S = L_{S\sigma} + L_h \qquad (4.25)$$

Donde el primer término es la inductancia de pérdida en el estator y el segundo es la inductancia magnetizante.

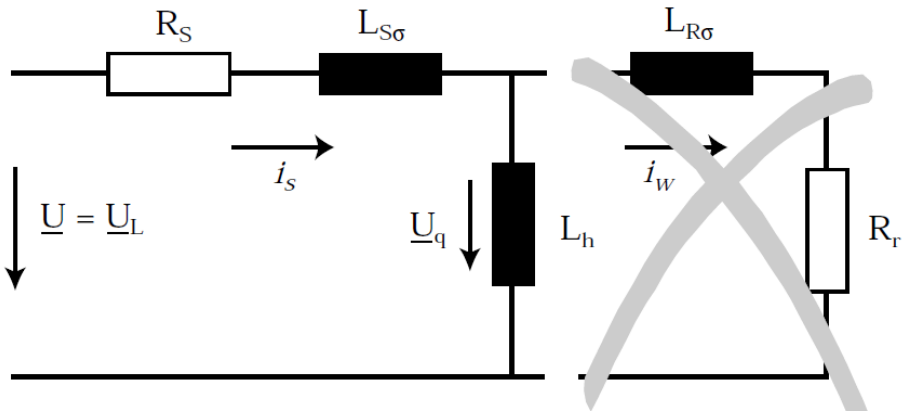


Figura 4.30: Circuito equivalente del motor sin carga.

En cambio, con carga en el rotor aparece la corriente activa y resulta el circuito de la figura 4.31.

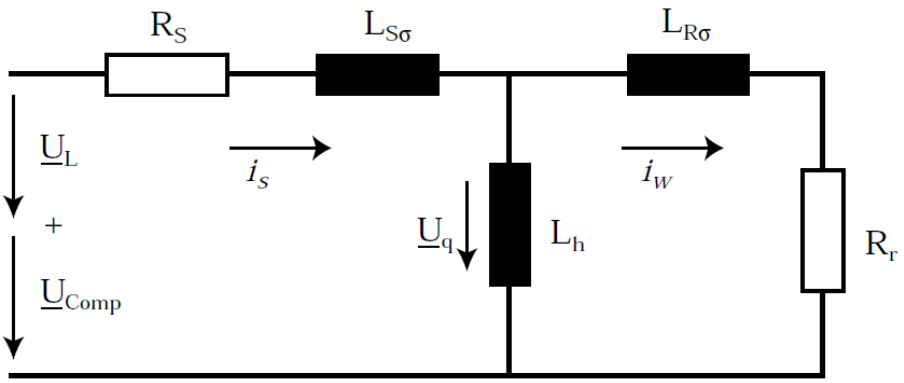


Figura 4.31: Circuito equivalente del motor con carga.

Donde la bobina que aparece en el circuito del rotor presenta la inductancia de pérdida en el rotor y R_r es la suma de la resistencia del bobinado en el rotor y la carga. Para producir esta corriente, se pone a disposición del motor una tensión de compensación U_{COMP}. Esta tensión se determina en función la corriente sin carga, la corriente activa y el rango de velocidad.

Teniendo en cuenta estas consideraciones, se presenta el diagrama en bloques del algoritmo de control VCC+ en la figura 4.32. En el modelo del motor se calculan las corrientes y ángulos para el compensador por carga y el generador del vector de tensión, conociendo la frecuencia. Calculando los valores sin carga, es posible estimar de forma mucho más precisa el torque de carga en el eje del motor.

El generador del vector de tensión calcula U_L y el ángulo del vector de tensión sin carga, Θ_L, utilizando el circuito de la figura 4.30. A estos valores se le suman los términos de compensación producidos por el compensador de carga. Este bloque toma como parámetros de entrada la frecuencia, las componentes sin carga y las corrientes reales en el motor. Las corrientes trifásicas en el motor son llevadas a otro sistema de referencia mediante las transformaciones de Clarke y Park, resultando la corriente reactiva, la que determina el flujo, y la corriente activa, la que impone el torque de carga.

El ángulo sin carga mejora el control para la aceleración a bajas velocidades. De esta forma, la corriente que produce el torque solo corresponderá a la carga en el motor. Sin este ángulo el vector de corriente tiende a aumentar y sobremagnetizar el motor sin producir torque.

Sumando U_{COMP}, se mantiene el nivel del campo magnético en su valor óptimo. Con $\Delta\Theta$ se corrige la desviación esperada del ángulo en el eje del motor en función de la carga. Por último, el vector de tensión de salida se representa en forma polar e ingresa al bloque ASIC (Circuito Integrado para Aplicaciones Específicas). En este bloque se calcula la secuencia de pulsos en el circuito inversor mediante SVM.

También se incluye compensación para el deslizamiento, con las corrientes activa y reactiva como entradas. Este bloque le permite al eje del motor girar a la velocidad sincrónica.

Por último, en el diagrama se observa que el variador sensa la tensión en el circuito intermedio y estima la temperatura para detectar fallas.

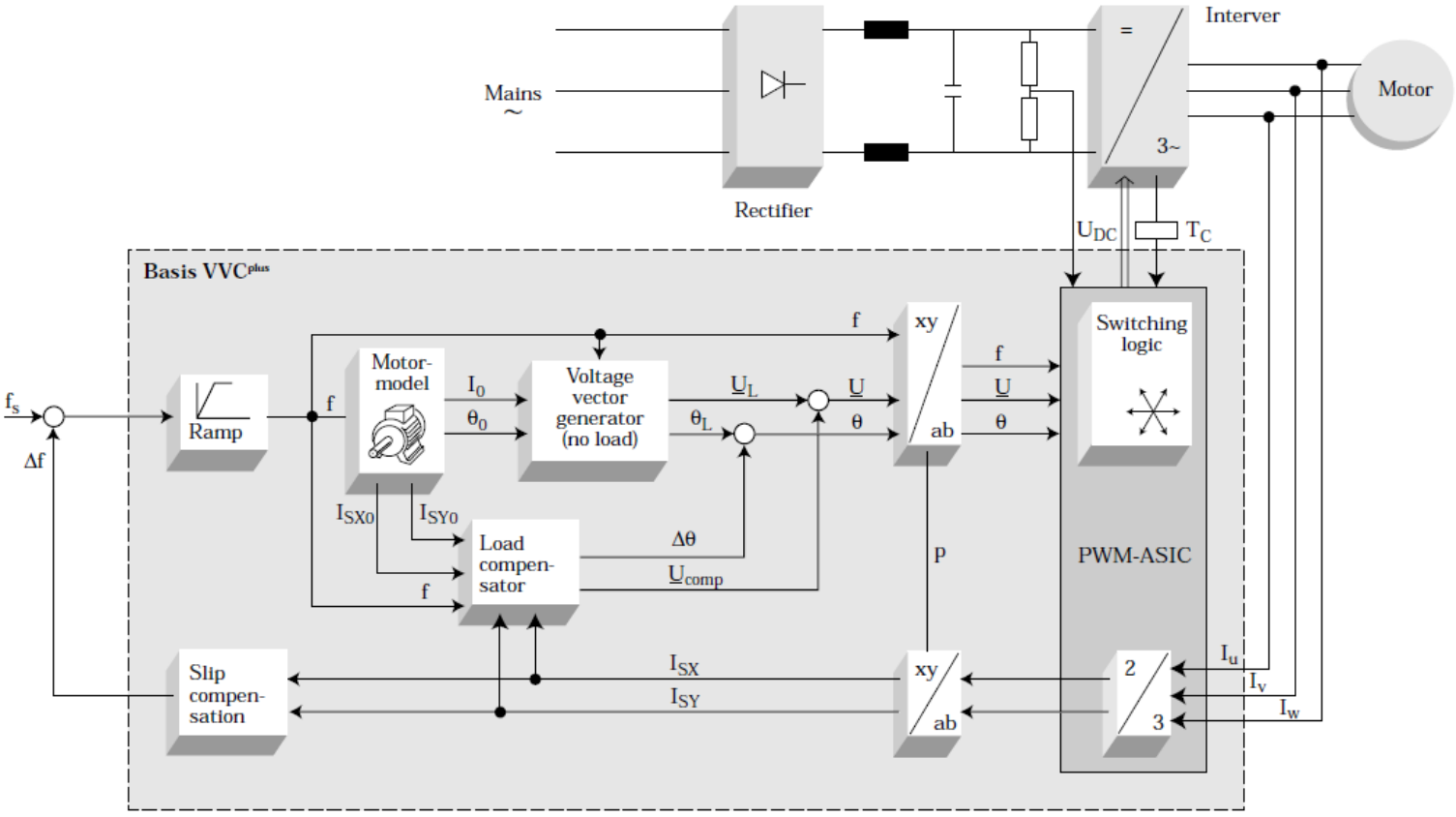


Figura 4.32: Diagrama en bloques del método de control VVC+.

4.6 Control y programación del VLT Micro Drive FC51

El primer paso en la programación consiste en realizar el Ajuste Automático del Motor (ATM), donde se le ingresan al convertidor de frecuencia los datos nominales del motor para que el equipo pueda obtener el circuito equivalente del motor. Este paso debe realizarse con el motor detenido.

El VLT Micro Drive FC51 posee una gran cantidad de parámetros a configurar, de los cuales repasaremos los más importantes. Empezamos con el grupo de parámetros uno, Carga/Motor, donde se seleccionó la siguiente configuración:

- Control de velocidad a lazo abierto.
- Algoritmo de control VCC+.
- Par constante.
- Se ingresaron los datos nominales del motor: potencia, tensión, corriente, frecuencia y velocidad.
- Se permite ingresar los valores de los componentes del circuito equivalente de forma manual, sin embargo estos valores se programaron con el ATM.

En el grupo de parámetros tres Referencia/Rampas se ajustó:

- Referencia mínima.
- Referencia máxima.
- Se deshabilitaron todas las fuentes de referencia, a excepción de la entrada analógica 53.
- Las rampas de aceleración y desaceleración se eligieron de tipo lineal y con valores por defectos, ya que la aceleración y desaceleración no son puntos que afecten el ensayo.

Por otro lado, en el grupo cuatro Limites/Advertencias se configuró:

- Dirección en ambos sentidos.
- Limites superior e inferior para la velocidad, adoptando la velocidad sincrónica nominal de 3000RPM.

En el grupo cinco E/S Digitales se deshabilitaron todas las entradas digitales, menos la 27 y 29. Una se utilizó para determinar el sentido de giro y la otra para el arranque y la detención del motor.

Por último, en el grupo seis E/S Analógica se configuro únicamente la entrada 53, dejando sin función al resto. A esta entrada se la utilizo como referencia, por lo que se ajustó la curva de frecuencia en función de la tensión en esta entrada.

Para las entradas digitales se utilizaron dos salidas del microcontrolador. Sin embargo, como el variador tiene una lógica de 24V y el microcontrolador de 3.3V, se añadió un circuito para intermediar. El circuito es mostrado en la figura 4.33 y consiste de un optoacoplador, el que es manejado por el microcontrolador. El colector del fototransistor es conectado a los 24V que ofrece el variador de velocidad mediante una resistencia pull-up.

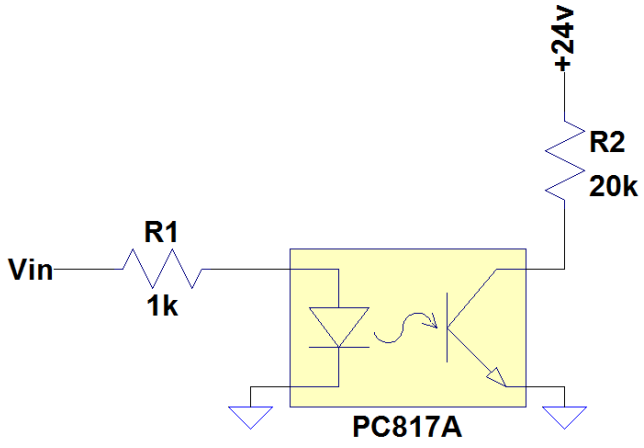


Figura 4.33: Circuito para manejar las entradas digitales del variador.

Cuando la salida del microcontrolador este en bajo, el fototransistor está en corte y no circula corriente por R_2, resultando una tensión de 24V en la salida de este circuito. En cambio, con la salida del µC en 3.3V, el fototransistor está en saturación y circula corriente por R_2 y la tensión en

la salida cae a 0V. La tensión de entrada, de verde, y salida, de azul, de este circuito se presentan en la figura 4.34. Resultado en una inversión de la lógica.

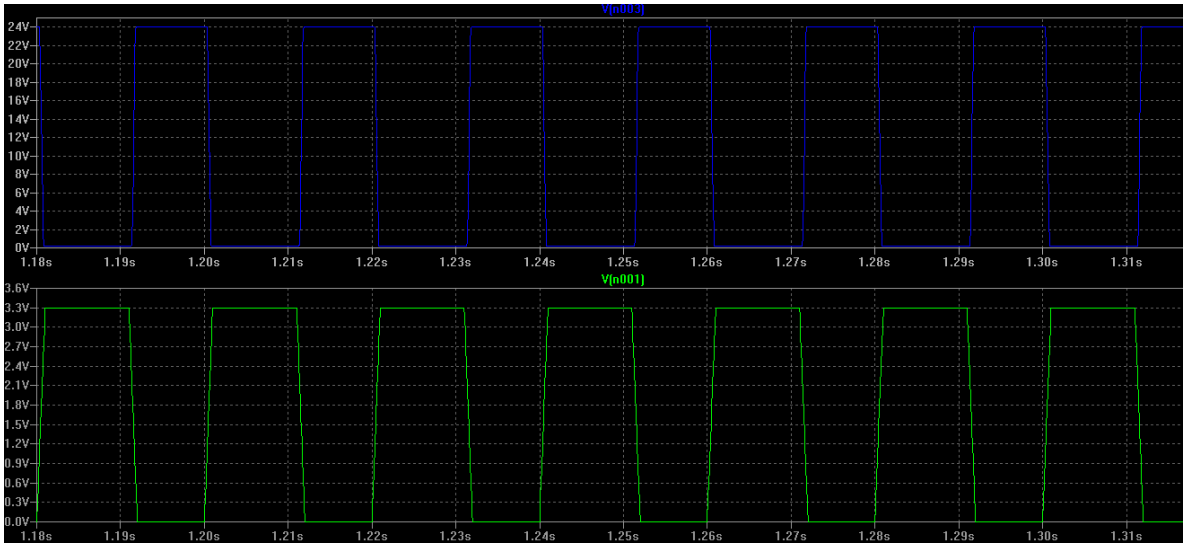


Figura 4.34: Tensiones de entrada y salida del circuito.

En cuanto a la tensión de la entrada analógica, la misma proviene del DAC del microcontrolador. La tensión de referencia para el DAC y ADC del microcontrolador proviene de una referencia de precisión shunt.

5

Sensor inductivo y Optoacoplador ranurado

5.1 Sensor de proximidad inductivo

Uno de los requerimientos para la automatización de esta máquina consistía en definir la duración del ensayo en función del recorrido tangencial del disco, razón por la cual se instaló en la máquina un sensor de proximidad inductivo. Este dispositivo detecta la presencia de elementos metálicos sin la necesidad de contacto. En este caso el elemento metálico a detectar son los dientes del eje del motor, de esta forma se obtiene información sobre la cantidad de vueltas dadas por el eje y en consecuencia del recorrido tangencial del disco.

Analizaremos el funcionamiento de este dispositivo, para esto haremos referencia a la figura 5.1, donde se muestra el diagrama interno del sensor inductivo. Ubicada justo detrás de la superficie activa, el lado derecho del diagrama, una bobina con núcleo de ferrite forma parte del circuito sintonizado de un oscilador. Cuando el oscilador entra en funcionamiento se produce un campo magnético de alterna, llamado campo de sensado, producido por la bobina. Este campo magnético irradia a través de la superficie activa del sensor, que es no-metálica. El circuito se mantiene sintonizado mientras que el campo de sensado detecte materiales no-metálicos.

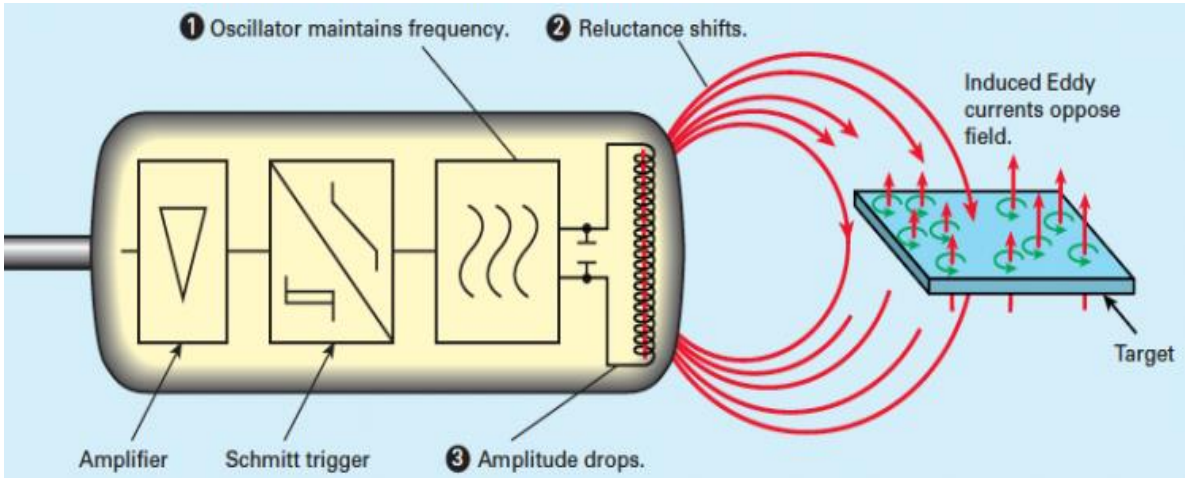


Figura 5.1: Diagrama de un sensor de proximidad inductivo.

En cambio, cuando se acerca un material metálico a la superficie activa, el campo magnético de alterna genera corrientes de Eddy que empiezan a circular dentro del objetivo. Estas corrientes se oponen al campo magnético que atraviesa al objetivo y son vistas como pérdida de potencia por el oscilador. A medida que se acerca el objetivo estas corrientes aumentan y se carga al oscilador, este efecto de carga tiene como consecuencia la caída en la amplitud de salida del oscilador. La caída en la amplitud de las oscilaciones es detectada por el circuito de trigger, que ahora cambia su salida de alto a bajo. Luego la etapa de salida, que invierte la salida del circuito de trigger, pasa de bajo a alto. En la figura 5.2 se pueden observar todas las formas de onda en el sensor de proximidad inductivo en función de la distancia con el objetivo.

Como se muestra en la figura 5.3, se coloca el sensor cerca del eje del motor para detectar los dientes en el eje y medir la cantidad de giros. De esta forma, si el motor gira a una determina velocidad, en la salida del sensor inductivo se obtendrá una onda cuadrada cuya frecuencia está dada por la ecuación 5.1:

$$f_{SENSOR} = N_{DIENTES} * f_{MOTOR} \qquad (5.1)$$

Donde $N_{DIENTES}$ es el número de dientes que presenta el eje del motor y f_{MOTOR} es la velocidad de giro. La salida del sensor podría ser utilizada como señal de realimentación en el variador de velocidad para trabajar a lazo cerrado, sin embargo, se optó por operar a lazo abierto.

Contando los pulsos en la señal de salida del sensor es posible determinar la cantidad de vueltas que dio el eje. La ecuación 5.2 muestra la relación entre el número de pulsos en la salida del sensor y la cantidad de vueltas:

$$N_{VUELTAS} = {}^{N_{PULSOS}}/_{N_{DIENTES}} \qquad (5.2)$$

Sin embargo, las ecuaciones 5.1 y 5.2 son para el eje del motor y no para la rueda. Entre el eje del motor y la rueda existe una reducción de 7.5 veces. Teniendo esto en cuenta, en la ecuación 5.3 se expresa la frecuencia y en la ecuación 5.4 el número de vueltas para la rueda:

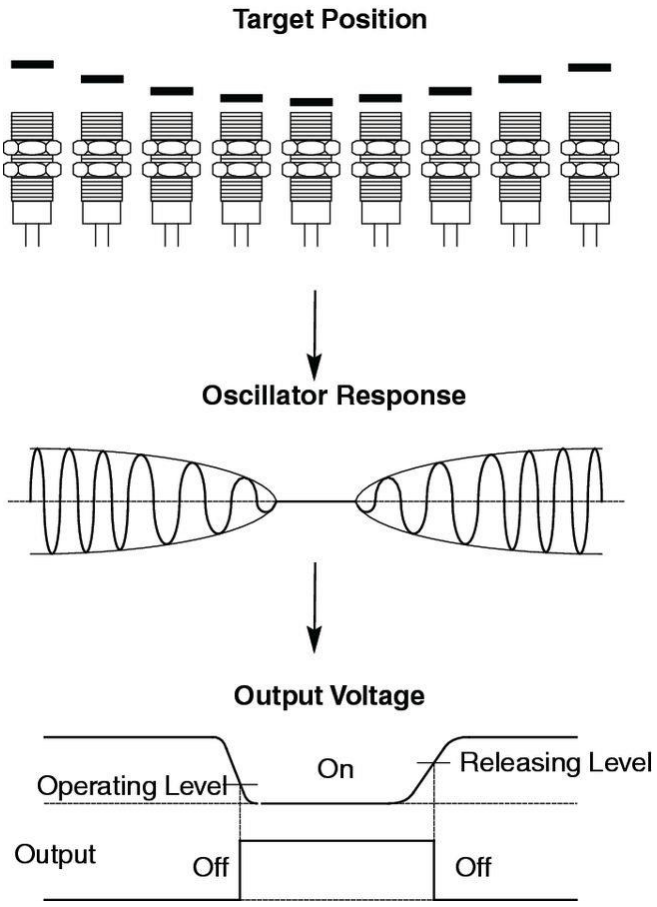


Figura 5.2: Formas de onda en el sensor inductivo.

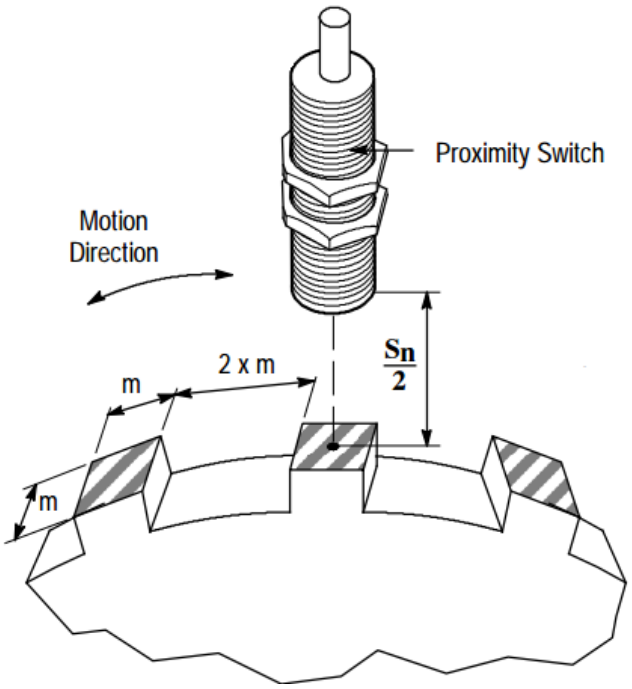


Figura 5.3: Medición del motor.

$$f_{RUEDA} = {f_{MOTOR}}/{7.5} = \frac{f_{SENSOR}}{7.5 * N_{DIENTES}} \quad (5.3)$$

$$N_{VUELTAS} = \frac{N_{PULSOS}}{7.5 * N_{DIENTES}} \quad (5.4)$$

Con el número de vueltas y el diámetro de la rueda de obtiene el recorrido tangencial del disco, como lo expresa la ecuación 5.5:

$$DISTANCIA = N_{VUELTAS} * \pi * DIAMETRO \quad (5.5)$$

El sensor ya instalado en la máquina es el IME08-1B5PSZW2K de SICK y se muestra en la figura 5.4.

Figura 5.4: El sensor IME08-1B5PSZW2K.

A continuación se detallaran las características más importantes de este dispositivo:

- Tamaño de rosca: 8M.
- Alcance de detección S_n: 1.5mm
- Distancia de conmutación asegurada S_a: 1.215mm
- Frecuencia de conmutación máxima: 4KHZ
- Tipo de conexión: Cable de tres hilos, 2m
- Tipo de salida: PNP
- Características eléctricas: C.C (Corriente Continua) tres hilos
- Grado de protección: IP67
- Tensión de alimentación: entre 10V y 30V
- Consumo de corriente máximo sin carga: 10mA
- Demora antes de disponibilidad máxima: 100ms
- Histéresis: entre 5% y 15%
- Reproductibilidad máxima: 2%
- Desviación con la temperatura: 10%

Por último, se muestra en la figura 5.5 el diagrama de conexión de este sensor.

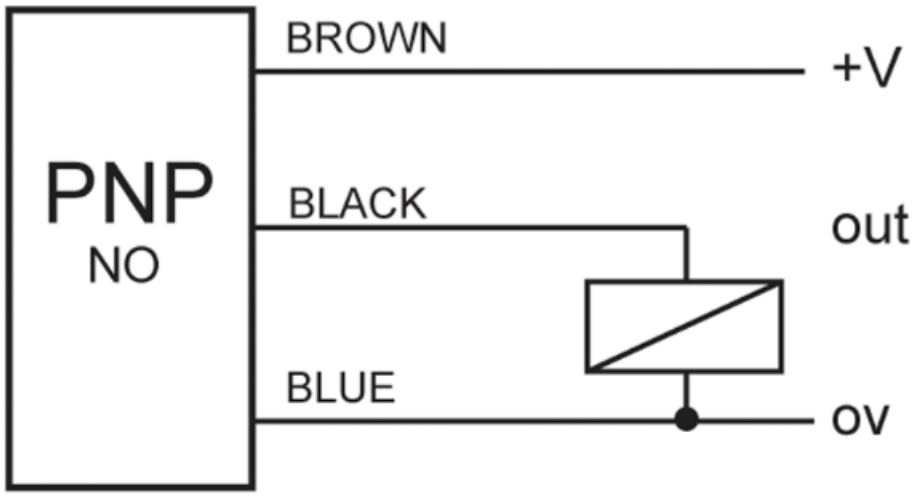


Figura 5.5: Esquema de conexión del sensor.

Este dispositivo se alimenta con corriente continua, tiene una conexión de tres hilos, salida tipo PNP y normalmente abierta.

5.2. Circuito de acondicionamiento

La señal del sensor inductivo debe ser procesada por uno de los canales de input capture del microcontrolador utilizado. Cada vez que aparece un pulso en este canal de entrada se producirá una interrupción a la que se la asocia con el número de secuencia actual del timer. De esta forma, en la rutina de interrupción se cuentan los números de pulsos y mediante el número de secuencia es posible verificar la frecuencia de la señal que ingresa, esto se analizara en el momento de presentar el código.

Sin embargo, no se puede ingresar directamente la salida del sensor inductivo en el microcontrolador, ya que operan con distintos niveles de tensión. Al sensor se lo alimenta con 15V provenientes de la fuente de alimentación, mientras que el µC trabaja con 3.3V. Por este motivo se acondiciona la señal de salida del sensor para que pueda ser ingresada en el microcontrolador.

El circuito utilizado es mostrado en la figura 5.6. La primera función que cumple dicho circuito es adaptar los niveles de tensión, por lo que se utiliza un simple divisor resistivo con la relación adecuada. Al divisor de tensión se le agregó un capacitor para filtrar ruido, con una constante de tiempo que permita el paso de la máxima frecuencia de conmutación del sensor. Por último, aparece un comparador con algunas resistencias, que conforman un comparador con histéresis. El resultado de añadir el comparador es aumentar la inmunidad al ruido para la señal que ingresa el microcontrolador.

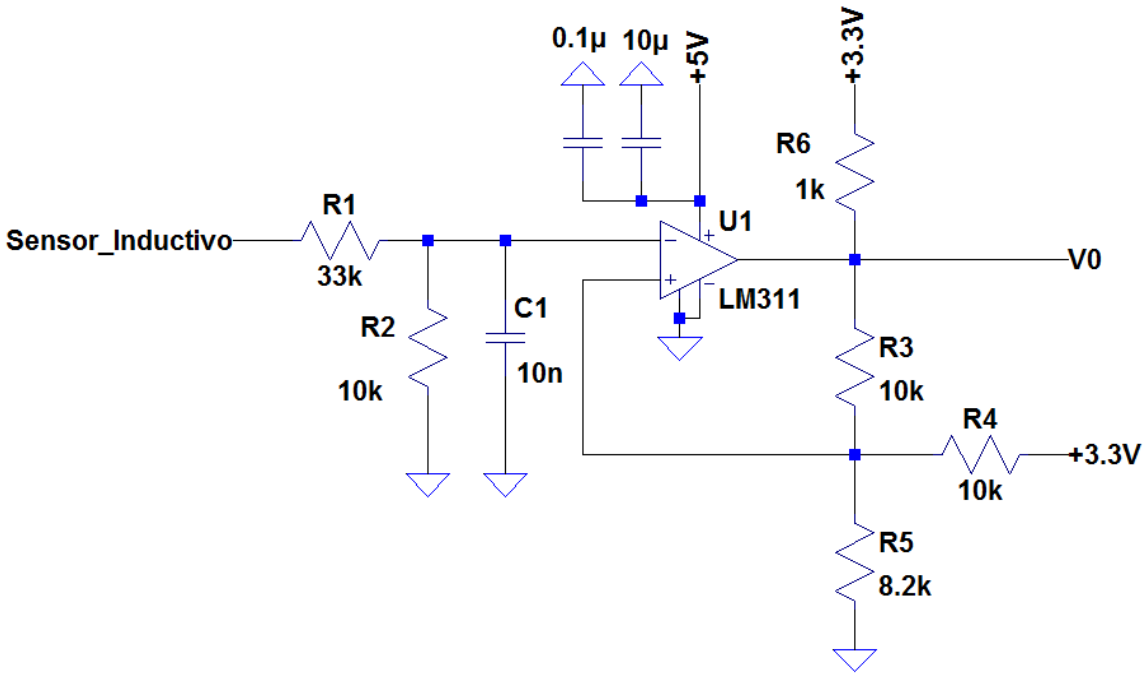


Figura 5.6: Circuito para el sensor inductivo.

Empezaremos el análisis de este circuito con los valores de las resistencias que integran el divisor de tensión, R_1 y R_2. La tensión de alimentación del sensor inductivo proviene de la fuente de alimentación y es ligeramente inferior a 15V. Por otro lado, la tensión de entrada al comparador debe ser menor a su tensión de alimentación, que es de 5V. Se decidió utilizar una relación entre R_1 y R_2 que permita pasar de 15V a aproximadamente 3.3V, además se adoptaron valores relativamente altos para no aumentar el consumo de corriente. En la ecuación 5.6 se presenta la tensión de salida del divisor cuando se ingresa con 15V:

$$V_{DIV} = \frac{R_2}{R_1+R_2} * V_{SENSOR} \cong \frac{10K\Omega}{10K\Omega+33K\Omega} * 15V \cong 3.49V \qquad (5.6)$$

Para seleccionar el valor del capacitor se necesita determinar el ancho de banda de interés, dado por la máxima frecuencia de conmutación del sensor de proximidad inductivo. El motor permite una máxima velocidad de 2840RPM, que mediante la reducción, se traduce en una velocidad máxima en la rueda de 376.66RPM. Se optó por dejar un margen respecto del valor máximo, limitando la velocidad en la rueda a 350RPM. Pero para determinar la velocidad máxima del sensor se debe recurrir a la ecuación 5.3, donde se ingresa con un número de dientes igual a 8 y se obtiene una velocidad en el sensor de 21.000RPM. Esta velocidad equivalente a una frecuencia de 350Hz. Para determinar el ancho de banda del divisor se toma cinco veces la frecuencia máxima del sensor, 1.75KHz, debido a que con cinco armónicos de la onda cuadrada la señal no se verá deformada. Resolviendo la transferencia de esta red se obtiene que la frecuencia de corte del filtro pasivo está dada por la ecuación 5.7:

$$f_C = \frac{1}{2\pi * C_1 * R_1 // R_2} \qquad (5.7)$$

Despejando C_1 resulta un valor de 11.85nF. Finalmente de adopta un valor comercial de 10nF para este capacitor.

El comparador utilizado es el LM311DR, cuyo esquema simplificado puede observarse en la figura 5.7.

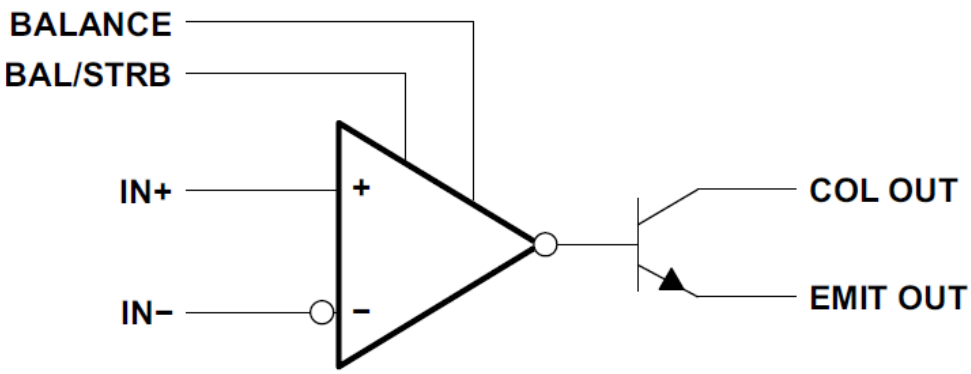


Figura 5.7: Esquema simplificado del LM311.

Se sale por el colector del transistor de salida mediante la resistencia de pull-up R_6 de 1KΩ a 3.3V. De esta manera se obtiene un nivel de tensión permitido por el microcontrolador. La tensión de 3.3V es sacada de la placa de desarrollo.

Mientras que las resistencias R_3, R_4 y R_5 junto la tensión de 3.3V aplicada a R_4, conforman la red que impone los niveles de comparación en la entrada no inversora del comparador. Esta configuración permite obtener dos niveles de comparación independientes uno del otro. Con los valores de la figura 5.6, cuando la salida del comparador está en alto el nivel de comparación se expresa con la ecuación 5.8:

$$V_{COMP} = \frac{R_5}{R_5 + R_3 // R_4} * V_{CC} = \frac{8.2K\Omega}{8.2K\Omega + 10K\Omega // 10K\Omega} * 3.3V = 2.05V \qquad (5.8)$$

Para que la salida vuelva a conmutar la tensión en la entrada inversora debe superar el valor calculado en la ecuación 5.8. Cuando esto suceda, el valor en la tensión de comparación será como lo indica la ecuación 5.9:

$$V_{COMP} = \frac{R_3 // R_5}{R_4 + R_3 // R_5} * V_{CC} = \frac{8.2K\Omega // 10K\Omega}{10K\Omega + 8.2K\Omega // 10K\Omega} * 3.3V = 1.025V \qquad (5.9)$$

Es decir que ahora la tensión de entrada deberá caer por debajo de este valor para que vuelva a conmutar la salida. Este nivel en la tensión de comparación proporciona una inmunidad al ruido de 2.275V, mientras que la V_{COMP} calculada en 8 tiene una inmunidad al ruido de 2.05V.

5.3 Simulación

Finalmente para demostrar como este circuito es capaz de adaptar y limpiar la señal entregada por el sensor inductivo, se mostrará el resultado de la simulación de este circuito en el programa LTSPICE. En la figura 5.8 se muestra de verde la señal del sensor inductivo para la velocidad máxima con un alto contenido de ruido y de azul la salida en el filtro pasivo.

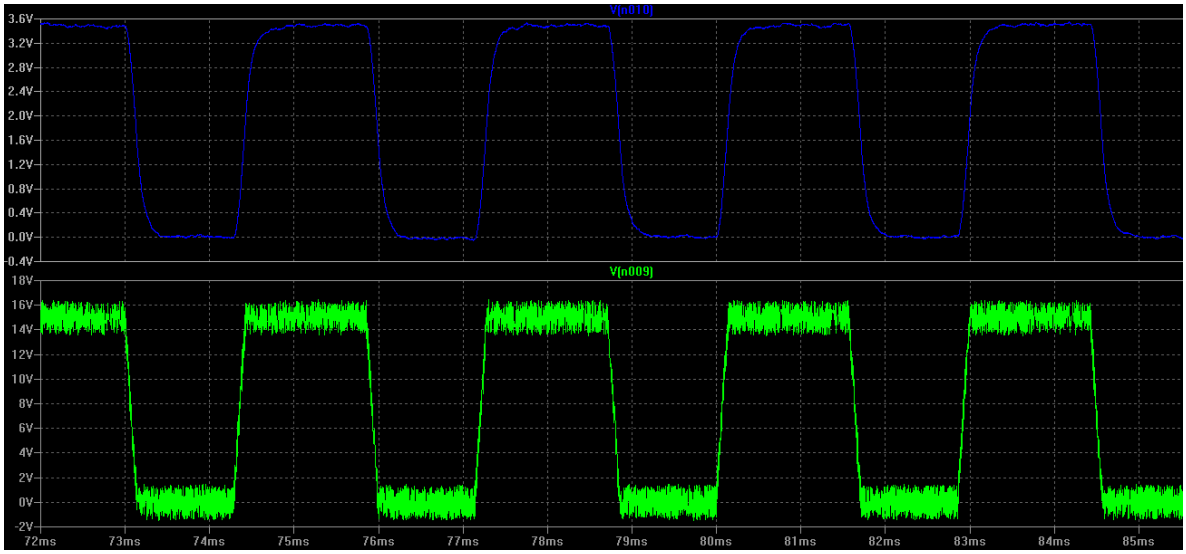


Figura 5.8: Señal del sensor en verde y salida del filtro pasivo en azul.

Se ve claramente como desaparece gran parte del ruido, como se baja el nivel pico de la tensión y una ligera deformación en la onda cuadrada, producto de los armónicos de alta frecuencia filtrados.

Por otro lado, en la figura 5.9 se presentan las formas de onda en el comparador. En el gráfico de abajo se aprecian las entradas al comparador y en el de arriba la salida del mismo. Cuando la señal en la entrada no inversora, la forma de onda de verde en esta simulación, está por encima del nivel de la entrada inversora, la señal de azul en el gráfico, la salida está en alto. En este momento la tensión de comparación está en 2.05V y cuando este valor sea superado por la señal de entrada, se producirá la conmutación de la tensión de salida. Siendo ahora 1.025V la tensión de comparación en esta nueva situación. Esto se mantendrá hasta que la señal de azul caiga por debajo de dicha tensión, momento en el que nuevamente conmuta la salida. Esta situación se repite de forma indefinida. Una última consideración es que la señal que finalmente entra al controlador esta invertida en fase con respecto a la salida de sensor.

5.4 Optoacoplador ranurado

También se requería de la instalación de otro sensor, que a diferencia del inductivo este no se encontraba instalado en la máquina, para verificar la correcta colocación de la carga y fundamentalmente para saber en qué momento empieza el ensayo.

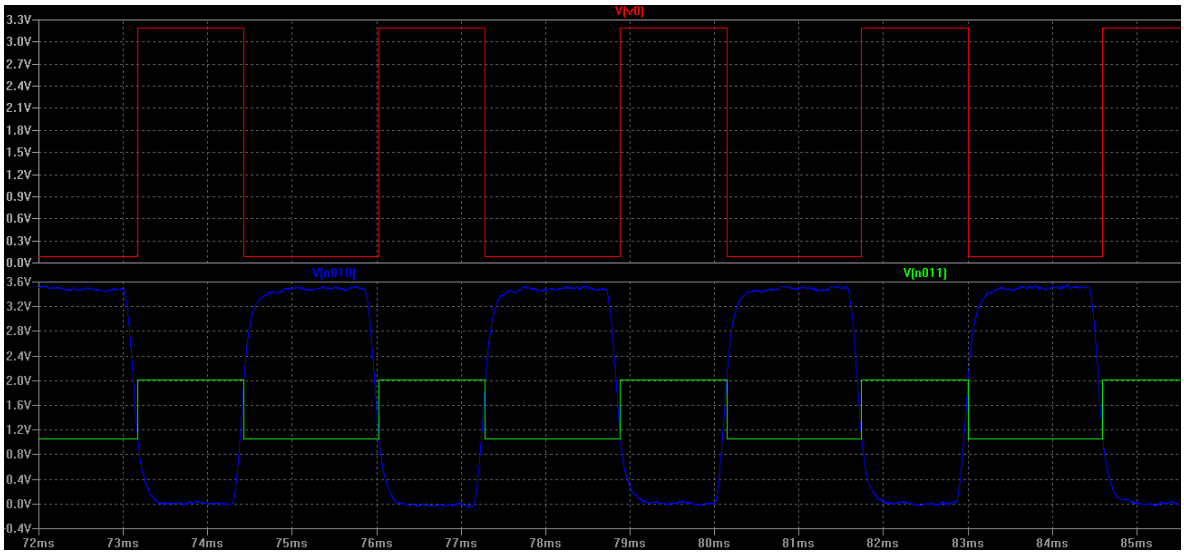


Figura 5.9: Formas de onda en el comparador.

Existen varios tipos de sensores que pueden cumplir con este propósito, como podría ser un sensor de fin de carrera, pero se optó por utilizar un optoacoplador ranurado. Este dispositivo ofrece una gran flexibilidad en cuanto a la instalación. El modelo utilizado es el HOA7720-M11 de la compañía Honeywell, cuya imagen se muestra en la figura 5.10 y su esquema interno en la figura 5.11.

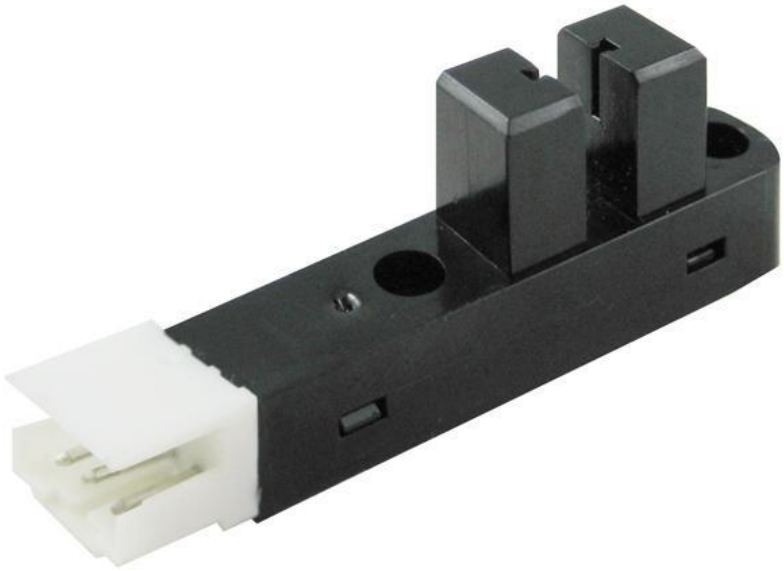


Figura 5.10: El sensor HOA7720-M11.

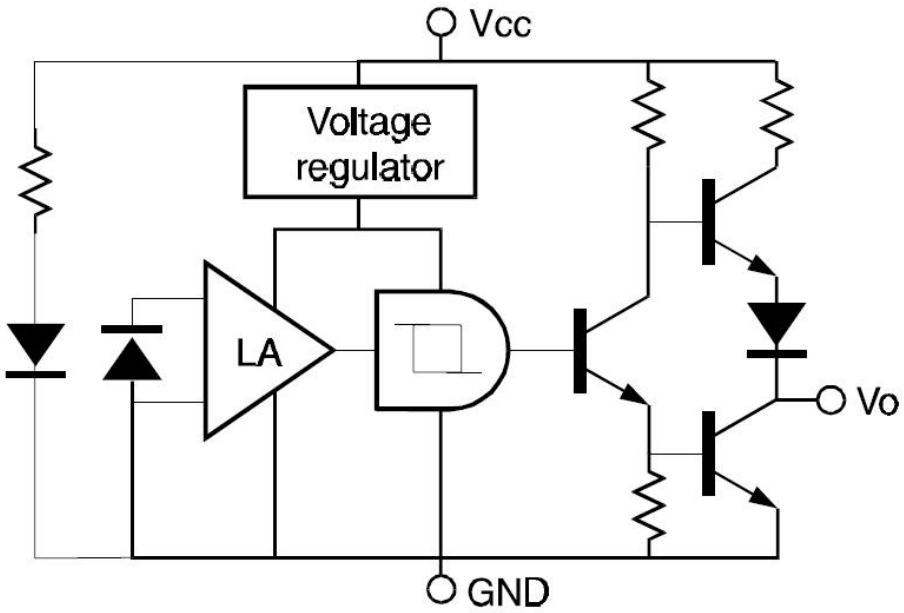


Figura 5.11: Esquema interno del optoacoplador ranurado.

Este dispositivo consiste de un diodo emisor infrarrojo enfrentado a un detector Optoschmitt. En el esquema se puede apreciar como el fotodetector está constituido por un fotodiodo, un amplificador, un regulador de tensión y un Schmitt trigger. En cuanto a la salida, esta serie ofrece de dos tipos, una tótem pole como se utilizó en este proyecto, y otra colector abierto.

La tensión de alimentación de este sensor es de 5V, suministrados por la fuente de alimentación, y presenta un consumo máximo de 40mA. Al trabajar con 5V, la salida de este dispositivo se conecta a una entrada tolerante a 5V en el microcontrolador. En cuanto a las características de la señal de salida, este dispositivo asegura un nivel de tensión en estado bajo no superior a 0.4V, mientras que en estado alto es superior a 2.4V.

La lógica es invertida, produciéndose un nivel alto en la salida cuando el camino óptico se interrumpe y un nivel bajo en caso contrario. De esta manera, mediante un arreglo mecánico, cuando la carga está en posición se interrumpe el paso de la luz y la salida conmuta de estado bajo a alto. Esta conmutación produce una interrupción en el microcontrolador, resultando finalmente en el comienzo del ensayo y la recolección de datos.

Este dispositivo no requiere de ningún tipo de circuito externo y presenta tres cables de conexión, el de alimentación, masa y salida.

6

Microcontrolador

6.1 Procesador

A la hora de elegir el microcontrolador, como primer paso se decidió que el procesador debía ser ARM. Esta elección se basó en la creciente popularidad que tiene el mismo en la industria y en las ventajas que ofrece.

La arquitectura ARM se diseñó para permitir implementaciones de tamaño muy reducido y de alto rendimiento. Estas arquitecturas tan simples permiten dispositivos con muy bajo consumo de energía. Se caracteriza fundamentalmente por ser una computadora de set de instrucciones reducido (Reduced Instruction Set Computer, RISC), como lo indica su propio nombre, ARM (Advanced RISC Machine).

Este tipo de procesadores presenta las siguientes ventajas:

- Bajo consumo de energía.
- Arquitectura simple de 32 bits.
- Performance y baja densidad de código.
- Buen manejo de interrupciones con 256 niveles de prioridad.
- Más de 30 sistemas operativos para utilizar.
- Portabilidad y reutilización de código, con más de 15 compañías desarrollando microcontroladores con procesadores ARM.

ARM no crea microcontroladores, diseña procesadores y varios componentes que los creadores de controladores necesitan. Estos diseños son llamados “Intellectualy Property” (IP) y el modelo de negocio se basa en licencias IP.

En el diseño de un microcontrolador, el procesador toma solo una parte pequeña del área del silicio. El resto del área la ocupan las memorias, los relojes, PLL, el bus y los periféricos como muestra la figura 6.1. Aunque muchos vendedores de microcontroladores usan procesadores ARM, el sistema de memoria, los periféricos y las características de operación

(voltaje, velocidad) pueden ser completamente diferente de un producto a otro. Esto permite a las compañías que realizan microcontroladores introducir ventajas y recursos adicionales en sus productos, haciéndolos diferentes a los del resto del mercado.

En la actualidad hay más de 15 empresas que desarrollan microcontroladores con procesadores ARM. Entre ellas están: Analog Devices, Atmel, Cypress, EnergyMicro, Freescale, Fujitsu, Holtek, Infineon, Microsemi, Milandr, NXP, Samsung, Silicon Laboratories, ST Microelectronics, Texas Instrument y Toshiba.

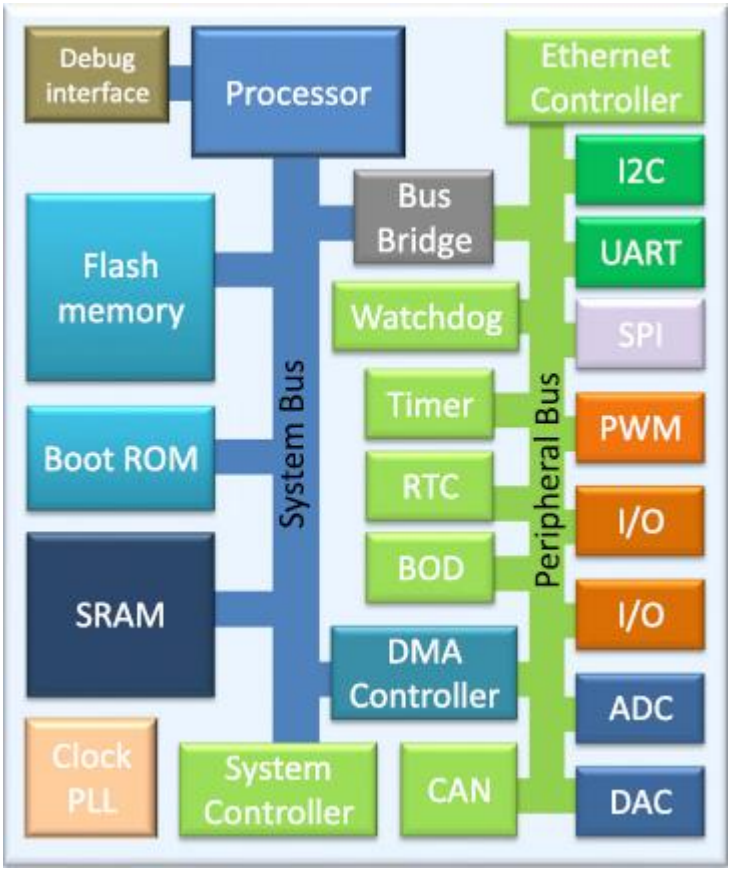


Figura 6.1: Diagrama en bloques del microcontrolador.

6.2 Elección del microcontrolador

Como se mencionó, hay una amplia variedad de microcontroladores con procesadores ARM. Para seleccionar el más adecuado primero hay que saber que demanda el trabajo que se está realizando. Algunos puntos clave a tener en cuenta son:

- Periféricos e interfaces.
- Requerimientos de memoria.
- Consumo de energía.
- Performance y máxima frecuencia.
- El package.
- Condiciones de operación (voltaje, temperatura).
- Costo y disponibilidad.

- Herramientas, software y soporte para el desarrollo.

En el caso de nuestro proyecto, los ítems que más pesaron en la elección fueron:

- Periféricos e interfaces:

Necesitábamos que poseyera un ADC, una interfaz SPI y un DAC. Los dos primeros se encuentran en la mayoría de los microcontroladores actuales, pero el DAC fue importante en la elección, ya que este periférico no se encuentra comúnmente en los microcontroladores.

- Herramientas, software y soporte para el desarrollo:

Este fue un punto muy importante en la selección, ya que éramos novatos con este tipo de tecnología.

Teniendo en cuenta lo mencionado, se optó por la placa de desarrollo NUCLEO con el microcontrolador STM32F410RB de la empresa ST Microelectronics, la cual se muestra en la figura 6.2. Como esta aclarado en la imagen, la placa cuenta con el programador incorporado, ahorrando la necesidad de comprar uno. Además cuenta con un cristal, botón de reset, y todos los pines del microcontrolador disponibles, con fácil acceso a partir de conectores. Esto ayudó y agilizó mucho el trabajo para empezar a trabajar con el microcontrolador.

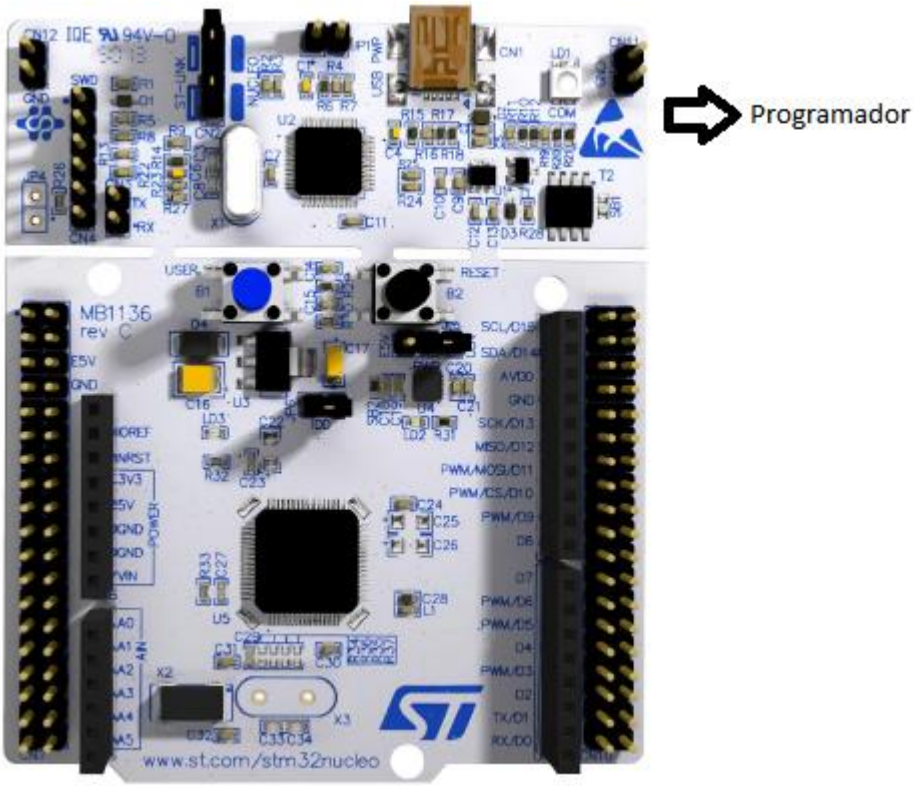


Figura 6.2: Placa Nucleo.

Estamos conforme con la elección del microcontrolador y con la empresa ST. Esta brinda mucha información y soporte para el desarrollo. Posee un software que fue de mucha utilidad en los inicios. Este es el STM32CubeMX, el cual seleccionando el microcontrolador que se está utilizando permite setear y ajustar los periféricos de forma gráfica, creando un proyecto en el

IDE deseado con los valores preestablecidos. Las figuras 6.3 y 6.4 muestran cómo se desarrolla la inicialización del microcontrolador de forma gráfica.

Aunque con el agregado a mitad del proyecto de guardar datos en una memoria externa, se hubiera optado por otro microcontrolador con USB-Host o USB-OTG para comunicarnos con un Pen-Drive.

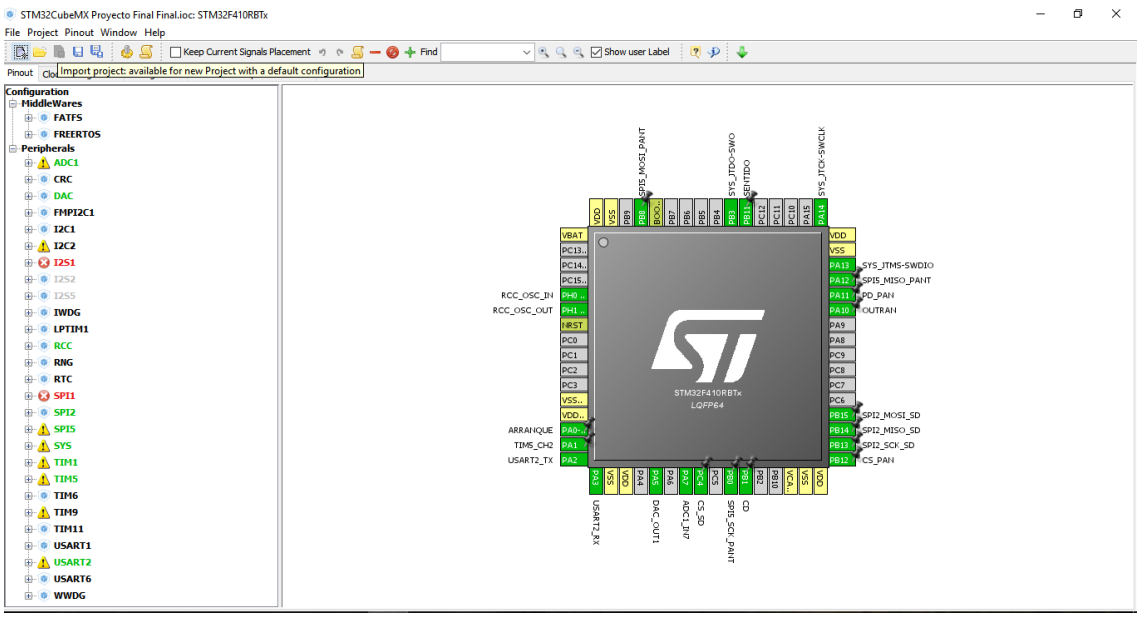


Figura 6.3: Configuración de pines.

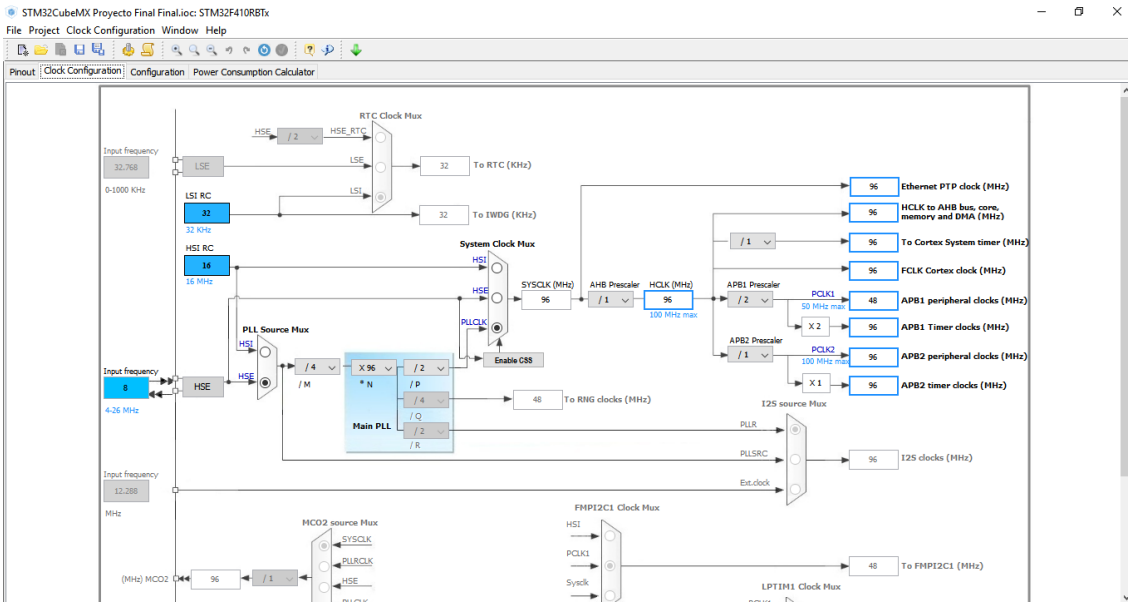


Figura 6.4: Configuración de relojes.

7

Interfaz Gráfica

7.1 El módulo de desarrollo VM800B

Es necesario contar con una interfaz en la cual el usuario pueda setear los parámetros del ensayo y ver los resultados que obtiene del mismo. Para ello se utilizó una pantalla táctil de la compañía FTDI denominada VM800B que se puede observar en la figura 7.1. Posee una dimensión de 5” y un display con una resolución de 480x272 pixeles.

Figura 7.1 Pantalla VM800B.

Viene integrada con el controlador FT800 de la misma compañía. Este controlador sensa cuando se toca la pantalla resistiva y además posee un sintetizador de sonido con una salida PWM para manejar un speaker. El dispositivo es controlado mediante una comunicación SPI o I2C con cualquier clase de microcontrolador que posea uno de estas interfaces. Un diagrama del sistema se puede observar en la figura 7.2.

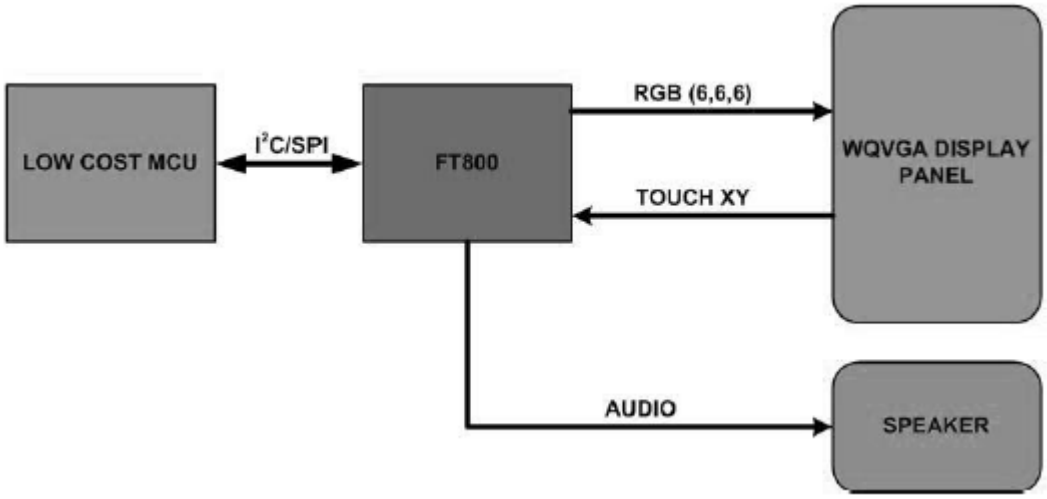


Figura 7.2: Diagrama del sistema de la interfaz.

El diagrama en bloques del controlador se puede observar en la figura 7.3.

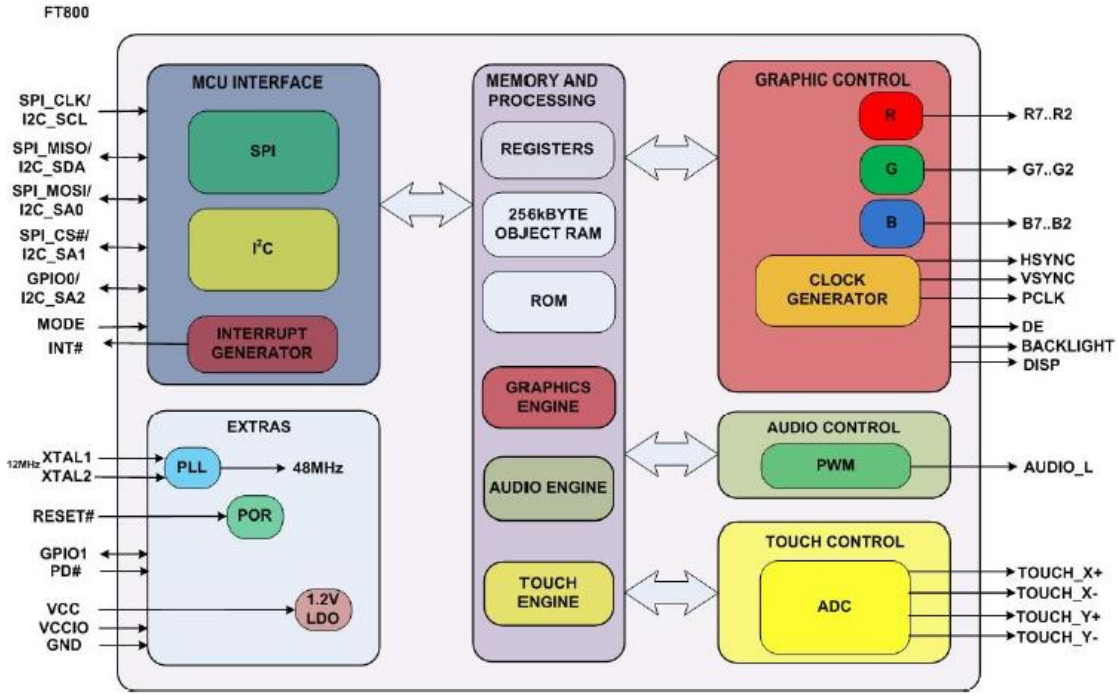


Figura 7.3: Diagrama en bloques FT800.

En el bloque de Memory and Processing se encuentra todo el firmware que nos permite interactuar con la pantalla. Graphics Engine y Audio Engine contiene primitivas que mediante el envío de los comandos adecuados podemos observar o escuchar distintas imágenes, iconos, formas y sonidos ya preseteados en el controlador FT800.

7.2 Programación

Para no tener que estar mirando la hoja de datos cada vez que se desea insertar una forma o un “widget” en una nueva pantalla, la empresa FTDI proporciona dos softwares, FTDI EVE SCREEN DESIGNER y FTDI EVE SCREEN EDITOR para poder crear las pantallas de forma más sencilla y visual. Los dos softwares son muy parecidos pero en nuestro caso trabajamos con el FTDI EVE SCREEN DESIGNER. En la figura 7.4 se puede ver como se conforma el programa.

En la parte de Widget Selection arrastramos lo que deseamos utilizar a la pantalla. Una vez que sea terminado el diseño, en Output Window se encuentran todos los comandos que le debemos enviar al controlador FT800 para que muestre la pantalla deseada.

En el caso de usar botones, slider o algún Widget en el cual se procede a actuar de acuerdo al touch, se debe añadir al código los correspondientes comandos TAG. Como se muestra en la figura 7.5, el controlador registra la posición XY en donde la pantalla ha sido presionada. Para el fácil manejo de botones, sin necesidad de saber todas las posiciones XY que abarca, se le puede asignar un valor TAG .Por ejemplo si se tiene un botón y se le asigna TAG = 1, para saber si ha sido presionado se debe verificar si el REGISTRO_TAG = 1.Este registro mientras no se presione la pantalla vale 0.

Figura 7.4: EVE SCREEN DESIGNER.

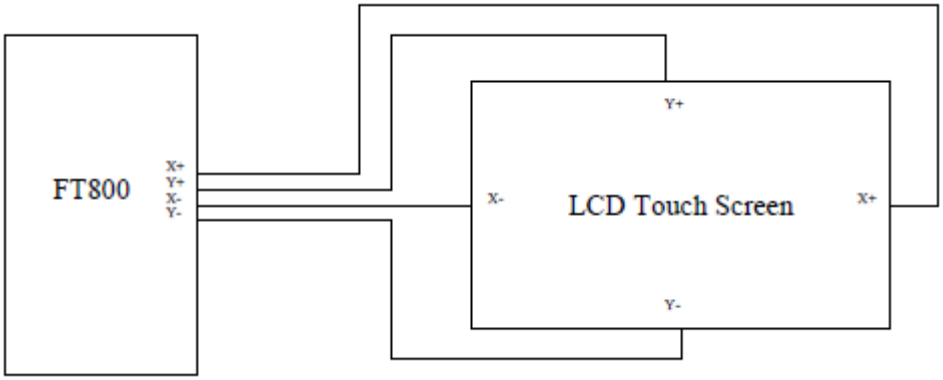


Figura 7.5: Configuración Touch.

7.3 Conexión

El microcontrolador general debe proveer una conexión SPI o I2C actuando como maestro y el FT800 se comporta como esclavo. La comunicación SPI contempla lo siguiente:

- Un máximo de transferencia de 30Mbps
- SPI modo 0
- Bit más significativo primero (MSB)

De acuerdo a la imagen de la pantalla de la figura 7.6 el conexionado es el siguiente:

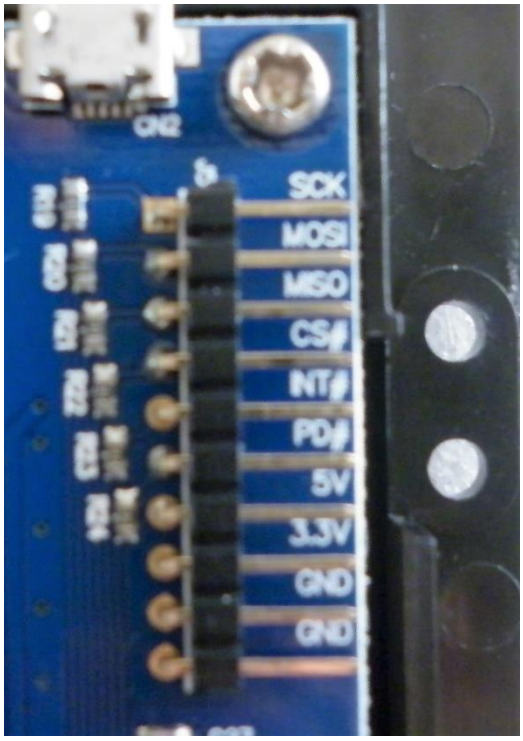


Figura 7.6: Intefaz.

- SCK – SPI clock
- MOSI – Master Out / Slave In – datos del controlador al FT800
- MISO – Master In / Slave Out –datos del FT800 al controlador
- CS# – SPI Chip Select, activo bajo
- INT# – salida de interrupciones del FT800
- PD#– entrada Power down del FT800
- GND- masas
- +3.3- alimentación
- +5V- alimentación

Se puede optar por alimentar la pantalla con 5V o 3.3V. El fabricante recomiendo utilizar una fuente de 5V/1A.

7.4 Conclusión

La pantalla no contiene gran cantidad de objetos gráficos y no se le pueden agregar nuevos a comparación de si se hubiera usado un pad Android. Otro punto es que al ser resistiva cuesta acostumbrase a tener que presionar la pantalla, pero le permite al usuario poder emplearla con guantes. Como conclusión, creemos que la pantalla cumple ampliamente con los requisitos requeridos para la aplicación, teniendo en cuenta que se va a encontrar en un taller de mecánica. Para el caso de una aplicación que requiera más interacción con el usuario, creemos que es aconsejable trabajar con un sistema operativo y un lenguaje de programación más alto que C.

8

Almacenamiento de Datos

8.1 SD

Mientras se fue desarrollando el proyecto, se observó que además del gráfico en la pantalla es conveniente que se puedan almacenar los datos del ensayo en una memoria no volátil, para que luego el usuario pueda trabajar con ellos en su computadora.

El puerto más común en las computadoras de la actualidad es el puerto USB, por esta razón hubiera sido una buena opción almacenar los datos en un Pendrive. Pero el microcontrolador ya había sido escogido y no posee un periférico USB-Host o USB-OTG para el manejo del protocolo. Se podría haber usado algún integrado que nos permita manejar el protocolo y funcione como USB-Host comandado por nuestro microcontrolador, pero resultaba en un costo y circuitería adicional. Por esta razón se optó por almacenar los datos en una tarjeta de memoria SD “Secure Digital”.

Las tarjetas SD están basadas en una memoria flash controladas por un microprocesador interno. Las mismas fueron diseñadas con el fin de tener gran capacidad de almacenamiento y seguridad, principalmente para su uso en tecnologías móviles. Esto se debe a que presentan bajo consumo y un tamaño muy pequeño. Los tamaños estándares son los siguientes:

- SD Dimensiones: 32x24x2.1 mm Peso: 2gr
- microSD Dimensiones: 11x15x1.0 mm Peso: 0.5gr

En este caso se utilizó una ranura para tarjetas SD, ya que no hay problema de espacio y las microSD son muy pequeñas y difíciles de manipular.

La comunicación con la tarjeta no se hace directamente con la memoria, sino con el microprocesador que incluye y se puede observar en la figura 8.1. Por tanto esta se realiza mediante comandos que se proporcionan en el manual de la misma. Las tarjetas SD tienen dos maneras de comunicarse, una mediante el protocolo propietario SD y el otro mediante una interfaz SPI.

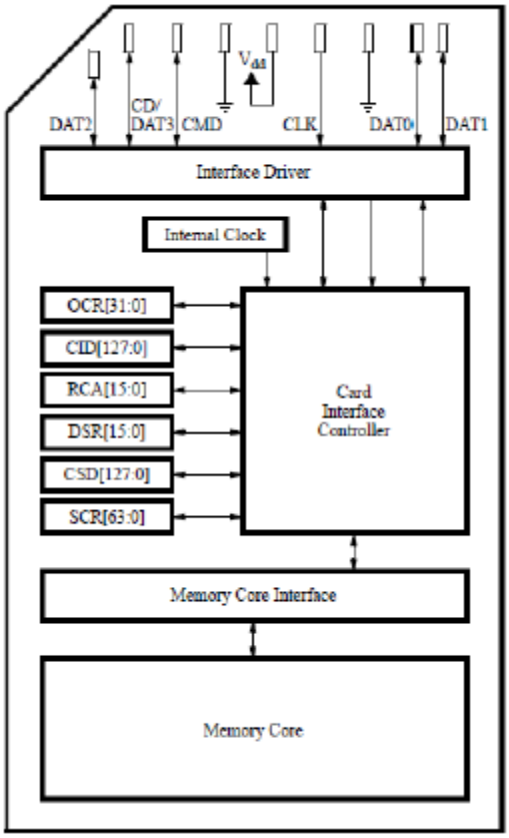


Figura 8.1: Diagrama en bloques de una tarjeta SD.

El protocolo SD utiliza todos los pines de la tarjeta. Donde los cuatro pines de datos son bidireccionales, a diferencia del SPI que para los datos se usan dos pines y estos son unidireccionales. Por lo que la comunicación SD es mucho más rápida, alcanzando 50MB/sec.

Ambas formas de comunicación utilizan una señal de reloj de entre 0 a 25MHz y alimentación entre 2.0V a 3.6V. El consumo de corriente en la operación de escritura puede ser superior 10mA y se aconseja proveerle 100mA. Cuando no está siendo utilizada, el host no le envía comandos, pasa automáticamente a modo de espera y casi no consume corriente.

Como se ha mencionado, la comunicación SD es mucho más rápida a comparación de la SPI, pero en este caso el microcontrolador empleado no posee periférico MMC (MultiMedia Card) para poder desarrollarlo y la velocidad de transferencia no es crítica. Entonces se ha empleado el modo SPI. Los pines utilizados de la tarjeta se observan en la figura 8.2, en este caso no se utilizan los pines 8 y 9 que corresponden al modo SD.

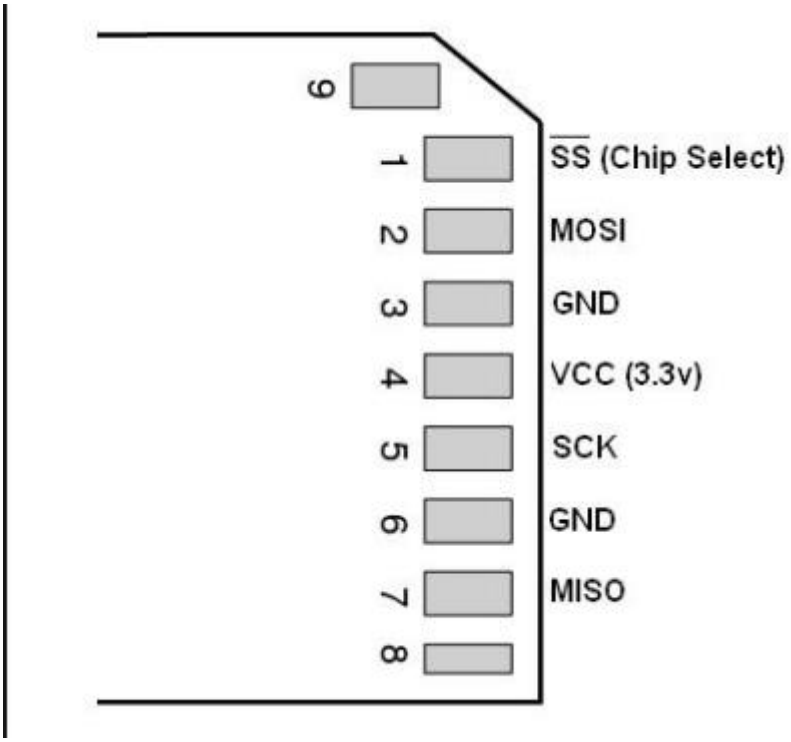


Figura 8.2: Pines de un tarjeta SD.

8.2 Implementación

Para controlar la tarjeta e interpretar los datos de la comunicación se necesita de software adicional. Este software va desde la capa de aplicación hasta la capa física, que en este caso es la comunicación SPI. Esto se puede observar en la figura 8.3.

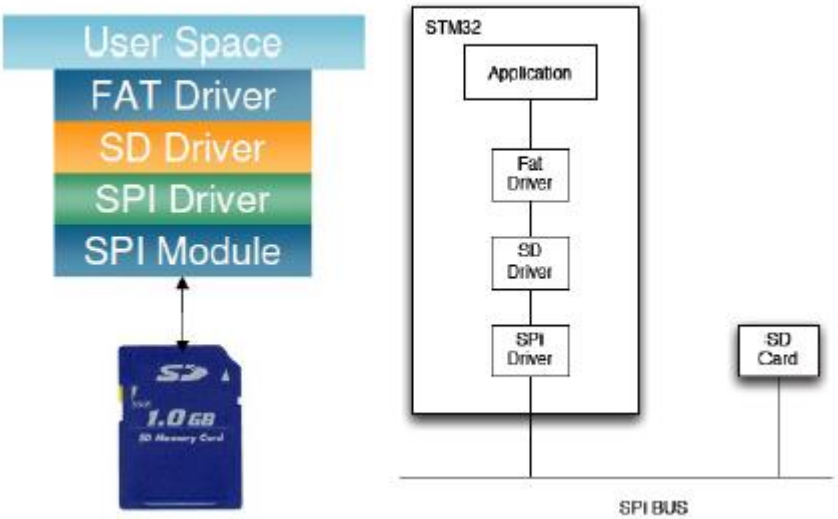


Figura 8.3: Capas de la comunicación.

La información en una tarjeta SD es organizada como un sistema de archivos. Este se implementa en el bloque denominado “FAT FS”. Para tal fin se utiliza el módulo genérico FAT FS disponible sin restricciones de uso. Dicho módulo implementa todas las funciones necesarias para acceder a unidades de memoria con un sistema de archivo FAT 12, 16 ó 32.

Una aplicación que desea acceder a datos almacenados o escribir en alguna posición de memoria utiliza comandos tales como open, read y write, que son provistos por FAT file system driver. Un nivel más abajo, SD driver, implementa los comandos, comunicándose en este caso con la capa física SPI.

Afortunadamente, no es necesario desarrollar el código de todas las capas, pues están estandarizadas las capas FAT driver y SD driver y son de libre acceso. Esto se debe a que los sistemas de archivos FAT son iguales para todas las tarjetas e implementan los mismos comandos. Lo que hay que desarrollar es la capa física, ya que se puede realizar una comunicación MMC o como en este caso SPI. Además los periféricos varían de acuerdo al microcontrolador que se esté utilizando.

Para desarrollar la comunicación con la SD se descargó la librería FAT de http://elm-chan.org/fsw/ff/00index_e.html y se desarrolló la capa inferior SPI en diskio.c para comunicar el hardware con esta librería.

9

Programa

9.1 Estado *Inicio*

El programa se puede describir como una máquina de estados. Los 9 estados utilizados se denominaron de la siguiente manera: *Inicio, Tipicos, Seteo, DecisionGuardar, NombreArchivo, InicioEnsayo, Ensayando, Standby* y *Rectificacion.*

A continuación se describe que función cumple cada uno. El estado *Inicio* se identifica por la primer pantalla que aparece al encender la máquina. Como muestra la figura 9.1, el usuario puede elegir si desea realizar un ensayo o rectificar la rueda de goma.

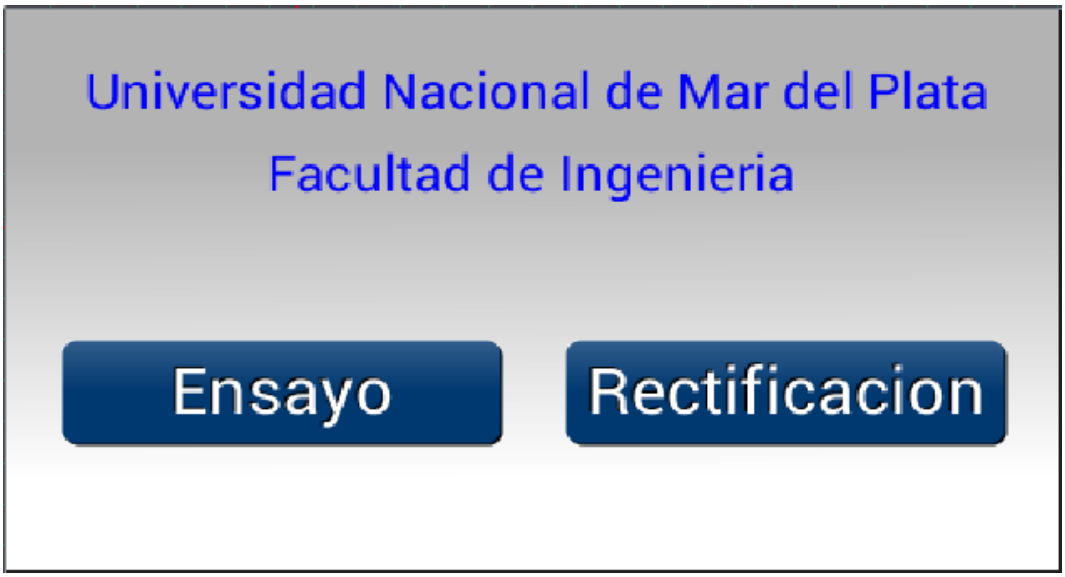


Figura 9.1: Pantalla Inicio.

Si se presiona el botón Ensayo se ingresa al estado Tipicos. Si se presiona el botón Rectificacion se va al estado Rectificacion.

9.2 Estado *Tipicos*

En este estado el operario debe seleccionar uno de los tres ensayos predefinidos, como muestra la figura 9.2.

Cada uno de los ensayos predefinidos posee seteado los siguientes parámetros, que luego podrán ser modificados en el estado Seteo.

Procedimiento A

Distancia: 4309 m. Duración: 30 min.

Velocidad de giro: 200 RPM. Diámetro: 228.6 mm.

Fuerza: 130 N.

TRIBOLOGIA LAB 5

Distancia: 718.2 m. Duración: 5 min.

Velocidad de giro: 200 RPM. Diámetro: 228.6 mm.

Fuerza: 75 N.

TRIBOLOGIA LAB 10

Distancia: 1436.3 m. Duración: 10 min.

Velocidad de giro: 200 RPM. Diámetro: 228.6 mm.

Fuerza: 75 N.

Figura 9.2: Pantalla *Tipicos.*

Cuando se presiona cualquiera de los ensayos se va al estado Seteo. Si se presiona Volver se va al estado Inicio.

9.3 Estado *Seteo*

Permite observar y modificar los parámetros del ensayo, como muestra la figura 9.3.

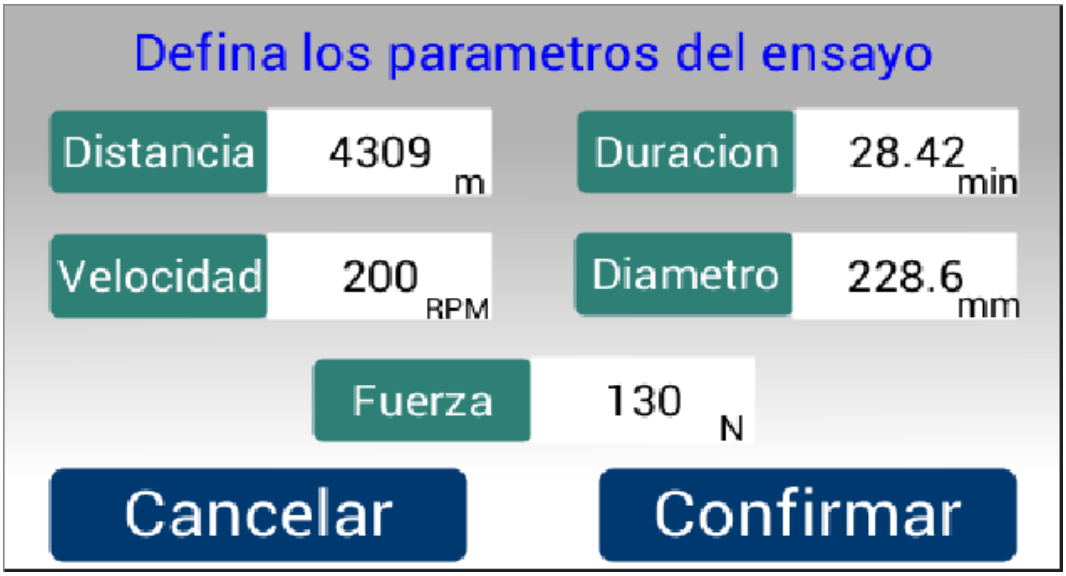


Figura 9.3: Pantalla Seteo.

Para modificar algún parámetro se debe presionar el botón del mismo y se abrirá una nueva pantalla, como se ilustra en la figura 9.4. En el teclado se ingresa el valor deseado y al presionar Enter se retorna a la pantalla Seteo de la figura 9.3.

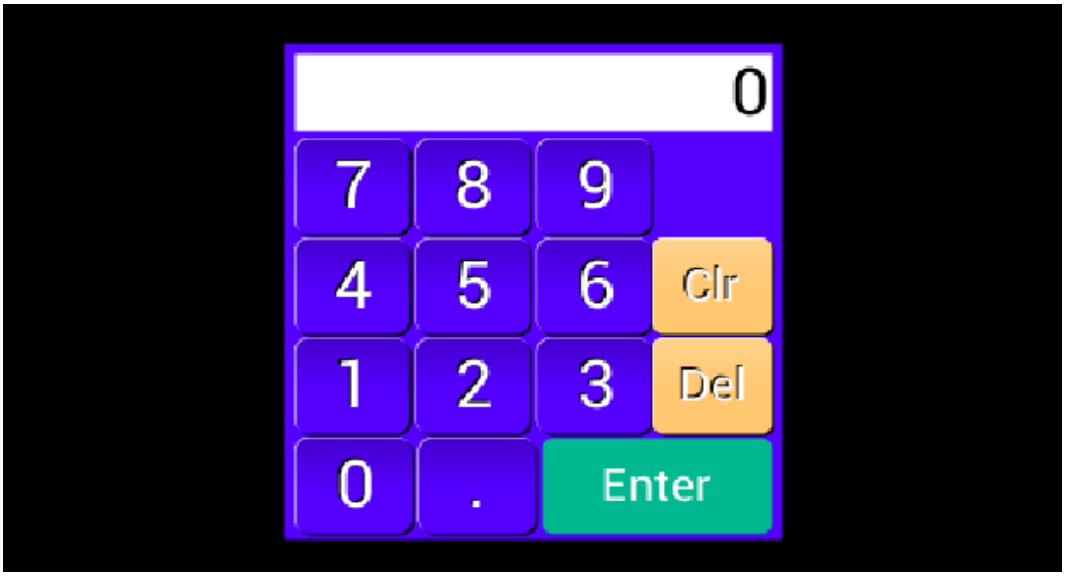


Figura 9.4: Teclado parámetros.

Al presionar el botón Confirmar el programa verifica que los datos ingresados estén dentro de los márgenes diseñados para el ensayo. Los valores que se pueden ingresar son los siguientes:

- 50 < Frecuencia < 350 [RPM]
- 200 < Diámetro < 300 [mm]
- 0 < Distancia < 17236 [m]
- 0 < Fuerza < 294 [N]
- 5 < Duración < 120 [min]

Al modificar la distancia se modifica la duración y viceversa automáticamente. Aunque al que más peso se le da en el ensayo es a la distancia, por esta razón cuando se modifica la frecuencia el valor de la distancia se mantiene y se corrige la duración. Al variar el diámetro del disco se conserva el valor de la distancia y se ajusta la duración.

Si los valores no están en el rango establecido, luego de presionar Confirmar el programa no pasa al siguiente estado. Aparece una pantalla de aviso, como se muestra en la figura 9.5.

Figura 9.5: Pantalla Error de configuración.

9.4 Estado *DecisionGuardar*

A este estado se accede una vez que se terminó de verificar que los valores ingresados son adecuados para realizar un ensayo. En éste lo primero que se realiza es preguntarle al usuario si desea almacenar los datos del ensayo, figura 9.6.

Si se presiona el botón NO, se va al estado InicioEnsayo directamente. Si se presiona el botón SI se verifica que se haya ingresado la tarjeta de memoria SD, donde se van a almacenar los datos. Para saber esto, el zócalo donde se ingresa la tarjeta posee un interruptor mecánico que cortocircuita uno de sus pines, denominado Card Detect, con GND en presencia de la tarjeta. El pin Card Detect se lo conectó a una de las entradas digitales del microcontrolador, entonces cuando la tarjeta no está conectada el microcontrolador percibe un nivel alto en ese pin y

cuando está conectada un nivel bajo. Si no se detecta la tarjeta se envía un mensaje de aviso, figura 9.7.

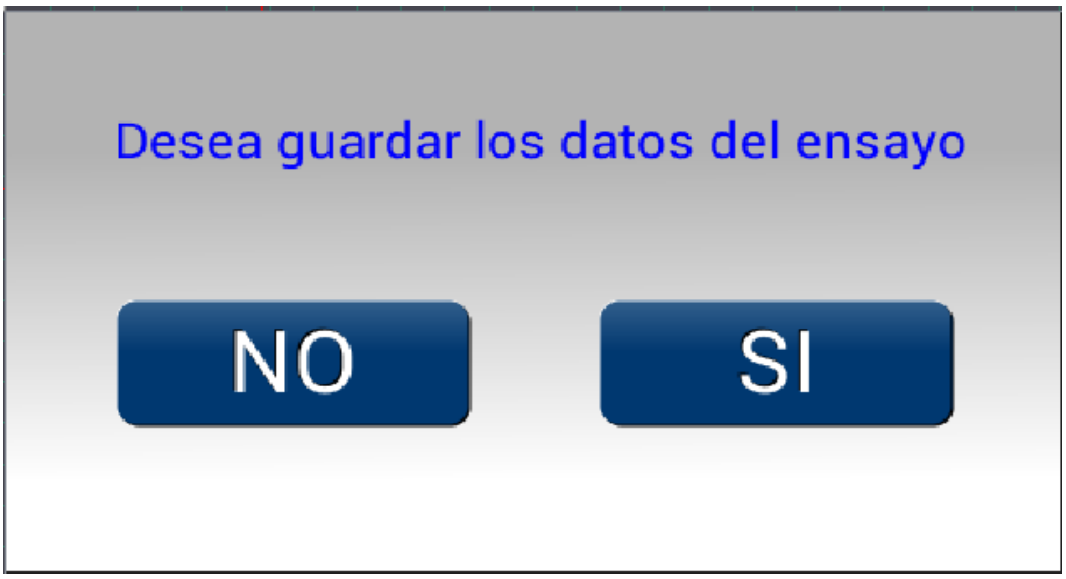


Figura 9.6: Pantalla Decisión Guardar.

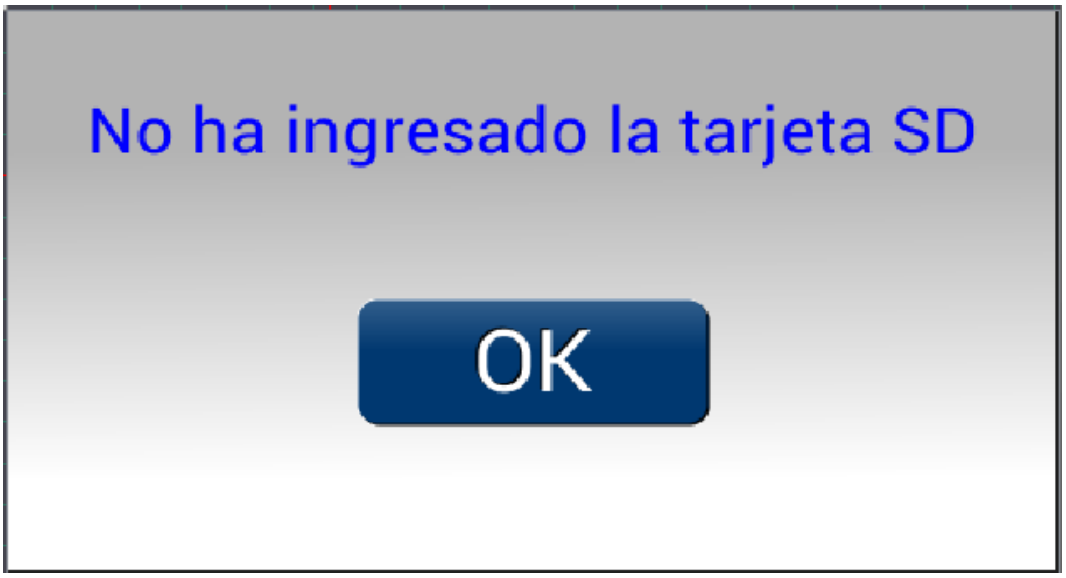


Figura 9.7: Pantalla de aviso cuando no se detectó SD.

En esta pantalla se espera hasta que el usuario presione OK y se vuelve a la pantalla de la figura 9.6. Si la tarjeta fue detectada se pasa al estado NombreArchivo.

9.5 Estado *NombreArchivo*

En este estado se debe seleccionar el nombre con el cual será guardado el archivo que contendrá los datos del usuario. El mismo será de formato CSV para poder ser abierto en cualquier hoja de cálculo con la separación en columnas y filas para mejor visualización y posterior análisis. Para nombrarlo se creó un teclado en la pantalla como muestra la figura 9.8.

Figura 9.8: Teclado Archivo.

Presionando el botón shift se cambia el formato de la teclas, de mayúscula a minúscula y vicerversa. Presionando el botón 123 se presenta un teclado numérico como se muestra en la figura 9.9.

Figura 9.9: Teclado numérico Archivo.

Presionado el botón Abc se vuelve al teclado alfabético de la figura 9.8. Al presionar el botón Enter en cualquier de los teclados, alfabético o numérico, se pasa al estado InicioEnsayo.

9.6 Estado *InicioEnsayo*

En este estado se hacen las últimas verificaciones previas a que comience el ensayo. Al presionar el botón Arrancar de la pantalla, figura 9.10, se pone a girar el motor y se toma una muestra de la tensión de salida del amplificador de la celda de carga, por medio del ADC. El valor de esa medición se utiliza como offset, correspondiente a la salida del amplificador con la celda sin carga. Ese offset luego se lo resta a todas las medidas captadas en el ensayo. Vendría a ser análogo a la función Tara en una balanza.

Cabe aclarar que antes de tomar esa medida, debe haber pasado un tiempo superior a 4 minutos desde que se encendió la maquina hasta ese momento. Esto es a causa de que los amplificadores y fundamentalmente la celda de carga, poseen una deriva con la temperatura. Cuando apenas se conecta la máquina, tanto los amplificadores como la celda están a temperatura ambiente produciendo una determinada tensión de salida. A medida que la placa está en funcionamiento la temperatura aumenta produciendo que se eleve el nivel de offset. Por medio de ensayos se descubrió que la placa alcanza una temperatura estable, produciendo un determinado nivel de offset que se mantiene constante a lo largo del tiempo, luego de 4 minutos de estar conectada. Esta teoría se comprobó midiendo la temperatura externa de los encapsulados.

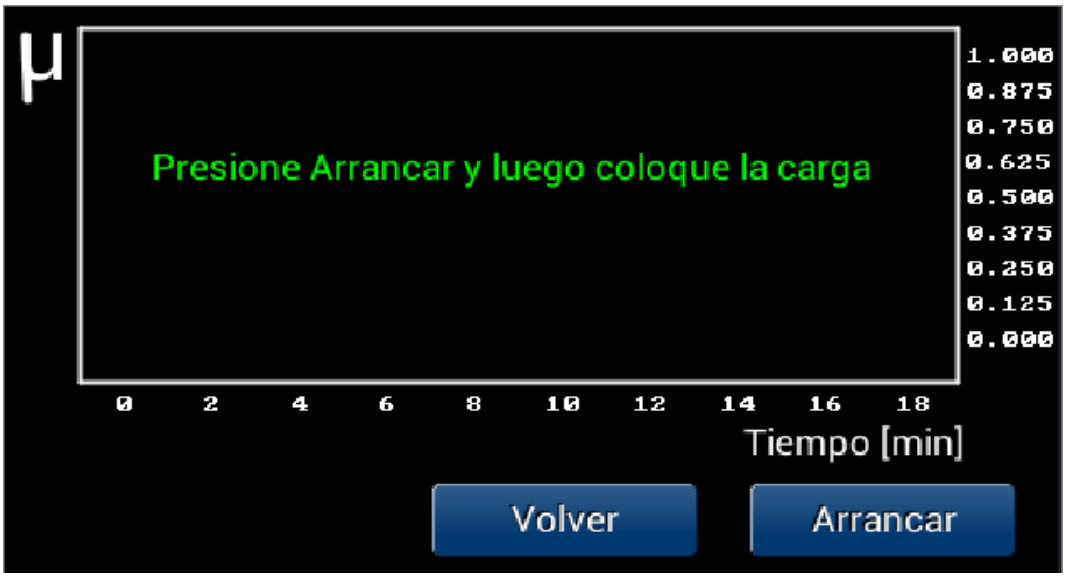


Figura 9.10: Pantalla Inicio ensayo.

Entonces si se presiona el botón Arrancar y la máquina ha estado encendida menos de 4 minutos, aparece la pantalla de la figura 9.11. Esta va a permanecer graficada hasta que pase el tiempo establecido, luego se toma el offset y se verifica que la tarjeta SD ingresada se encuentre en condiciones para poder almacenar los datos. Si se encuentra dañada o sin espacio

aparece una pantalla de aviso, la cual corresponde a la figura 9.12. El usuario puede optar por Salir y volver al estado Inicio, o continuar con el ensayo a pesar que no se guardaran los datos.

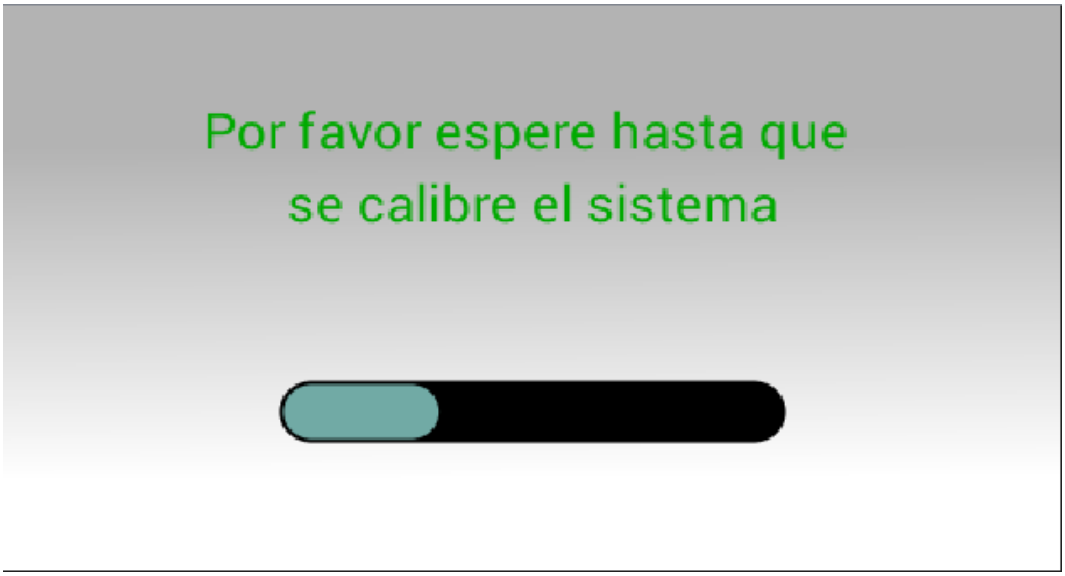


Figura 9.11: Pantalla de espera para la calibración.

Si se prosigue con el ensayo la siguiente pantalla es la de la figura 9.13, el programa se queda esperando que se coloque la carga. Cuando esto suceda, el optoacoplador ranurado entregara un nivel alto a la entrada digital elegida del microcontrolador.

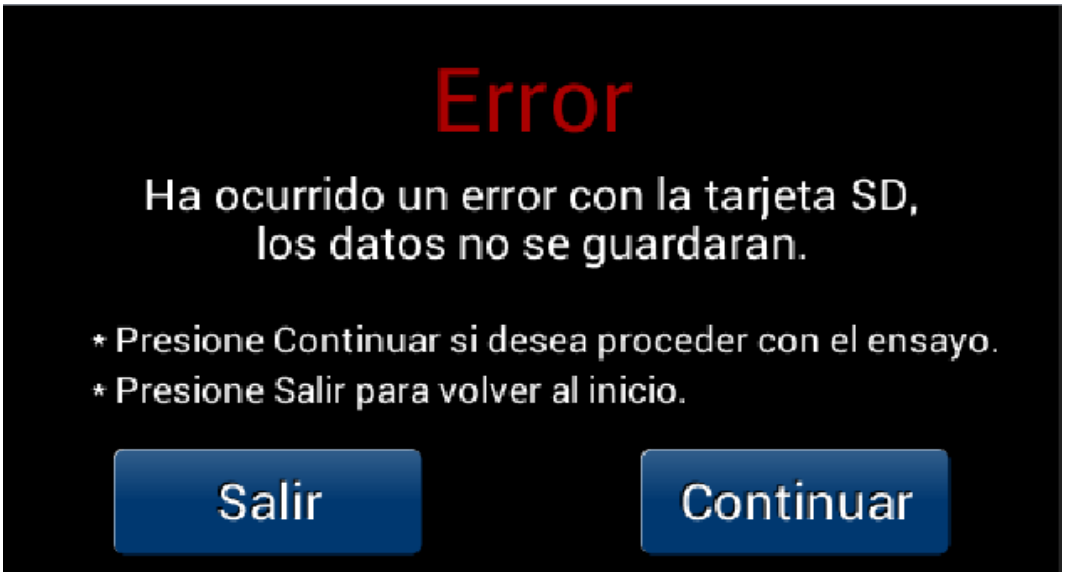


Figura 9.12: Pantalla de error tarjeta SD.

Si se presiona Detener se apaga el motor y se vuelve a la pantalla de la figura 9.10. Si se coloca la carga se va al estado Ensayando y empieza el ensayo.

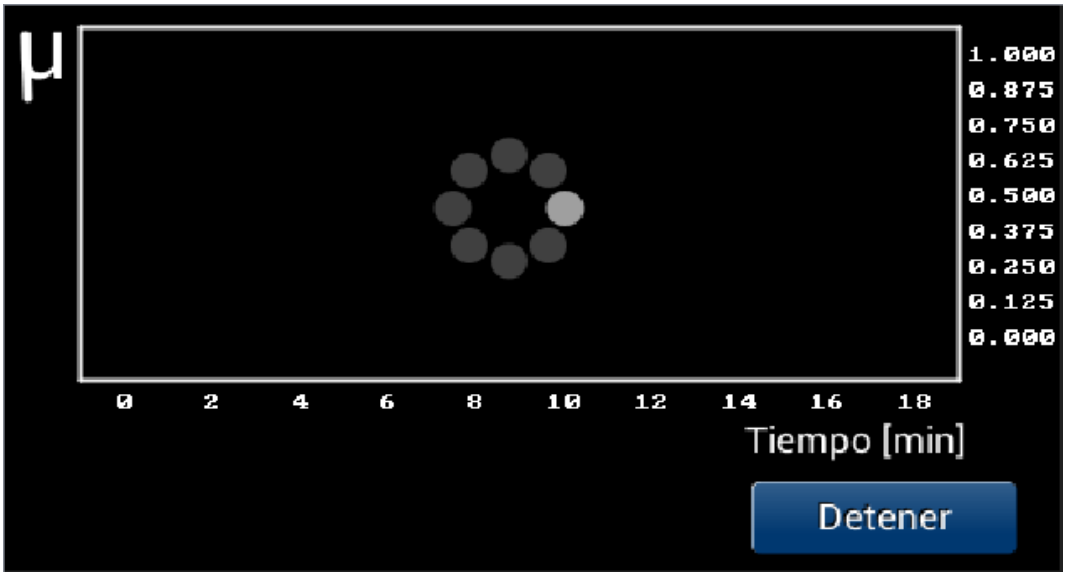


Figura 9.13: Pantalla de espera hasta que se coloque la carga.

9.7 Estado *Ensayando*

En este estado se desarrolla el ensayo, la probeta ya está friccionado con la rueda y se monitorea la distancia recorrida, se controla la velocidad del motor, se adquieren valores del esfuerzo en la celda, se almacenan datos en la SD y se grafica μ (mu) en función del tiempo en la pantalla, como se observa en la figura 9.14.

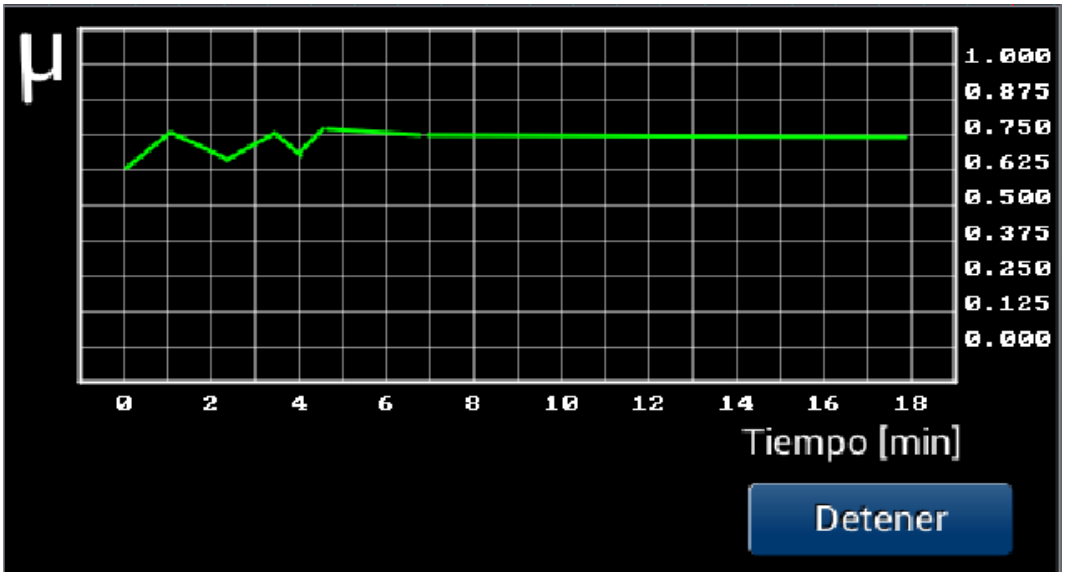


Figura 9.14: Pantalla Ensayando.

Control de la velocidad del motor:

El variador de velocidad ofrece distintos tipos de entrada para proporcionar la señal de referencia, de las cuales se optó por la entrada analógica de tensión. Esta entrada permite valores desde a 0V hasta 10V, donde el nivel de tensión es proporcional a la velocidad de referencia. Para implementar la función entre la velocidad y la tensión en esta entrada, deben ajustarse cuatro parámetros en el convertidor de frecuencia, que definen los dos extremos de una rampa, resultando una curva como la que se muestra en la figura 9.15.

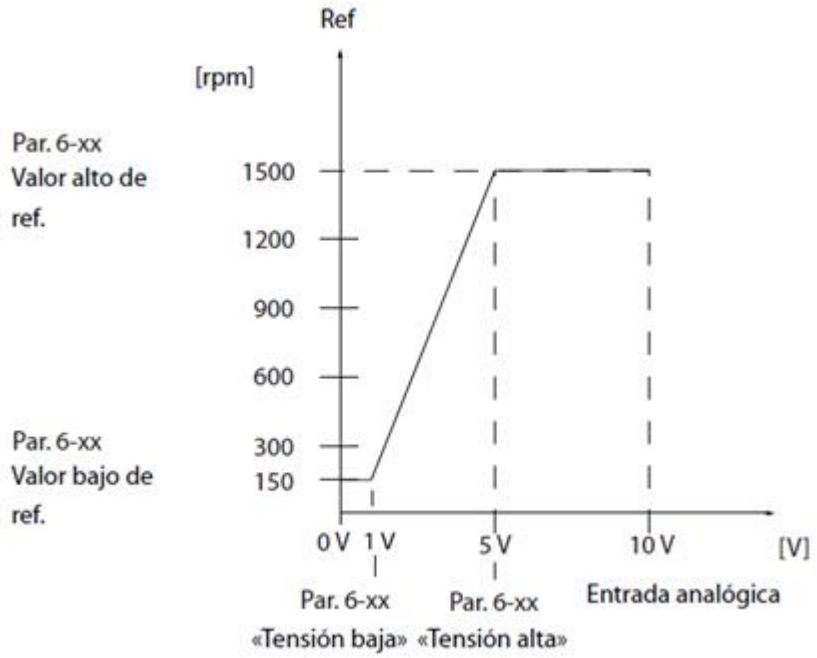


Figura 9.15: Velocidad de referencia vs Tensión.

La tensión para esta entrada es proporcionada por el DAC del microcontrolador, con un rango teórico de entre 0V y 3.3V. Sin embargo, la precisión de este periférico es muy pobre cerca de los extremos. Razón por la cual se ajustó un límite inferior de 0.2V y un límite superior en 3.2V, intervalo en el cual el DAC presenta una tensión de salida muy estable y precisa.

Teniendo en cuenta la velocidad nominal de 2840RPM del motor y la reducción de 7.5 presente entre el eje del motor y la rueda, se estableció una velocidad máxima de 350RPM en la rueda, que se traducen en 2625RPM en el motor. Mientras que el límite inferior se ajustó a 50RPM en la rueda, 375RPM en el eje del motor.

La tensión en la salida del DAC se ajusta modificando un registro de 12 bits. En el código se implementó una función que cumpla con la rampa generada en el variador de velocidad a partir de dos puntos. Para 3.2V en la salida el registro debe ajustarse a 3971, generando una velocidad de referencia en el motor de 2625RPM. Mientras que para 0.2V se coloca un valor de 248 en el registro, obteniéndose una velocidad de 375RPM en el eje. La función resultante es la mostrada en la ecuación 9.1:

$$Registro = 12.41 * f_{RUEDA} - 372.5 \quad (9.1)$$

Por otro lado, además de la información del recorrido tangencial de la rueda, también se puede obtener un control sobre la velocidad de la misma mediante la señal del sensor inductivo. La salida del sensor es ingresada a un canal de Input Capture del microcontrolador, que se configuró de forma tal que se produzca una interrupción en el código cada vez que aparece un flanco ascendente en el canal. Se guardan los valores del contador del timer en el momento de la interrupción. Con la diferencia entre valores sucesivos y usando como referencia la frecuencia de clock del timer, se obtiene la frecuencia de la señal de salida del sensor, como se expresa en la ecuación 9.2:

$$f_{SENSOR} = \frac{f_{CLOCK}}{(Contador_t - Contador_{t-1})} \qquad (9.2)$$

Considerando la relación de ocho que existe entre la frecuencia del sensor y la velocidad de la rueda, debido a la cantidad de dientes que presenta el sensor, se puede saber si la velocidad de la rueda es efectivamente la esperada. Se creó una función en el código que aumente el valor en el registro del DAC cuando la velocidad sensada es inferior a la pretendida y que disminuya en caso contrario. Este proceso continuará hasta que se llegue al valor con el cual se obtenga la menor diferencia entre la frecuencia de referencia y la sensada.

Capturas del esfuerzo en la celda de carga:

El dato de interés para el usuario es el coeficiente de rozamiento. Este valor es el cociente entre la fuerza normal, dato entregado por el usuario, y la fuerza de roce, que es medida mediante la celda de carga. Para obtener la fuerza de roce se aplica la ecuación 9.3:

$$F_R = {(V_{ADC} - V_{OFFSET})}/{G} \qquad (9.3)$$

Donde VADC es la tensión registrada por el ADC, VOFFSET es el nivel de tensión sin carga tomado antes del inicio del ensayo y G es la ganancia del amplificador y la celda de carga. Para determinar el valor de la ganancia se registró la tensión en el ADC para distintas pesas colocadas en la celda de carga. Las pesas fueron medidas previamente con una balanza de un gramo de precisión. Con estos datos se ingresó a la herramienta Curve fitting de MATLAB y se obtuvo la curva mostrada en la figura 9.16. En el eje x aparece el peso en gramos y en el eje y2 la tensión en mV. La curva muestra un comportamiento lineal con una pendiente de 0.1092V/Kg, valor que es la ganancia del sistema.

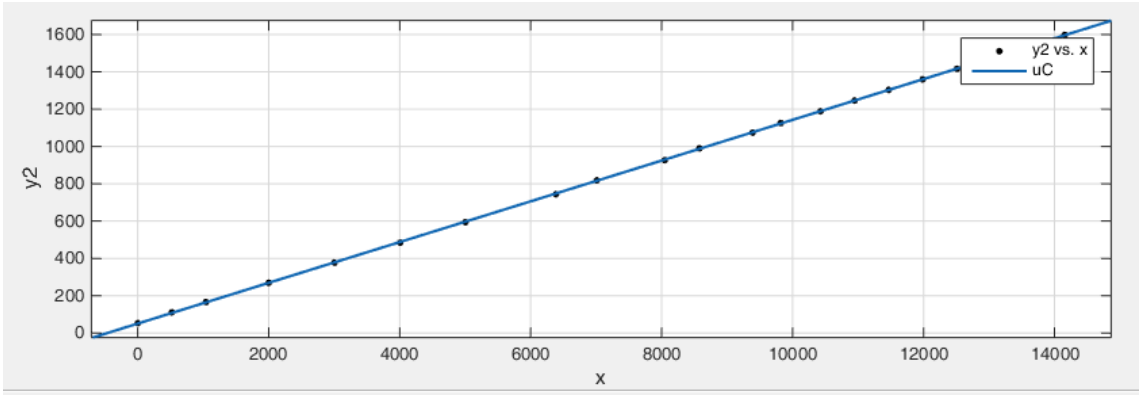


Figura 9.16: Curva Peso vs Tensión.

Sin embargo, para mitigar el ruido presente en las mediciones del ADC se implementó un filtro digital. Haciendo uso del programa MATLAB, se partió de un filtro de cuarto orden en tiempo continuo con una constante de tiempo tal que al cabo de un segundo se obtenga un valor estable. Con la función c2d se pasó de tiempo continuo a discreto con una frecuencia de muestreo de 100Hz y el método Tustin, que utiliza la transformación bilineal para obtener el mejor mapeo en el dominio de la frecuencia entre sistemas continuos y discretos. El filtro resultante se muestra en la ecuación 9.4:

$$y[n] = 2.234 * 10^{-5} * x[n] + 8.937 * 10^{-5} * x[n-1] + 1.341 * 10^{-4} x[n-2] + 8.937 * 10^{-5} x[n-3] + 2.234 * 10^{-5} x[n-4] + 3.45 * y[n-1] - 4.463 * y[n-2] + 2.566 * y[n-3] - 0.5534 * y[n-4] \quad (9.4)$$

La respuesta al escalón de este filtro puede observarse en la figura 9.17.

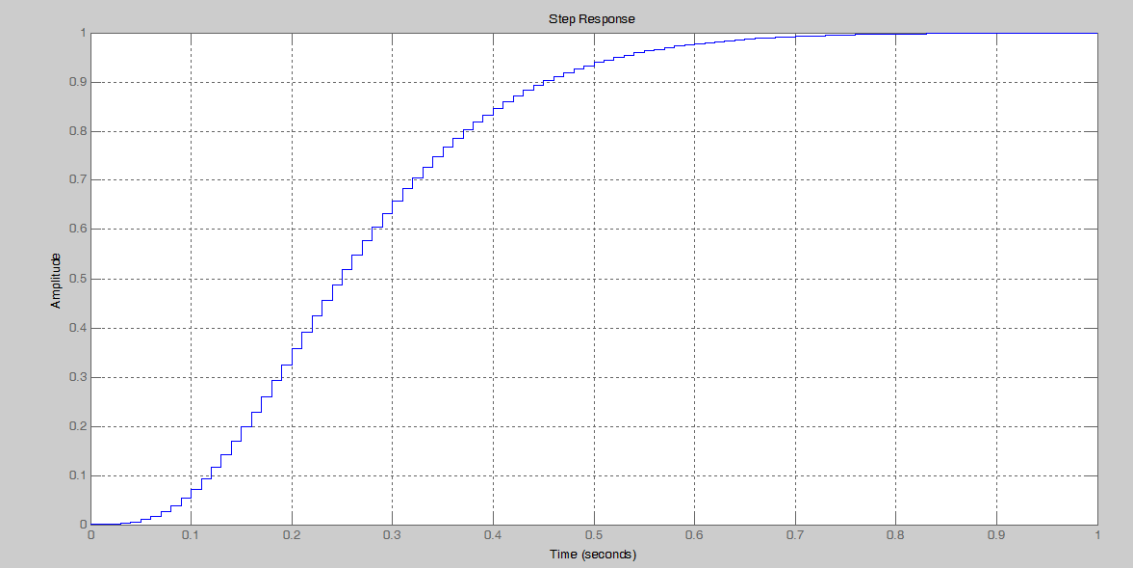


Figura 9.17: Respuesta al escalón del filtro digital.

Se programó un timer que interrumpa cada 10ms, el tiempo de muestreo. En la rutina de interrupción se toma el valor del registro del ADC y se le aplica el filtro digital de la ecuación 9.4. Al cabo de 100 interrupciones se toma la salida del filtro y se vuelven todas las variables a cero. De esta manera, cada un segundo se obtienen valores del registro del ADC con ruido casi nulo. La tensión en el ADC filtrada está dada por la ecuación 9.5:

$$V_{ADC} = \frac{Registro_{ADC}}{4095} * 3.3V \quad (9.5)$$

Donde 3.3V es la tensión de referencia y 4095 es por los 12 bits del ADC. Con este valor se ingresa en la ecuación 9.3 y finalmente se calcula μ para enviarlo a la tarjeta SD.

Datos almacenados en la SD:

Apenas se ingresa a este estado, si el usuario optó por guardar los datos, se crea el archivo .CSV con el nombre elegido. Se almacena el valor de los parámetros y se coloca el nombre de los datos que se almacenarán, como muestra la figura 9.18.

Velocidad de giro (RPM)	Duracion (min)	Distancia (m)	Diametro (mm)	Fuerza (N)
200	30	4309	288,2	130
tiempo(seg)	distancia(m)	u(mu)		

Figura 9.18: Datos almacenados en la tarjeta SD.

Luego, cuando transcurre el ensayo, se guarda el valor de µ (mu) con el tiempo y la distancia recorrida cuando se tomó este valor. Estos datos se guardan con una frecuencia de 1 segundo.

Monitoreo de la distancia recorrida:

Para que la rueda ejecute la distancia establecida se utiliza el sensor inductivo. Éste sensa los engranajes de un disco dentado colocado en el eje que une el motor con la rueda de goma. Éste disco posee 8 dientes, entonces por cada vuelta que realiza la rueda el sensor inductivo produce 8 pulsos.

Con la distancia y el diámetro de la rueda ya fijados, el programa calcula a cuantos pulsos del sensor inductivo equivale la distancia del ensayo como muestra la ecuación 9.6.

$$NumerodePulsos = \frac{8 * distancia}{\pi * diametro} \qquad (9.6)$$

De esta manera, cada vez que se produce un pulso el programa interrumpe y le resta un valor a NumerodePulsos. Cuando esta variable llega a 0, el ensayo termina y aparece la pantalla de la figura 9.19.

Figura 9.19: Pantalla de fin del ensayo.

Cuando se presione el botón OK se retorna al estado Inicio.

9.8 Estado *StandBy*

Además de las tareas mencionadas se debe verificar que la carga permanezca en la posición correcta todo el ensayo. Para ello se monitorea que la palanca que coloca la carga se encuentre en posición, la salida del optoacoplador ranurado debe permanecer en alto. Si esto no ocurre se va al estado denominado StandBy que produce la pantalla de la figura 9.20.

Figura 9.20: Pantalla estado *Standby.*

En ese momento se paran las interrupciones, se deja de contar pulsos, pero la rueda continua girando a la frecuencia establecida. Como muestra la figura 9.20, el usuario puede elegir por continuar el ensayo colocando nuevamente la probeta o dar por finalizado el ensayo presionando el botón Salir.

9.9 Estado *Rectificacion*

En este estado se puede realizar la rectificación de la rueda de goma. El operario tiene la opción de escoger la velocidad de giro de la rueda y puede encender y parar el motor cuando desee. La figura 9.21 corresponde a la pantalla que aparece apenas se ingresa a este estado y cuando el motor está parado. La figura 9.22 corresponde a cuando la rueda está girando.

Por comodidad y diseño de la máquina, para realizar la rectificación, la rueda gira en el sentido contrario al que lo realiza en el ensayo.

Si se ingresa un valor no adecuado aparece una pantalla de aviso, como muestra la figura 9.23.

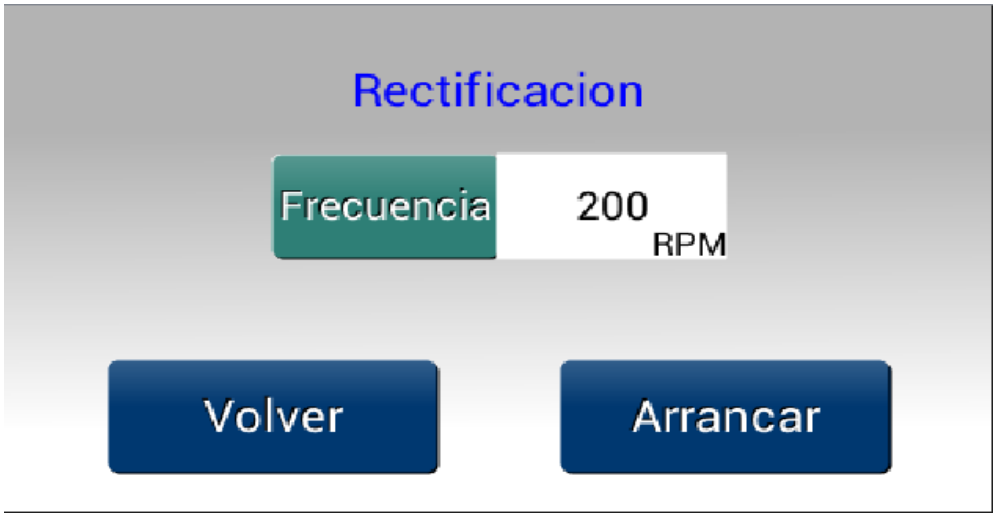


Figura 9.21: Pantalla rectificación, motor detenido.

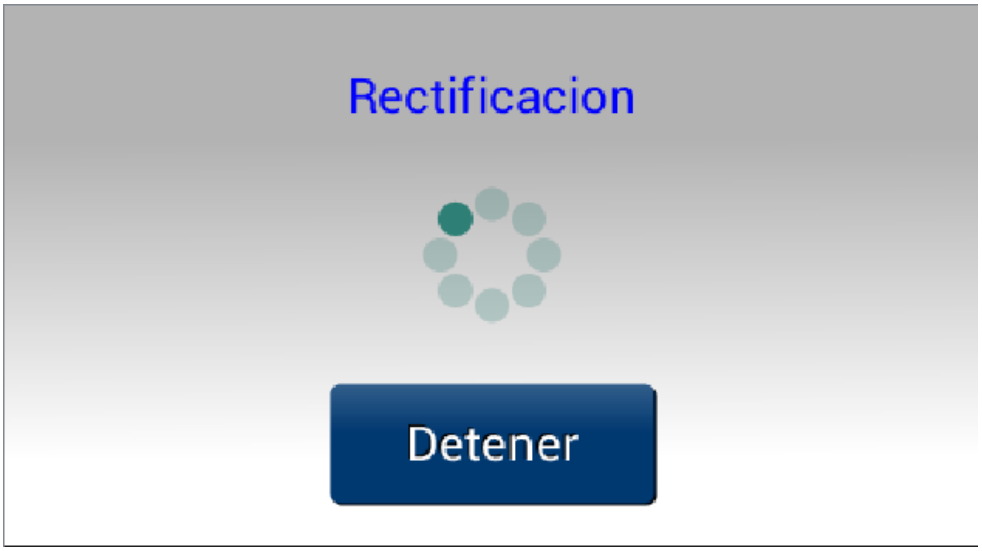


Figura 9.22: Pantalla rectificación, motor encendido.

Figura 9.23: Pantalla para velocidad de rectificación errónea.

9.10 Diagrama de flujo

El programa se puede sintetizar en el diagrama de flujo de la figura 9.24.

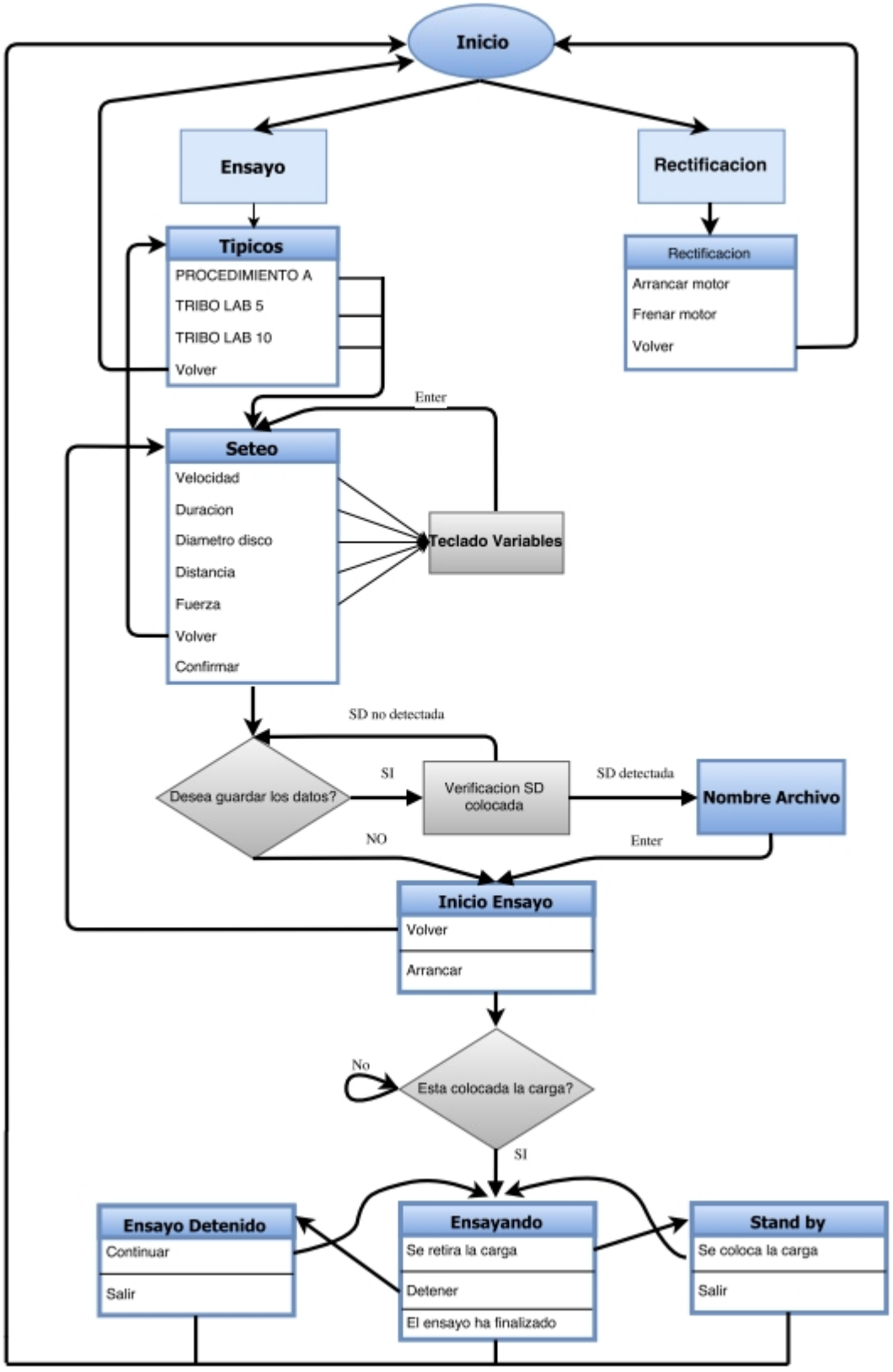


Figura: 9.24.

10
PCB e Instalación

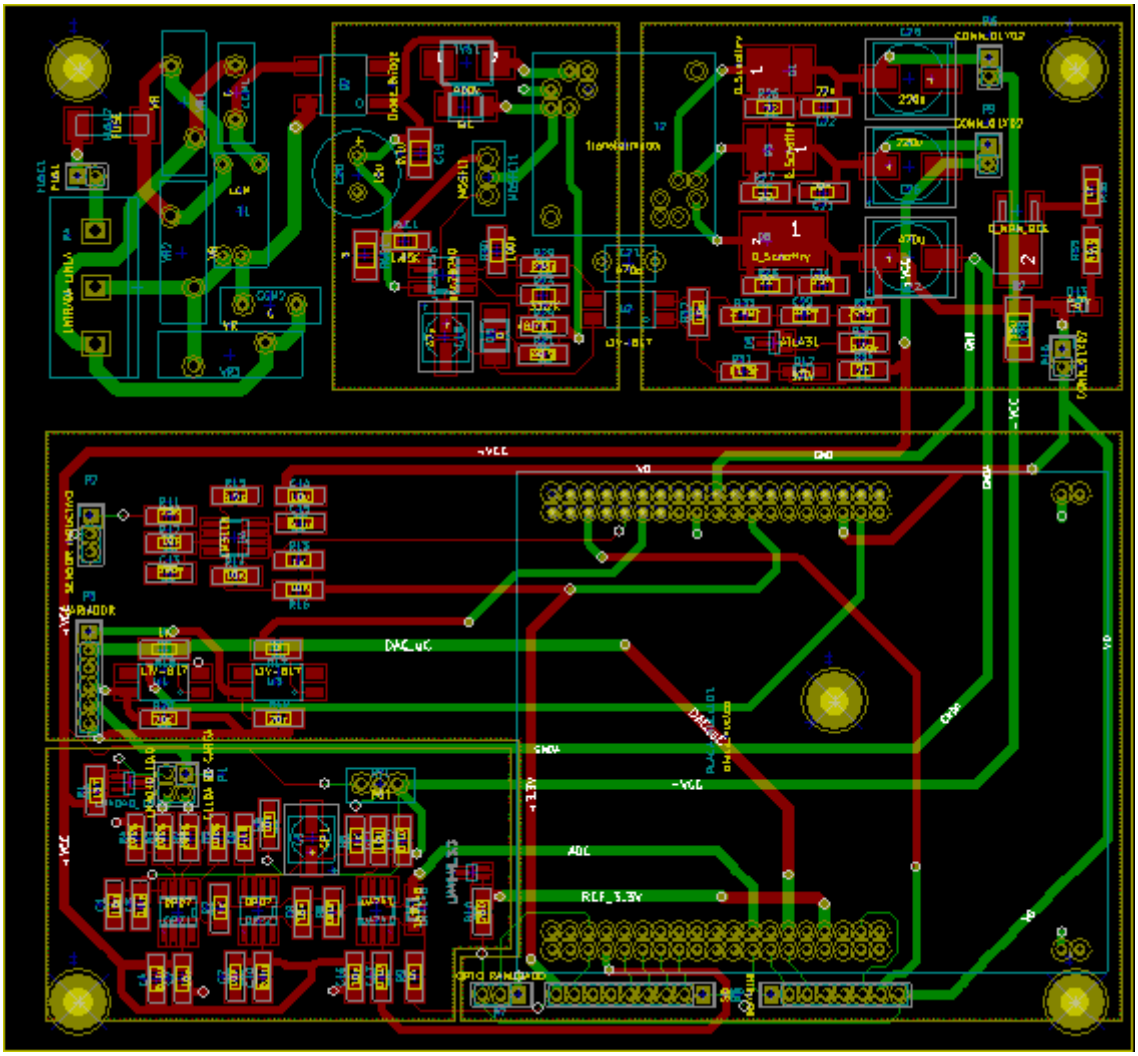
10.1 PCB

Se realizó un circuito impreso de dos capas aunque, solo se colocaron componentes en la capa superior. La utilización de las dos caras del material optimiza su utilización y facilita el ruteo.

Se trató de utilizar mayormente componentes de montaje superficial (SMD), pues presentan menor tamaño y precio. Se soldaron de manera manual, por esta razón se eligieron partes con tamaño unificado a 1206.

El layout de la placa se puede observar en la figura 10.1. A continuación comentaremos algunas de las cuestiones tenidas en cuenta en el diseño:

- **Creepage:**
  Es la distancia que separa a dos conductores en la placa. Cuando se trabaja con tensiones superiores a 30VAC se debe tener en cuenta por la distancia entre trazos sobre la superficie del PCB para evitar un arco. Por esta razón en la parte de entrada de la fuente, hasta llegar al puente de diodos, se lo contempló en el diseño.
  Para ello se utilizó el estándar IPC2221A (Generic Standard on Printed Board Design) y se diseñó para un creepage de 3mm. Entonces, se trabajó con una separación mínima entre los pads y pistas entre Fase y Neutro de 3mm.



Pistas capa inferior. Pistas capa superior.

Figura 10.1: Pistas de ambas capas.

- **Especificaciones del controlador de la fuente conmutada:**
  Principalmente disminuir el área del lazo de realimentación y colocar la resistencia de sensado de corriente lo más cerca del capacitor de entrada, de manera de minimizar el área de la espira radiativa que contiene el capacitor de entrada, el transformador, el MOSFET y la resistencia de sensado de corriente.

- **Planos de masa:**
  Se crearon cuatro, como se puede ver en la figura 10.2. El que posee el numero 1 corresponde al lado del primario la fuente que debe estar aislado del resto del circuito. El numero 2 pertenece al circuito secundario de la fuente, el numero 3 a la parte analógica de la placa y el numero 4 a la digital. Estos últimos solo se unen en el capacitor de salida de la fuente, en la figura 10.2 denominado *punto de unión.*
  Esta división y unión en un solo punto se realizó para evitar el acoplamiento de ruido eléctrico producido por los transitorios de corriente de la parte digital circulando por trazos con impedancia significativa.

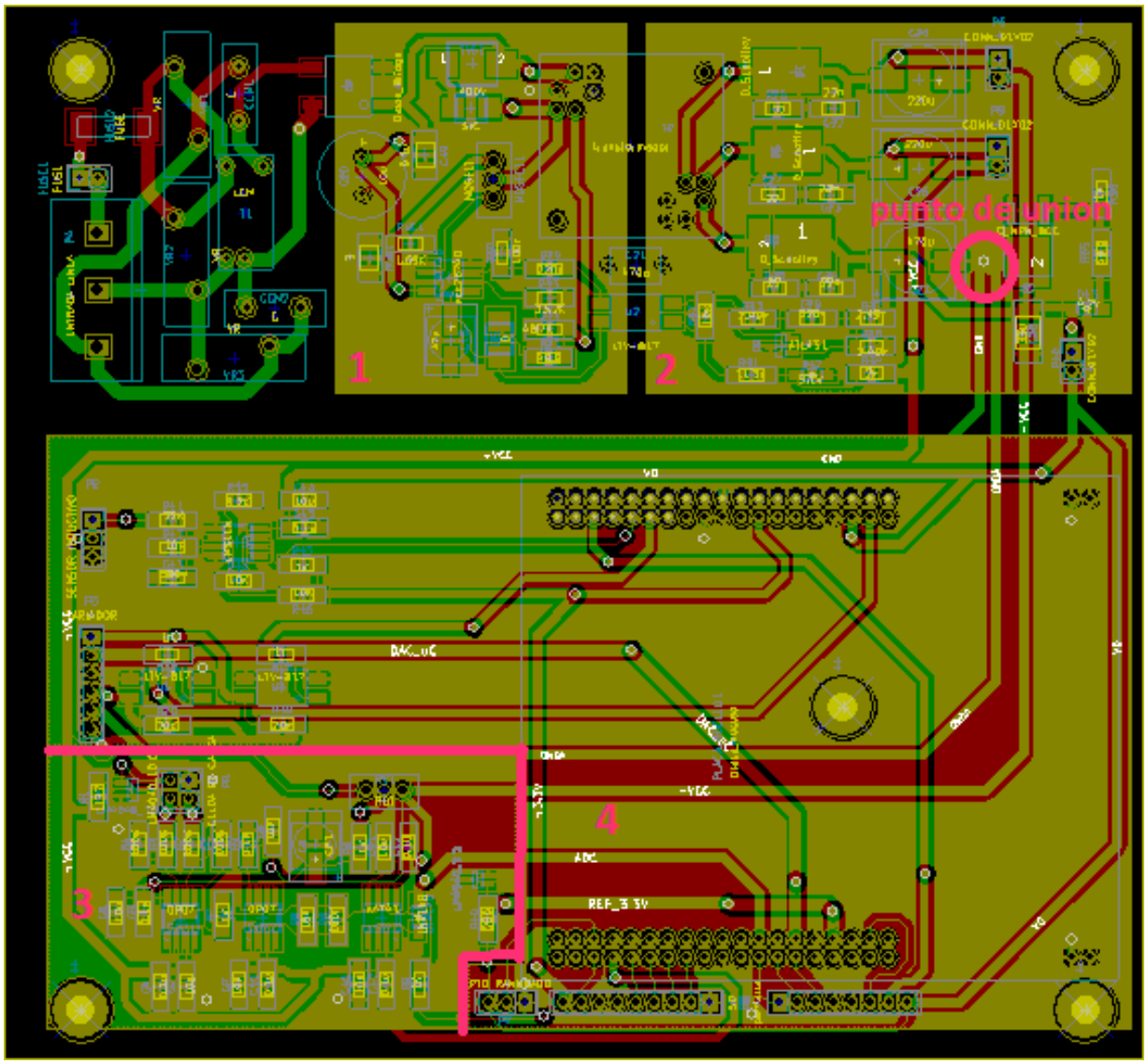


Figura 10.2: Planos de masa.

Para realizar el diseño se utilizó el programa KiCad. Este presenta la ventaja de ser gratuito y poseer una interfaz amigable con el usuario. No presenta gran cantidad de librerías con huellas de componentes, pero se pueden crear fácilmente.

Además se realizó un circuito impreso que posee un zócalo para albergar la tarjeta SD y un conector para la comunicación SPI. El layout se puede observar en la figura 10.3.

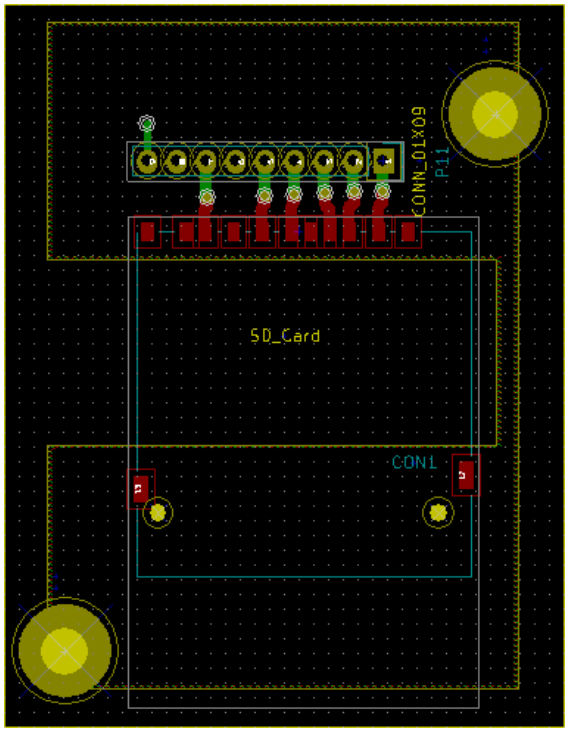


Figura 10.3: Layout de la placa que contiene el zócalo de la tarjeta SD.

10.2 Instalación

En esta sección se hará una breve descripción acerca de la instalación, mostrando imágenes de la misma y comentando algunos detalles tenidos en cuenta.

Empezamos con el gabinete plástico montado en la parte trasera de la máquina, ya instalado antes de comenzar con el proyecto, mostrado en la figura 10.4. Dentro del gabinete se encuentran la placa que controla el proceso, la fuente de alimentación utilizada para alimentar las dos tiras de led presentes en el interior de la máquina y un riel din con una llave termomagnética para interrumpir una fase y borneras de paso.

El riel din esta levantado aproximadamente 10cm para que la llave termomagnética pueda ser accedida por el usuario en caso de falla. Se ingresa con la tensión de línea y se distribuye hacia el variador de velocidad, la fuente externa para los leds y la placa.

La placa cuenta con conectores para la tensión de línea, el portafusible, sensor inductivo, variador de velocidad, celda de carga, optoacoplador ranurado, la placa con el zócalo de la tarjeta SD y la pantalla táctil. Para todas estas conexiones se utilizó cable mallado, a excepción del portafusible y el sensor inductivo

En las figuras 10.5, 10.6 y 10.7 se puede apreciar la máquina vista desde distintas perspectivas. El cable azul es el de la celda de carga. Se encuentra enrollado debido a que el cable forma parte de las compensaciones por temperatura incluidas por el fabricante en el puente de Wheatstone.

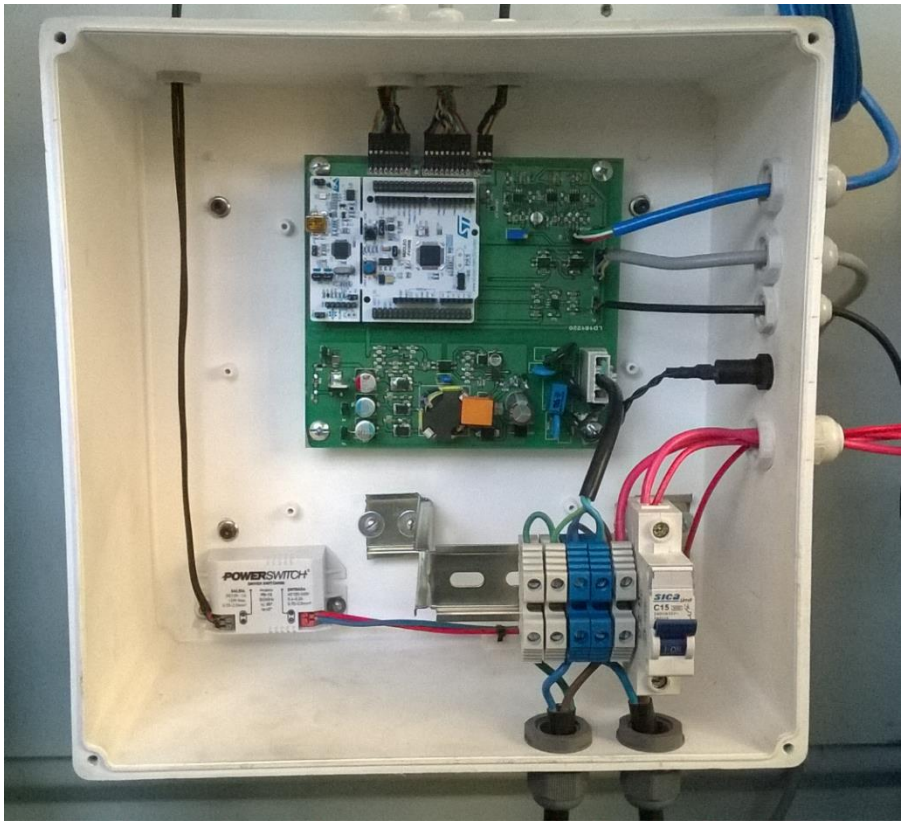


Figura 10.4: Gabinete plástico.

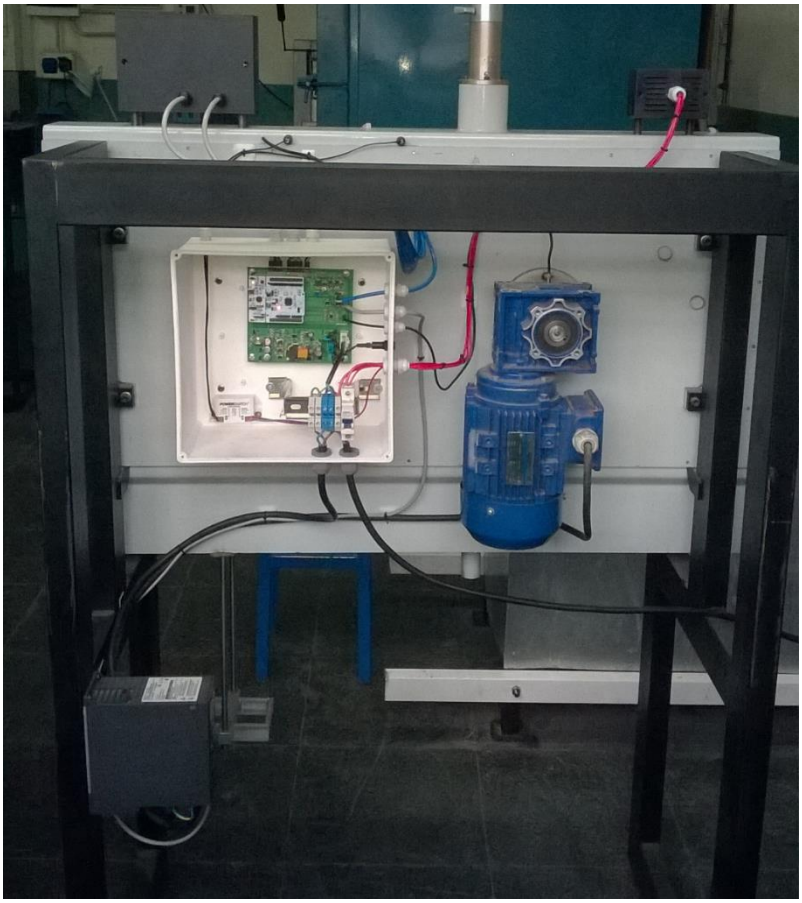


Figura 10.5: Máquina vista de atrás.

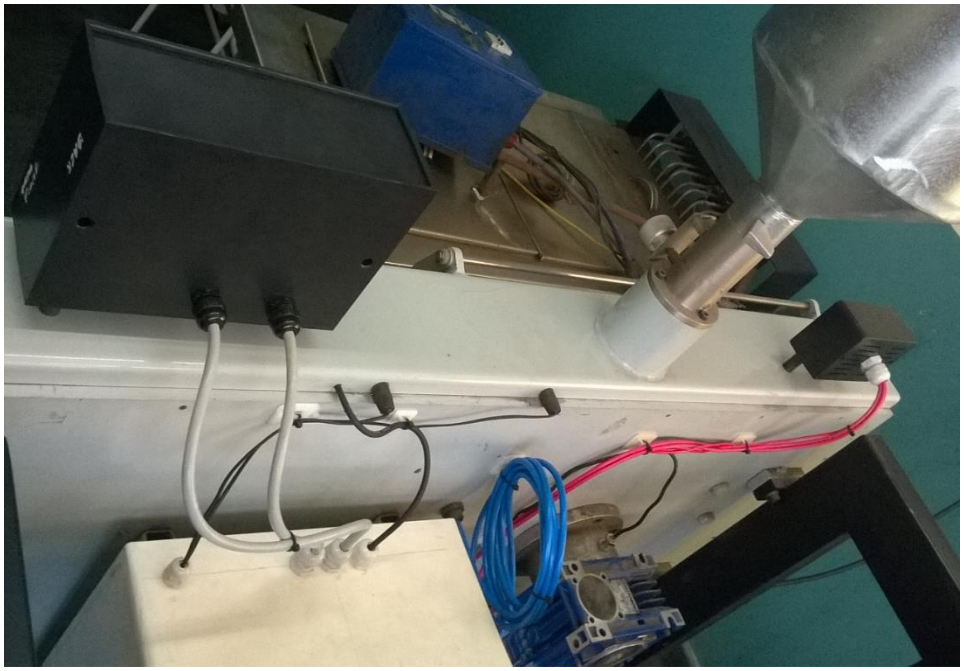


Figura 10.6: Máquina vista de arriba.

Figura 10.7: Máquina vista de frente.

En la parte superior aparecen dos gabinetes plásticos de color negro. El más pequeño tiene dos interruptores independientes, uno para el habilitar la alimentación del sistema y otro para el encendido y apagado de las tiras de led. Mientras que en el otro gabinete están la pantalla táctil y la ranura para colocar la tarjeta SD.

Por último, en las figuras 10.8 y 10.9 se muestra la colocación de la celda de carga y optoacoplador ranurado respectivamente. Para el optoacoplador se armó un sistema tal que el camino óptico se interrumpa cuando se esté aplicando la carga.

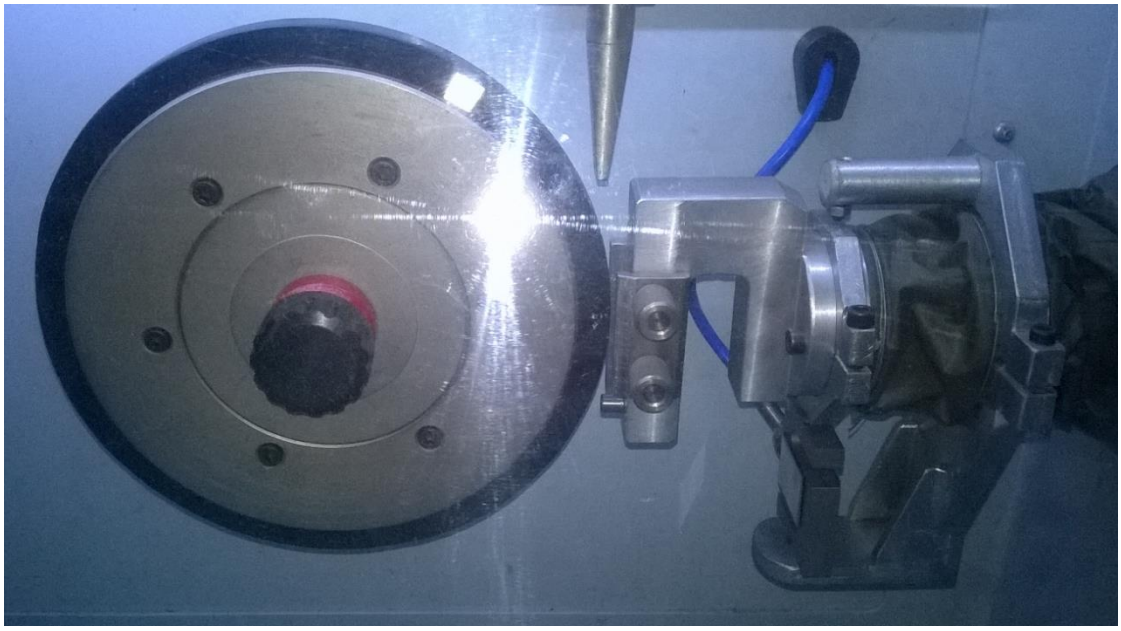


Figura 10.8: Instalación de la celda de carga.

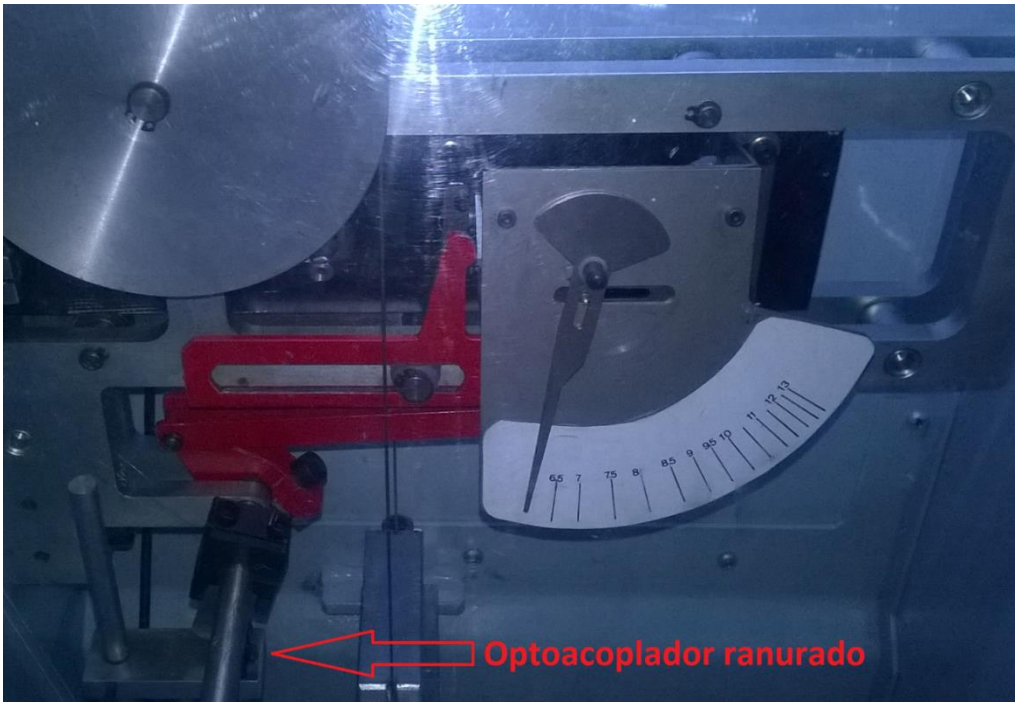


Figura 10.9: Instalación del optoacoplador ranurado.

11

Resultados

11.1 Fuente de alimentación

A continuación se mostrarán las formas de onda obtenidas en la fuente de alimentación. La primera aparece en el figura 11.1, donde se muestran los tres pulsos iniciales que envía el controlador para comprobar cualquier situación de falla.

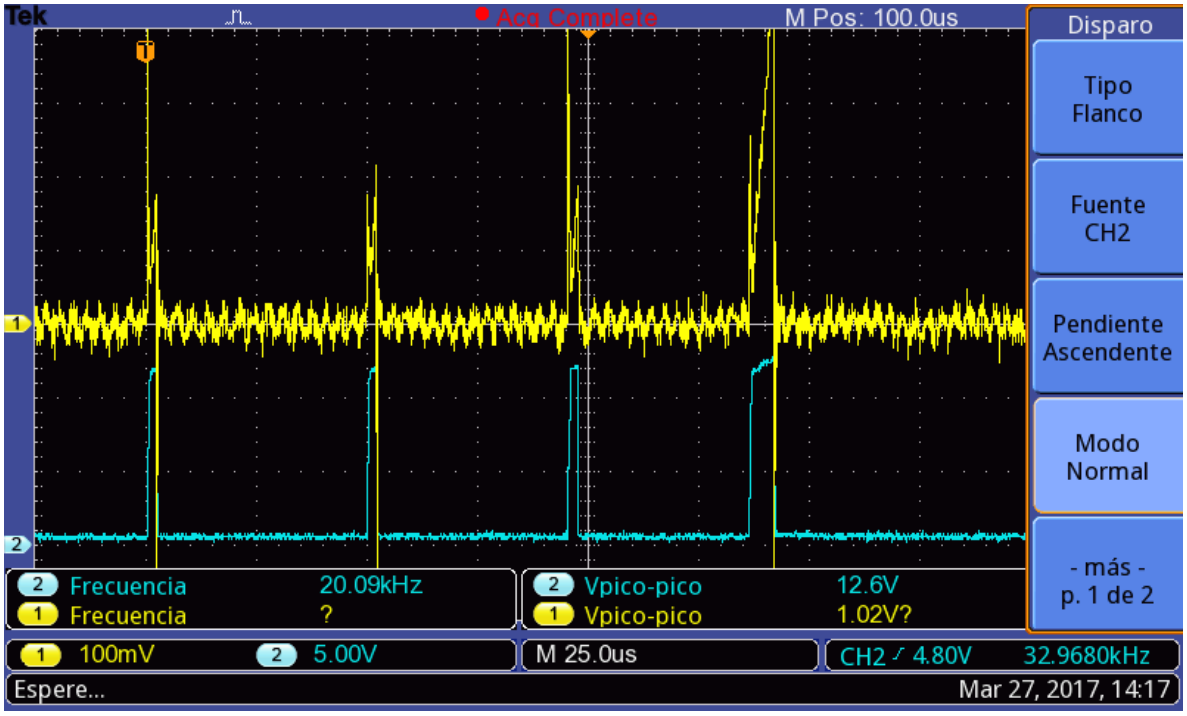


Figura 11.1: Pulsos iniciales.

La característica de estos pulsos, de celeste en la imagen, coincide con lo especificado por el fabricante. Tienen una amplitud de cerca de 13V y produce una corriente pico de la cuarta parte de la máxima, como se verifica con la forma de onda de la tensión en la resistencia de sensado de amarillo.

Luego de estos tres pulsos, el controlador ajusta la corriente pico al 63% del máximo y modula en frecuencia para cargar el capacitor de salida con corriente promedio constante. En la figura 11.2 aparecen más en detalle el último de los pulsos iniciales y el primero con las características mencionadas.

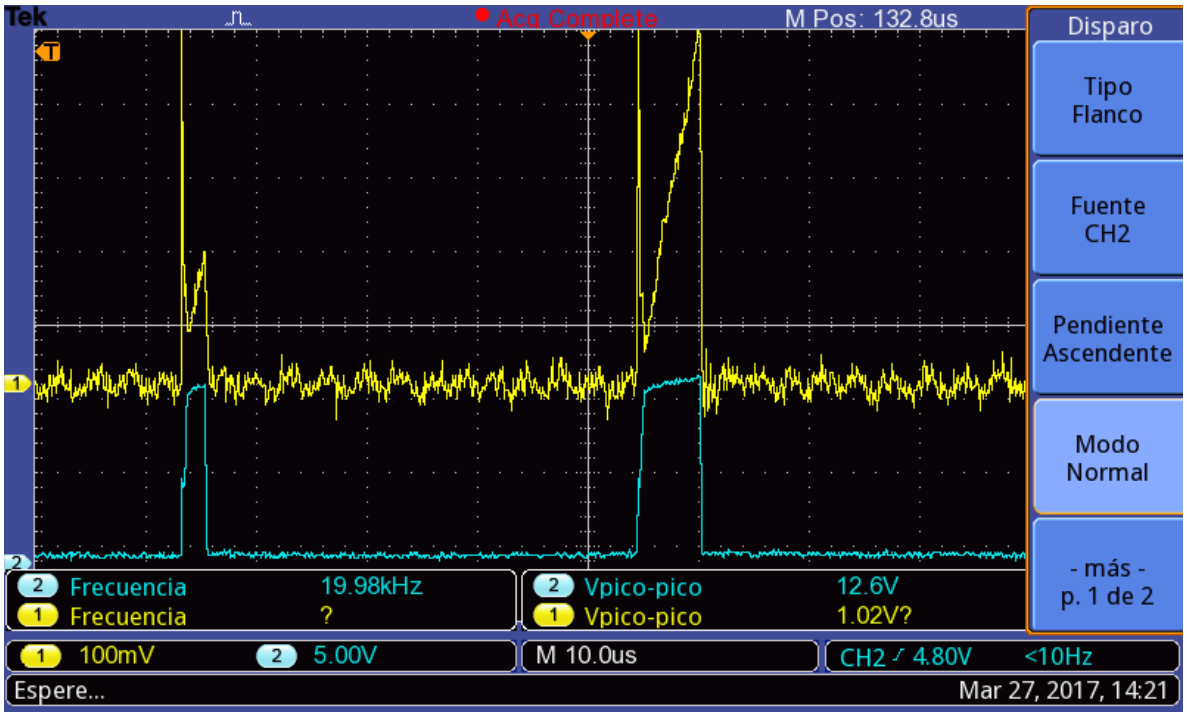


Figura 11.2: Pulsos iniciales en detalle.

La corriente máxima se corresponde con 0.773V en el pin CS, la cuarta parte con 0.19325V y el 63% con 0.48699V. Estas relaciones pueden corroborarse en la amplitud pico de la señal de amarillo.

La señal de gate del MOSFET y la tensión en la resistencia de sensado se muestran por última vez en la figura 11.3, pero esta vez en con la tensión de salida en 5V. La corriente pico en el primario esta al máximo y la frecuencia de los pulsos es superior a 32KHZ. Esto nos dice que el controlador está operando en la zona que denominamos FM3, donde se modula la frecuencia desde 32KHZ hasta 100KHZ. En el momento de tomar estas imágenes el circuito se encontraba con poca carga, razón por la cual la fuente estaba entregando solo una fracción de su potencia máxima.

Seguimos con la tensión de otro de los pines del controlador, VS, cuya forma de onda aparece en amarillo en la figura 11.4.

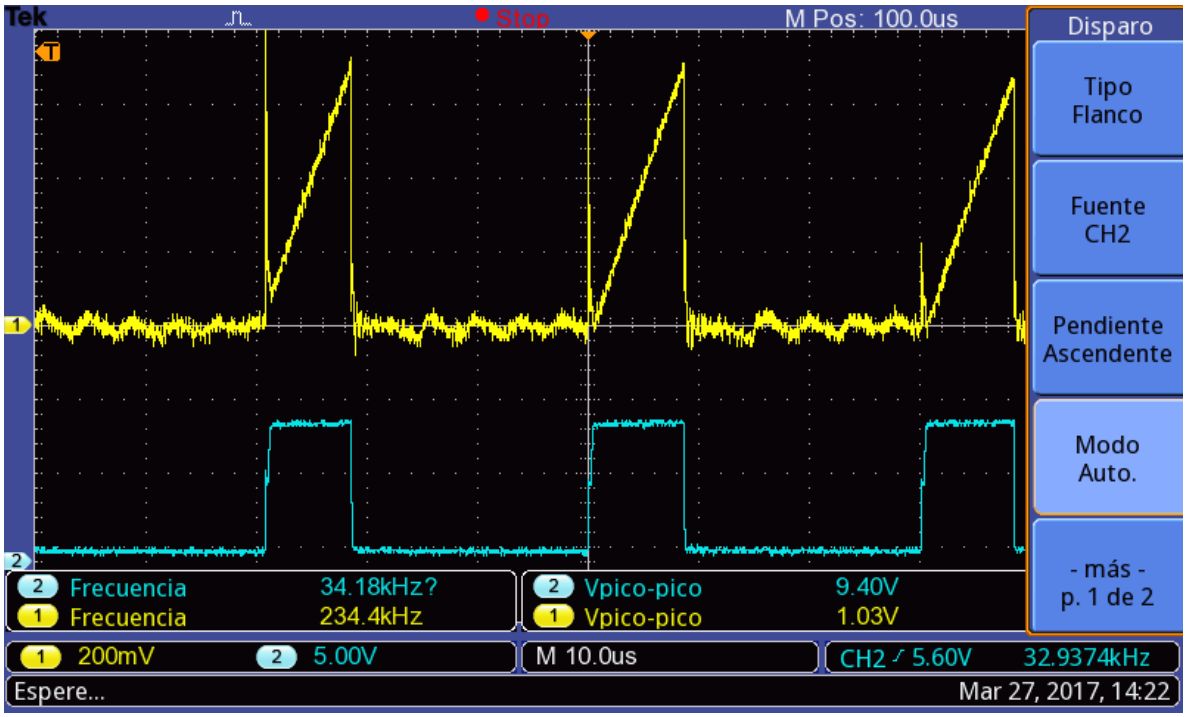


Figura 11.3: Tensión en los pines DRV y CS con la salida ya establecida.

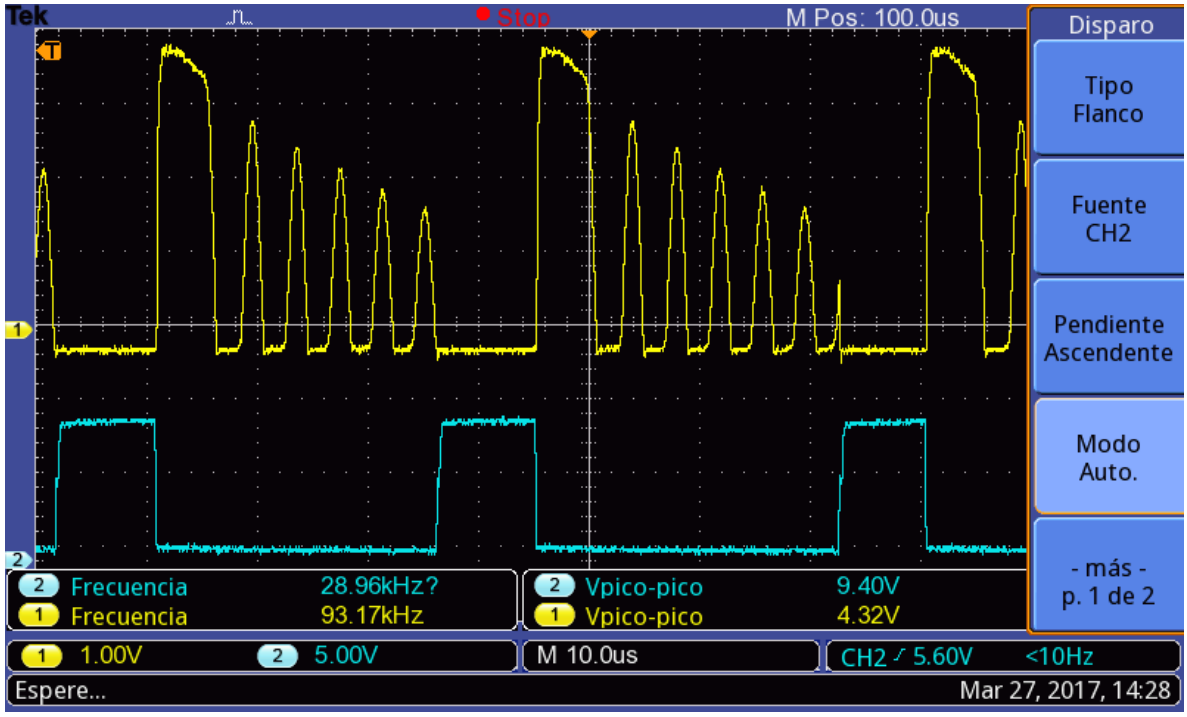


Figura 11.4: Tensión en los pines VS y DRV.

La tensión en VS es ligeramente negativa en el tiempo ON, momento en el que mediante la corriente en RS1 se sensa la tensión de entrada. Durante el tiempo de OFF, esta tensión es muestrea la salida.

Por último, en la figura 11.5 se grafica en amarillo a la tensión en el pin FB y en celeste la tensión de salida. A medida que la tensión en FB es menor, mayor potencia entrega la fuente. Por este motivo cuando cae la amplitud de la señal en amarillo, más rápido es el aumento en la tensión de salida.

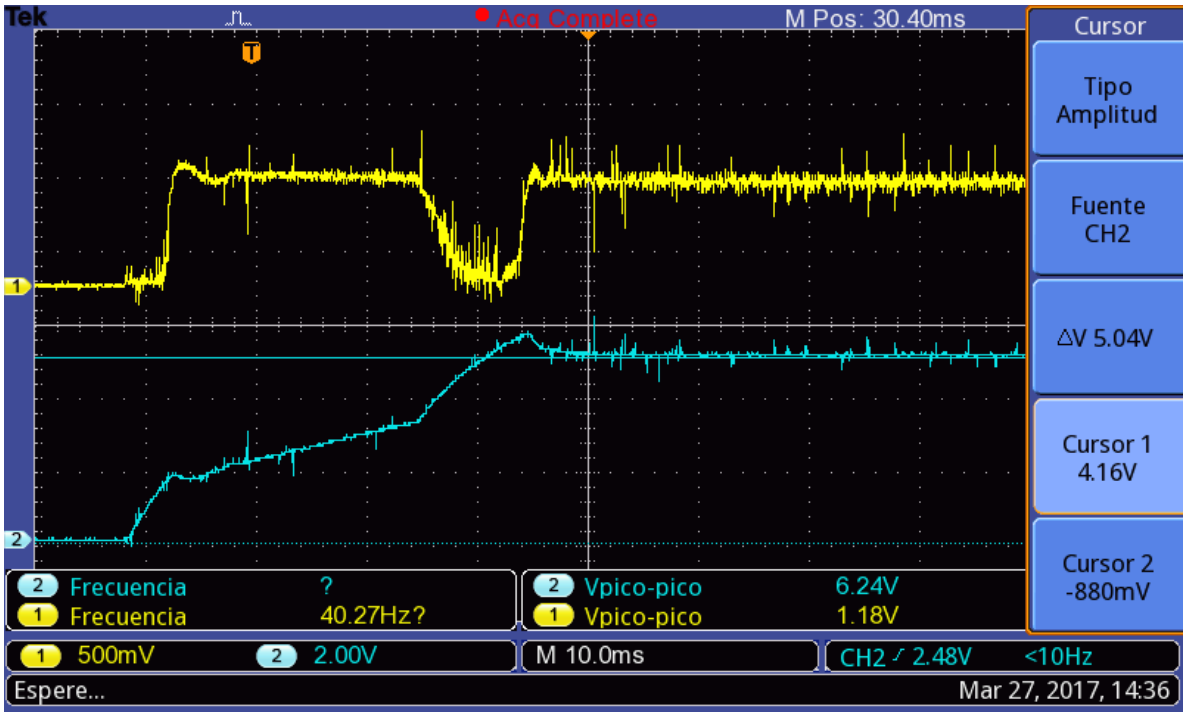


Figura 11.5: Transitorio en la tensión de salida y en el pin FB.

Todas las formas de onda vistas son consistentes con el desarrollo realizado en la etapa de diseño y lo especificado por el fabricante del controlador Flyback.

Durante la prueba de la fuente de alimentación se pudieron comprobar otras características del UCC28740. En los primeros ensayos con el controlador en funcionamiento, se utilizó una fuente de continua externa para proveer la tensión de entrada. Cuando la amplitud de la fuente externa estaba por debajo del nivel establecido en el diseño para la tensión de entrada, el controlador solo enviaba los tres pulsos iniciales de manera repetida.

Por otro lado, se observó como el controlador se apagaba y encendía de forma cíclica cuando no había señal de realimentación. La tensión de salida llegaba a su valor máximo impuesto por diseño, momento en el cual el controlador se apagaba. Luego de que la tensión en el pin VDD cayera por debajo de 7.75V, el controlador se volvía a encender y un tiempo después la tensión de salida legaba nuevamente al máximo. Este procesos de repetía de forma continua hasta que aparecía la señal de realimentación.

Finalmente, se colocó una carga externa en la salida –VCC, debido a que la tensión en esta salida excedía en modulo los -15V esperados. Esto ocurría debido a la poca carga presenta en esta salida, en contraste con la salida de principal, la de mayor carga.

11.2 Ensayos

Para terminar haremos un breve análisis de los datos adquiridos durante un ensayo típico. Los parámetros del ensayo fueron los siguientes:

- Velocidad de la rueda: 200RPM
- Duración: 5.35min
- Distancia: 718.2m
- Diámetro de la rueda: 227mm
- Fuerza Normal: 125N

Los datos para el análisis fueron tomados del archivo .CSV creado en la tarjeta SD, recordando que se registran valores cada un segundo. En la figura 11.6 se muestra el gráfico resultante del coeficiente de roce en función de la distancia tangencial recorrida por la rueda.

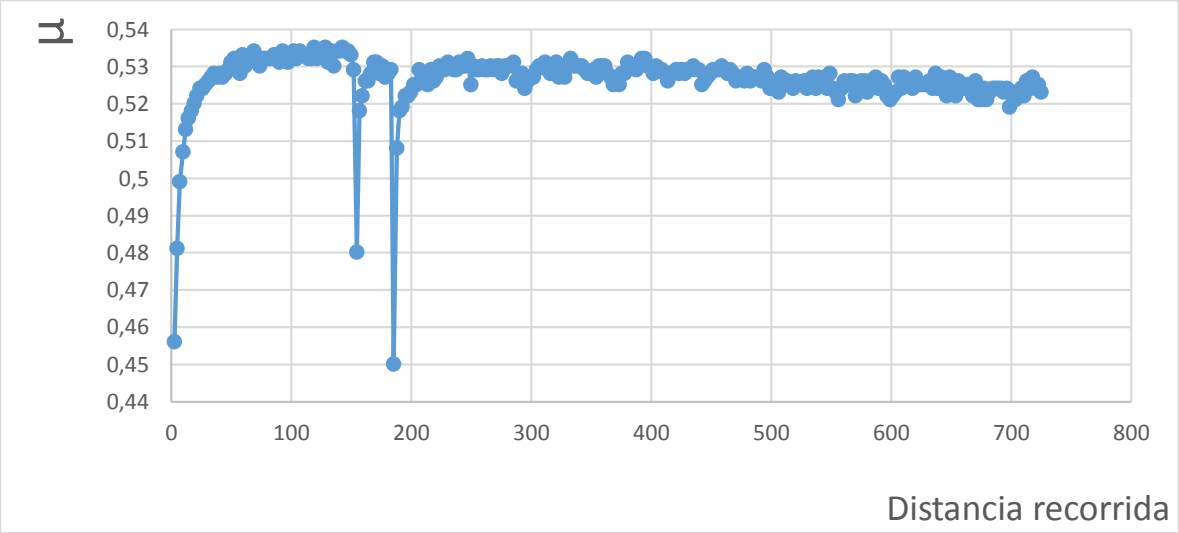


Figura 11.6: Gráfico de μ en función de la distancia recorrida.

En la curva se puede apreciar un pequeño tiempo de establecimiento al inicio, hasta que llega a un valor superior a 0.53 y luego permanece constante en dicho valor. Luego aparecen dos picos de gran amplitud hacia abajo. Estos picos aparecen como consecuencia de colocar y retirar la carga durante el ensayo. Realizando pruebas con la máquina en funcionamiento, se observó que el valor de μ es muy dependiente de la posición de la boquilla y el caudal de arena. Mirando la curva se puede determinar en qué momento hubo cambio en el caudal de arena, por ejemplo por obstrucción de la boquilla. En general, el coeficiente de roce tiende a un valor constante. Sin

embargo, se observa un pendiente ligeramente negativa sobre el final. Está pendiente tiene que ver con el desgaste del material, cuanto más gastada esta la probeta, mayor es la pendiente.

En cuanto las variaciones que se observan, son producto de la vibración en el sistema mecánico que acciona la palanca. La vibración genera cambios en la amplitud de μ ligeramente inferiores a 0.01. Este número representa en la medición de la fuerza de roce mediante la celda de carga un cambio de aproximadamente 125gr. Durante la calibración de las mediciones del esfuerzo se midieron 3003gr con 3gr de error, que representa un 0.099%. Es decir, que el error en la medición es despreciable frente a la vibración que presenta en el sistema y no debería tenerse en cuenta.

Finalizamos con las figuras 11.7 y 11.8, donde se muestran imágenes de la probeta antes y después del ensayo respectivamente, para apreciar la huella producto del desgaste que se produce sobre la misma.

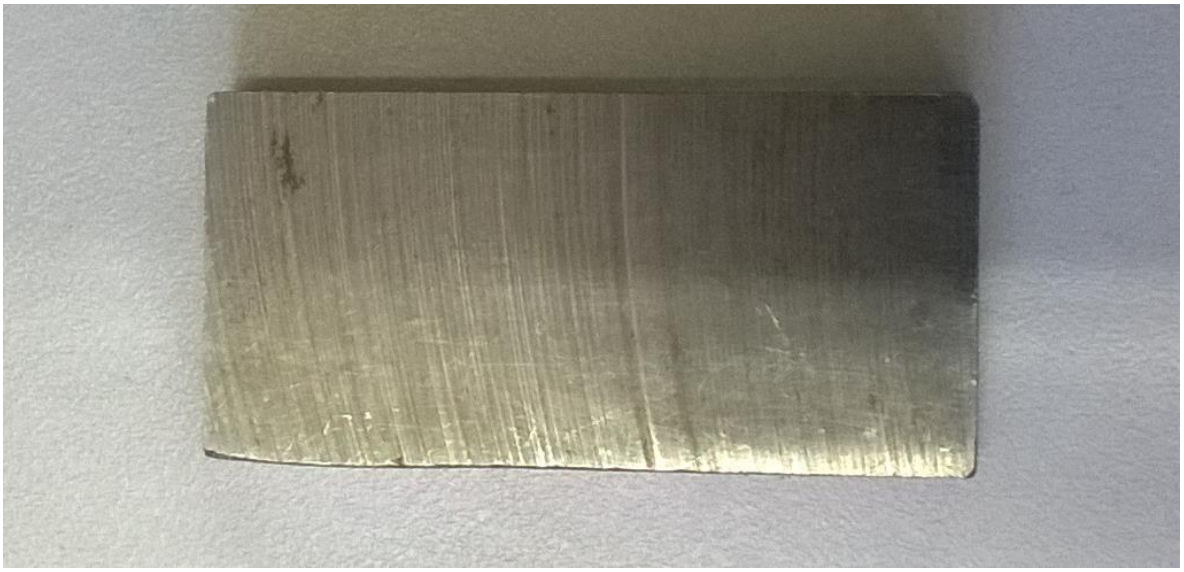


Figura 11.7: Probeta antes del ensayo.

Figura 11.8: Probeta después del ensayo.

11.3 Conclusión

Tras la finalización de la instalación, correcciones de pequeños detalles en el programa y las diversas pruebas realizadas en la máquina, podemos concluir que los resultados fueron positivos.

Como se mencionó anteriormente, la vibración presente en el sistema genera variaciones en la adquisición muy superiores al error en la medición del esfuerzo. Motivo por el cual el pequeño error en la calibración es despreciable.

Por otro lado, el programa que ajusta la velocidad en la rueda de goma mediante la información obtenida del sensor inductivo también funcionó de manera correcta. Esta rutina siempre corrige el valor cargado inicialmente en el registro de DAC. Los ensayos se miden por la distancia tangencial recorrida por la rueda, pero se hace un cálculo del tiempo que aproximadamente debería durar el mismo. Dicho calculo mostro ser muy preciso, ya que para ensayos de 30 minutos se observaron diferencias de entre uno y cinco segundos. Este hecho nos marca que la velocidad en la rueda es la esperada.

Mientras que la interfaz con el usuario, contiene todas las funcionalidades requeridas. Cuestiones como poder continuar un ensayo después de retirar la carga y continuarlo volviendo a colocarla, permite que el usuario pueda solucionar problemas como la posición de la boquilla de arena. Además de poder cumplir con la norma ASTM-G65, objetivo principal de la utilización de esta máquina, también se tiene flexibilidad para poder realizar ensayos fuera de la misma. Ejemplos de esto, es la posibilidad de elegir una velocidad para la rueda distinta de los 200RPM, permitir una duración de hasta dos horas, etc.

Por último, también se incluyó la posibilidad de la adquisición de datos. Este punto no fue requerimiento al inicio del proyecto, pero se pudo implementar con el hardware disponible haciendo un pequeño esfuerzo desde el punto de vista del software. Resulta muy útil para el usuario tener a disposición los datos para su posterior análisis, y no quedarse solo con la curva mostrada durante el ensayo.

En conclusión, la automatización de la máquina para ensayos de abrasión cumple todos los puntos acordados al comienzo del proyecto. Esperamos que su utilización resulte de la utilidad para las tareas de investigación y la facultad.

Apéndice A

Código del Programa

```
 /**
  ******************************************************************************
  * File Name          : main.c
  * Description        : Main program body
  ******************************************************************************
  **/

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "adc.h"
#include "dac.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* USER CODE BEGIN Includes */

/*Pantalla------------------------------------------------------------------*/
#include "ft800.h"
#include "co_processor.h"
#include "ft_gpu.h"
#include "pantallas.h"
/*sd------------------------------------------------------------------*/
#include "Funciones propias.h"
#include "ff.h"

/*Genericos------------------------------------------------------------------*/
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/*Miscellaneous------------------------------------------------------------------*/
#define pi 3.141592
unsigned char i,cambio_teclado;
char tecla[1],txt[22], t1[10], t2[10], t3[10], t4[10], t5[10], t6[10], t7[10], t8[10], t9[10];
float offset_Celda = 0, tensionadc = 0, fuerza = 0;
float y = 0, Y1 = 0, y2 = 0, y3 = 0, y4 = 0, x = 0, x1 = 0, x2 = 0, x3 = 0, x4 = 0;
int numeropuntos,longitudstring, comparostring;
char termino_ensayo=0;
/*Para interrupciones------------------------------------------------------------------*/
/*TIM1*/
int registroadc;
float tiempoensayo,mu;
unsigned int resto,numeroint,k;
char tiempoensayo_char[10];
unsigned long numero_pulsos;
char read_outran=0;
/*TIM5*/
unsigned int contador1,contador2;
float frecuenciasensor;
/*TIM6*/
unsigned int contador_calibracion=0;
/*TIM9*/
float suma = 0;

/*Parametros del ensayo------------------------------------------------------------------*/
float frecuencia = 0, diametro = 0, distancia = 0, carganormal = 0, duracion = 0;
```

```
char frecuencia_char[10]="0", diametro_char[10]="0", distancia_char[10]="0";
char carganormal_char[10]="0", duracion_char[10]="0", mu_char[10] = "0";

/*Manejo de ft800-----------------------------------------------------------*/
unsigned char valortag,presionado;
unsigned int cmdBufferRd, cmdBufferWr;

float graficocada, registrodac, intervalografico, deltat, registrodac_anterior = 0;
int xpan[576], ypan[576],intervalograficoint;

/*Maquina de estados-------------------------------------------------*/
enum
Estados{Inicio,Tipicos,Rectificacion,Seteo,DecisionGuardar,NombreArchivo,InicioEnsayo,Ensayando,S
tandby};
enum Estados estado=Inicio;

/*SD-----------------------------------------------------------------------*/
char quiere_guardar;
/* FATFS -----------------------------------------------------------*/
typedef enum {
  SHIELD_NOT_DETECTED = 0,
  SHIELD_DETECTED
}ShieldStatus;

/* Private define ------------------------------------------------------------*/
#define SD_CARD_NOT_FORMATTED                    0
#define SD_CARD_FILE_NOT_SUPPORTED               1
#define SD_CARD_OPEN_FAIL                        2
#define FATFS_NOT_MOUNTED                        3

/* Private variables ---------------------------------------------------------*/

char* pDirectoryFiles[MAX_BMP_FILES];
FATFS SD_FatFs;  /* File system object for SD card logical drive */
char SD_Path[4]; /* SD card logical drive path */

FRESULT fr;
FIL fil, fil2;       /* File object */
UINT bw, bw2;
int num_files = 0;
char variables[] = "Velocidad de giro (RPM);Duracion (min);Distancia (m);Diametro (mm);Fuerza
(N)";
char paso_valores[]="tiempo(seg);distancia(m);u(mu)";
/*-------------------------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void Error_Handler(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

static void SDCard_Config(void);

/* USER CODE END PFP */

int main(void)
{
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
```

```
MX_DAC_Init();
MX_SPI2_Init();
MX_SPI5_Init();
MX_TIM5_Init();
MX_USART2_UART_Init();
MX_TIM6_Init();
MX_TIM1_Init();
MX_TIM9_Init();

ft800_Init(); // Inicio de pantalla

/*Calibracion-----------------------------------------------------------------*/
  calibrate();
/*Se empiezo a interrumpir para contar 4 minutos------------------------------*/
  HAL_TIM_Base_Start_IT(&htim6);

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    switch (estado)
    {
    case Inicio:

              //restablecer_variables();          //paso a todas las variables a su valor
              inicial
              PantallaInicio(valortag);           // menu principal
              HAL_Delay(500);
              while (valortag!=1 && valortag!=2) valortag = ft800memRead8(REG_TOUCH_TAG);

              ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

              presionado=valortag;
              PantallaInicio(valortag);
              HAL_Delay(200);                     //si se ve el efecto 3D
              while (presionado== 1 || presionado == 2 ) presionado =
              ft800memRead8(REG_TOUCH_TAG);
              PantallaInicio(presionado);

              switch (valortag)
              {
              case 1: estado=Tipicos;
              break;
              case 2:
                  frecuencia=200;
                  sprintf(frecuencia_char,"%6.2f",frecuencia);
                  estado=Rectificacion;
              break;
              }
    break;                //Inicio

    case Tipicos:

              PantallaTipicos();
              while(valortag!=0) valortag=ft800memRead8(REG_TOUCH_TAG); //espero hasta que se
              ponga en 0 de un toque de la pantalla anterior
              while(valortag!=3 && valortag!=4 && valortag!=5 && valortag!=6 && valortag!=103
              && valortag!=104) valortag = ft800memRead8(REG_TOUCH_TAG);

              ft800memWrite8(REG_PLAY, 1);    //suena valor seteado en ft800.c

              switch(valortag)
              {
                  case 3:                             //Procedimiento A

                      distancia=4309;                 //m
                      frecuencia=200;                 //RPM
                      carganormal=130;                //N
                      diametro=228.6;                 //mm
```

```
                duracion=distancia/(frecuencia*pi*(diametro/1000));  //divido por 1000
                para pasar a metros

                estado=Seteo;

            break;
            case 4:                             //TRIBO LAB 5

                distancia=718.2;
                frecuencia=200;
                carganormal=75;
                diametro=228.6;
                duracion=distancia/(frecuencia*pi*(diametro/1000));

                estado=Seteo;

            break;
            case 5:                              // TRIBO LAB 10

                distancia=1436.3;
                frecuencia=200;
                carganormal=75;
                diametro=228.6;
                duracion=distancia/(frecuencia*pi*(diametro/1000));

                estado=Seteo;
            break;

            case 6:                         //volver
                estado=Inicio;
            break;

        }
        sprintf(duracion_char,"%6.2f",duracion);
        sprintf(frecuencia_char,"%6.2f",frecuencia);
        sprintf(carganormal_char,"%6.2f",carganormal);
        sprintf(diametro_char,"%6.2f",diametro);
        sprintf(distancia_char,"%6.2f",distancia);
break;          //Tipicos

case Seteo:
        PantallaSeteo(valortag);
        HAL_Delay(500);
        while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
        while (valortag!=7 && valortag!=10 && valortag!=11 && valortag!=12 &&
        valortag!=13 && valortag!=14 && valortag!=15)
        valortag = ft800memRead8(REG_TOUCH_TAG);

        ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

        presionado=valortag;
        PantallaSeteo(valortag);          //para efecto tecla presionada
        HAL_Delay(200);                   //efecto 3d
        while (presionado!=0) presionado= ft800memRead8(REG_TOUCH_TAG);//espero que
        vuelva a 0
        PantallaSeteo(presionado);

        switch (valortag)                           // Toco para setear una variable
        {
            case 7:                          //frecuencia
                while(valortag!=9) //mientras no presiona Enter me quedo aca
                {
                    TecladoVariables(frecuencia_char,presionado);    //pantalla teclado
                    numerico
                    while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                    while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9
                    && valortag!=101)
                    valortag = ft800memRead8(REG_TOUCH_TAG);
```

```
            ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

            presionado=valortag;
            TecladoVariables(frecuencia_char,valortag);
            HAL_Delay(200); //efecto 3d
            while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

            if (valortag!=9) //9:Enter
            frecuencia=CargoValor (frecuencia_char,frecuencia,valortag);

        } //cierro  while
        duracion=distancia/(frecuencia*pi*(diametro/1000));    //cambio
        frecuencia actualizo duracion
        sprintf(duracion_char,"%6.2f",duracion);              //6 enteros dos
        decimales

    break;
    case 10:                          //duracion
        while(valortag!=9)
        {
            TecladoVariables(duracion_char,presionado);
            while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
            while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9
            && valortag!=101)
            valortag = ft800memRead8(REG_TOUCH_TAG);

            ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

            presionado=valortag;
            TecladoVariables(duracion_char,valortag);
            HAL_Delay(200); //efecto 3d
            while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

            if (valortag!=9)
            duracion=CargoValor (duracion_char,duracion,valortag);
        }// cierra while
        distancia=duracion*frecuencia*pi*diametro/1000;
        sprintf(distancia_char,"%6.2f",distancia);

    break;
    case 11:                          //diametro
        while(valortag!=9)
        {
            TecladoVariables(diametro_char,presionado);
            while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
            while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9
            && valortag!=101)
            valortag = ft800memRead8(REG_TOUCH_TAG);

            ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

            presionado=valortag;
            TecladoVariables(diametro_char,valortag);
            HAL_Delay(200); //efecto 3d
            while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

            if (valortag!=9)
            diametro=CargoValor (diametro_char,diametro,valortag);
        }
        duracion=distancia/(frecuencia*pi*(diametro/1000));
        distancia=duracion*frecuencia*pi*diametro/1000;
        sprintf(distancia_char,"%6.2f",distancia);
        sprintf(duracion_char,"%6.2f",duracion);

    break;
    case 12:                          //distancia
        while(valortag!=9)
        {
            TecladoVariables(distancia_char,presionado);
```

```
                    while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                    while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9
                    && valortag!=101)
                    valortag = ft800memRead8(REG_TOUCH_TAG);

                    ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

                    presionado=valortag;
                    TecladoVariables(distancia_char,valortag);
                    HAL_Delay(200); //efecto 3d
                    while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

                    if (valortag!=9)
                    distancia=CargoValor(distancia_char,distancia,valortag);
                }
                duracion=distancia/(frecuencia*pi*(diametro/1000));
                sprintf(duracion_char,"%6.2f",duracion);
            break;
            case 13:                        //Fuerza (carga normal)
                while(valortag!=9)
                {
                    TecladoVariables(carganormal_char,presionado);
                    while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                    while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9
                    && valortag!=101)
                    valortag = ft800memRead8(REG_TOUCH_TAG);

                    ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

                    presionado=valortag;
                    TecladoVariables(carganormal_char,valortag);
                    HAL_Delay(200); //efecto 3d
                    while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);
                    if (valortag!=9)
                    carganormal=CargoValor(carganormal_char,carganormal,valortag);
                }
            break;
            case 14:                        //Volver
                estado=Tipicos;
            break;
            case 15:                        //confirmar
                if
                ((frecuencia<350)&&(frecuencia>50)&&(diametro>200)&&(diametro<300)&&(carg
                anormal<294)&&
                    (carganormal>0)&& (distancia>0)&&(distancia<17236)&&(duracion>5)&&
                    (duracion<120))
                {
                    estado=DecisionGuardar;
                }
                else
                {
                    ErrorSeteo();       //error en valores ingresados
                    while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                    while (valortag==0)  //aprieto ok y vuelvo a seteo valortag=16
                    valortag = ft800memRead8(REG_TOUCH_TAG);
                    ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c
                }
            break;
        }  //termino switch valortag
break;          //Seteo

case DecisionGuardar:

            SiDeseaGuardar(presionado);
            while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
            while (valortag!=17 && valortag!=18) valortag = ft800memRead8(REG_TOUCH_TAG);

            ft800memWrite8(REG_PLAY, 1);
```

```
                presionado=valortag;
                SiDeseaGuardar(presionado);
                HAL_Delay(200); //efecto 3d
                while (presionado!=0) presionado = ft800memRead8(REG_TOUCH_TAG);

                if (valortag==17) // NO
                 {
                  quiere_guardar=0; //=no guardar valores
                  estado=InicioEnsayo;
                 }
                if (valortag==18) // SI
                 {
                    if (HAL_GPIO_ReadPin(GPIOB,CD_Pin)==0) // verifico este la SD (CD
                    activo BAJO)
                    {
                     quiere_guardar=1;
                     memset(txt, 0, 22); //limpio lo que habia en txt
                     i=0;                //empieza a escribir desde txt[0]
                     estado=NombreArchivo;
                    }
                    else
                    {
                        NoIngresoSD(presionado);
                        while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                        while (valortag!=19) valortag = ft800memRead8(REG_TOUCH_TAG);

                        ft800memWrite8(REG_PLAY, 1);

                        NoIngresoSD(valortag);
                        HAL_Delay(200); //efecto 3d
                    }
                 }

break;        //DecisionGuardar

case NombreArchivo:

            if (cambio_teclado==0)  TecladoABC(presionado) ;
            else                    Teclado123(presionado);

            while (valortag!=0) valortag = ft800memRead8(REG_TOUCH_TAG);
            while (valortag==0) valortag = ft800memRead8(REG_TOUCH_TAG);

            ft800memWrite8(REG_PLAY, 1);
            presionado=valortag;

            if (cambio_teclado==0)  TecladoABC(presionado) ;
            else                    Teclado123(presionado);

            HAL_Delay(200); //efecto 3d
            while (presionado!=0) presionado=ft800memRead8(REG_TOUCH_TAG);

            switch (valortag)
              {
               case 21:                    //borrar caracter
                              i--;
                              txt[i]='\0';
               break;

               case 22:                   //voy a numerico
                              cambio_teclado=1;
               break;

               case 23:                    //enter
                               strcat(txt,".csv");
                               estado=InicioEnsayo;

               break;
```

```
                case 24:                    //vuelvo a abc
                            cambio_teclado=0;
                break;
                default:                    //presiono una de la teclas alfanumericas
                            if (valortag==95 || (valortag<91 && valortag>64) ||
                            (valortag<58 && valortag>47) ||valortag==45 ||
                            (valortag>=97 && valortag<=122))
                            {
                                sprintf(tecla, "%c",valortag); //a tecla le paso el
                                valor tag con formato
                                if (i<17)       //esto para que no escriba un nombre
                                muy largo
                                {
                                    txt[i]=tecla[0];
                                    i++;
                                }
                            }

            }

break; //NombreArchivo
case InicioEnsayo:

        while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
        habilitacion_dac_grafico();
        Inicio_ensayo(valortag,presionado);
        HAL_Delay(500);
        while (valortag!=25 && valortag!=26) valortag = ft800memRead8(REG_TOUCH_TAG);
        //25:arrancar 26:volver

        ft800memWrite8(REG_PLAY, 1);
        presionado=valortag;
        Inicio_ensayo(valortag,presionado);

        HAL_Delay(200);
        while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

        if (valortag==25)                   //presiono arrancar
        {
            while(contador_calibracion<=1000) //todavia no pasaron los 4 minutos de la
            calibracion
            {
            }
            Inicio_ensayo(valortag,presionado);

            /*empieza a girar motor*/

            HAL_GPIO_WritePin(SENTIDO_GPIO_Port,SENTIDO_Pin,GPIO_PIN_RESET);        //
            sentido horario
            HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_RESET);    //
            Arranco el motor

            if (quiere_guardar==1)          //inicializo sd
            {
                SDCard_Config();
                fr = f_open(&fil, txt, FA_WRITE | FA_CREATE_ALWAYS);
                fr = f_write(&fil, variables, sizeof(variables), &bw);
                fr = f_write(&fil, "\n", 2, &bw);
                enviar_flotante(frecuencia);
                fr = f_write(&fil, ";", 1, &bw);
                enviar_flotante(duracion);
                fr = f_write(&fil, ";", 1, &bw);
                enviar_flotante(distancia);
                fr = f_write(&fil, ";", 1, &bw);
                enviar_flotante(diametro);
                fr = f_write(&fil, ";", 1, &bw);
                enviar_flotante(carganormal);
                fr = f_write(&fil, "\n", 2, &bw);
                fr = f_write(&fil, "\n", 2, &bw);
```

```
                fr = f_write(&fil, paso_valores, sizeof(paso_valores), &bw);
                fr = f_write(&fil, "\n", 2, &bw);

                if (fr != 0) //0=HAL_OK
                {
                    PantallaErrorSD();

                    while (valortag!=125 && valortag!=126) //(125=Salir) (126=Continuar)
                    valortag=ft800memRead8(REG_TOUCH_TAG);

                    if (valortag==125)
                    {
                        HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin,
                        GPIO_PIN_SET); //apago motor
                        estado=Inicio;
                    }
                    if (valortag==126)
                    {
                        Inicio_ensayo(25,0); //25 de haber apretado arrancar
                        HAL_Delay(500);
                    }

                }

            }

            if (valortag!=125) // si vale 125 q no ejecute lo siguiente y q se vaya a
            estado=Inicio, pues quiere Salir (errorSD)
            {
                numero_pulsos=(unsigned int)(8*distancia*1000/(pi*diametro));//numero
                de pulsos para que termine el ensayo

                while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                while (valortag!=27 && !HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin)) //sale
                cuando:valortag=27(detener) o outran=1
                valortag = ft800memRead8(REG_TOUCH_TAG);

                ft800memWrite8(REG_PLAY, 1);

                if (HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin)==1)
                {
                    ft800memWrite8(REG_PLAY, 1);

                    estado=Ensayando;           //se coloco la carga
                }

                if (valortag==27)        //detener antes de poner la carga
                {
                    ft800memWrite8(REG_PLAY, 1);
                    HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET);
                    //apago motor

                    if (quiere_guardar==1)
                    {
                      fr=f_close(&fil);
                      fr=f_mount(NULL, "", 0);
                    }
                } //cierro if de detener

            } //cierra if si es que hubo un error en sd y desea salir
        } //cierro if si presiono arrancar

        if (valortag==26)               //vuelvo a Tipicos
        {
            estado=Tipicos;             //si toco volver
        }

break; //InicioEnsayo
```

```
case Ensayando:

        pantallaEnsayando();

        HAL_TIM_Base_Start_IT(&htim1);              // Habilito interrupciones para
        graficar
        HAL_TIM_IC_Start_IT(&htim5, TIM_CHANNEL_2); // Habilito input capture para
        contar pulsos del sensor


        valortag=0;
        while (valortag!=28 && valortag!=29 && read_outran && !termino_ensayo)
        //28:detener - 29:ok cuando finaliza - termino_ensayo:termino numero de pulsos
        {
        }
        ft800memWrite8(REG_PLAY, 1);

        if (termino_ensayo==1) //termino el ensayo
        {
            pantallaEnsayando();
            while (valortag!=29) valortag= ft800memRead8(REG_TOUCH_TAG);  //espero
            hasta que apriete ok
            numeropuntos = 0;
            k=0;
            termino_ensayo=0;
            for (int j = 0; j<=288; j++)
            {
                xpan[j]=0;
                ypan[j]=0;
            }
            estado=Inicio;
        }

        if(valortag!=29) //si presiona ok que no entre aca
        {
        if (HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin)==0) //se saco la carga
        {
         HAL_TIM_Base_Stop_IT(&htim1);//paro interrupciones
         HAL_TIM_IC_Stop_IT(&htim5, TIM_CHANNEL_2);
         ft800memWrite16(REG_SOUND,0x6c07); //pongo alarma a la pantalla
         ft800memWrite8(REG_PLAY, 1);
         estado=Standby;
        }
        }

        if (valortag==28)                   //Detener
        {
            HAL_TIM_Base_Stop_IT(&htim1);//paro interrupciones
            HAL_TIM_IC_Stop_IT(&htim5, TIM_CHANNEL_2);

            Ensayando_detenido();
            while (valortag!=0) valortag = ft800memRead8(REG_TOUCH_TAG);
            HAL_Delay(500);
            while (valortag!=130 && valortag!=131) valortag =
            ft800memRead8(REG_TOUCH_TAG);//130:Salir, 131:Continuar

            ft800memWrite8(REG_PLAY, 1);

            if (valortag==130) //salir
            {
                HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET);
                //paro el motor
                numeropuntos = 0;
                numeroint=0;
                k=0;
                if (quiere_guardar==1)
                {
                    fr=f_close(&fil);
                    fr=f_mount(NULL, "", 0);
```

```
                }
                estado=Inicio;
            } //cierro if de Salir
            if (valortag==131) HAL_Delay(500);// continuar
        }// cierro if de detener

break; //Ensayando

case Standby:

        pantallaStandby();

        HAL_Delay(3000);

        ft800memWrite16(REG_SOUND,0x00);    //silencio
        ft800memWrite8(REG_PLAY, 1);        //paro alarma
        ft800memWrite16(REG_SOUND,0x0057);  //vuelvo a sonido standar

        while (valortag!=0) valortag = ft800memRead8(REG_TOUCH_TAG);
        while (valortag!=34 && !HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin))     //34:salir
        valortag = ft800memRead8(REG_TOUCH_TAG);

        ft800memWrite8(REG_PLAY, 1);

        if (HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin)==1) //se coloco la carga nuevamente
        estado=Ensayando;

        if (valortag==34) //salir
        {
        HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET); //paro el
        motor
        numeropuntos = 0;
        numeroint=0;
        k=0;
        if (quiere_guardar==1)
            {
              fr=f_close(&fil);
              fr=f_mount(NULL, "", 0);
            }
        estado=Inicio;
        }
break; //Standby

case Rectificacion:

        while(valortag!=0) valortag=ft800memRead8(REG_TOUCH_TAG);
        pantallaRectificacion(presionado);
        HAL_Delay(500);
        while (valortag!=30 && valortag!=31 && valortag!=32 && valortag!=33 ) valortag
        = ft800memRead8(REG_TOUCH_TAG);

        ft800memWrite8(REG_PLAY, 1);
        presionado=valortag;
        pantallaRectificacion(presionado);
        while(presionado!=0) presionado=ft800memRead8(REG_TOUCH_TAG);
        HAL_Delay(200);
        pantallaRectificacion(valortag);

        switch(valortag)
        {
        case 30:         //modificar frecuencia
            while(valortag!=9)
            {
                TecladoVariables(frecuencia_char,presionado);
                while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                while ((valortag<46 || valortag>57) && valortag!=8 && valortag!=9 &&
                valortag!=101)
                valortag = ft800memRead8(REG_TOUCH_TAG);
```

```
                    ft800memWrite8(REG_PLAY, 1); //suena valor seteado en ft800.c

                    presionado=valortag;
                    TecladoVariables(frecuencia_char,valortag);
                    HAL_Delay(200); //efecto 3d
                    while (presionado != 0) presionado = ft800memRead8(REG_TOUCH_TAG);

                    if (valortag!=9)
                    {
                      frecuencia=CargoValor (frecuencia_char,frecuencia,valortag);
                      //frecuencia=frecuencia*7.5;///para tener en cuent la reduccion
                      //depeues lo voy a cambiar en la funcion del dac
                      duracion=distancia/(frecuencia*pi*(diametro/1000));   //si setean
                      radio 2*pi*radio OJO!!               //cambio frecuencia actualizo
                      duracion
                      sprintf(duracion_char,"%6.2f",duracion);         //6 enteros dos
                      decimales
                    }
                }
              break;
              case 31:         //Arrancar

                if (frecuencia<350 && frecuencia>50)
                {
                    HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
                    registrodac = -372.5 + 12.41*frecuencia;      // Esta funcion permite
                    setear una velocidad desde 50 RPM a 350RPM
                    registrodac = round(registrodac);
                    HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
                    // seteo referencia
                    HAL_GPIO_WritePin(SENTIDO_GPIO_Port,SENTIDO_Pin,GPIO_PIN_SET);
                    //sentido antihorario
                    HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_RESET);
                    // Arranco el motor
                    presionado=31;
                }
                else
                {
                    ft800memWrite8(REG_PLAY, 1);
                    while (valortag!=0) valortag= ft800memRead8(REG_TOUCH_TAG);
                    ErrorRectificacion();
                    while (valortag!=140) valortag= ft800memRead8(REG_TOUCH_TAG); //140:OK
                    ft800memWrite8(REG_PLAY, 1);

                }
              break;
              case 32:  //Detener

                  HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET); //paro
                  el motor

              break;
              case 33:

                  estado=Inicio; //toco volver ,vuelvo estado inicial

              break;
              }
     break; //Rectificacion

     } //ciero switch principal
  }
  /* USER CODE END WHILE */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
```

```
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;

    /**Configure the main internal regulator output voltage
    */
  __HAL_RCC_PWR_CLK_ENABLE();

  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 96;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USER CODE BEGIN 4 */

/*funciones sd ------------------------------------------------------------*/
static void SDCard_Config(void)
{
  uint32_t counter = 0;

  if(FATFS_LinkDriver(&SD_Driver, SD_Path) == 0)
  {
    /* Initialize the SD mounted on adafruit 1.8" TFT shield */
    BSP_SD_Init();

    /* Check the mounted device */
    fr = f_mount(&SD_FatFs, (TCHAR const*)"/", 0);

        /* Initialize the Directory Files pointers (heap) */
```

```
        for (counter = 0; counter < MAX_BMP_FILES; counter++)
        {
            pDirectoryFiles[counter] = malloc(11);
        }

  }
}


void enviar_flotante(float f)
{
   int int_part, fraction;
   int_part = (int) f;
   fraction = (f - int_part) * 1000.0;   // three digits after decimal.
   fr=f_printf(&fil, "%d", int_part);
   fr = f_write(&fil, ",", 1, &bw);
   f_printf(&fil, "%d", fraction);
}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

/**
  * @}
  */

/**
  * @}
*/

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : main.h
  * Description        : This file contains the common defines of the application
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H
  /* Includes ------------------------------------------------------------------*/

/* USER CODE BEGIN Includes */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_nucleo.h"
#include "stm32_adafruit_sd.h"
#include "stm32_adafruit_lcd.h"
#include <stdio.h>
#include <stdlib.h>

/* FatFs includes component */
#include "ff_gen_drv.h"
#include "sd_diskio.h"
#include "fatfs_storage.h"

/*eeprom*/
#include "eeprom.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
#define MAX_BMP_FILES 25
#define MAX_BMP_FILE_NAME 11


/* USER CODE END Includes */

/* Private define ------------------------------------------------------------*/

#define ARRANQUE_Pin GPIO_PIN_0
#define ARRANQUE_GPIO_Port GPIOA
#define CS_SD_Pin GPIO_PIN_4
#define CS_SD_GPIO_Port GPIOC
#define SPI5_SCK_PANT_Pin GPIO_PIN_0
#define SPI5_SCK_PANT_GPIO_Port GPIOB
#define CD_Pin GPIO_PIN_1
#define CD_GPIO_Port GPIOB
#define CS_PAN_Pin GPIO_PIN_12
#define CS_PAN_GPIO_Port GPIOB
#define SPI2_SCK_SD_Pin GPIO_PIN_13
#define SPI2_SCK_SD_GPIO_Port GPIOB
#define SPI2_MISO_SD_Pin GPIO_PIN_14
#define SPI2_MISO_SD_GPIO_Port GPIOB
#define SPI2_MOSI_SD_Pin GPIO_PIN_15
#define SPI2_MOSI_SD_GPIO_Port GPIOB
#define OUTRAN_Pin GPIO_PIN_10
#define OUTRAN_GPIO_Port GPIOA
#define PD_PAN_Pin GPIO_PIN_11
#define PD_PAN_GPIO_Port GPIOA
#define SPI5_MISO_PANT_Pin GPIO_PIN_12
#define SPI5_MISO_PANT_GPIO_Port GPIOA
#define SENTIDO_Pin GPIO_PIN_11
#define SENTIDO_GPIO_Port GPIOB
#define SPI5_MOSI_PANT_Pin GPIO_PIN_8
#define SPI5_MOSI_PANT_GPIO_Port GPIOB


#endif /* __MAIN_H */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    stm32f4xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  **/

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_hal.h"
#include "stm32f4xx.h"
#include "stm32f4xx_it.h"
#include <math.h>
#include "pantallas.h"
#include "ft800.h"

/* USER CODE BEGIN 0 */
#include "ff.h"
#include "Funciones propias.h"
/* de la sd -------------------------*/
extern FRESULT fr;
extern FIL fil;       /* File object */
extern UINT bw;


/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/
extern ADC_HandleTypeDef hadc1;
extern TIM_HandleTypeDef htim1;
extern TIM_HandleTypeDef htim5;
extern TIM_HandleTypeDef htim6;
extern TIM_HandleTypeDef htim9;
extern DAC_HandleTypeDef hdac;

extern char quiere_guardar,termino_ensayo;
extern unsigned char valortag;
extern int ypan[576], xpan[576];
extern unsigned int contador1, contador2, numeroint, resto;
extern int intervalograficoint;
extern int numeropuntos, registroadc;
extern float  frecuenciasensor;
extern float carganormal, tiempoensayo, suma, offset_Celda, fuerza;
extern float y, Y1, y2, y3, y4, x, x1, x2, x3, x4;
extern float diametro,distancia,frecuencia;
extern float registrodac, registrodac_anterior;
extern char tiempoensayo_char[10], mu_char[10],distancia_char[10];

extern unsigned long numero_pulsos;
extern char read_outran;
extern float mu;
#define pi 3.141592

extern unsigned int contador_calibracion,k;

unsigned char ultimo_ajuste = 0, primer_ajuste = 0;
unsigned int j = 0, pulsos = 0;
float suma_sensor = 0, error = 0, erroranterior = 0;

/******************************************************************************/
/*            Cortex-M4 Processor Interruption and Exception Handlers         */
/******************************************************************************/

/**
* @brief This function handles Non maskable interrupt.
*/
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
```

```
  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
* @brief This function handles Hard fault interrupt.
*/
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN HardFault_IRQn 1 */

  /* USER CODE END HardFault_IRQn 1 */
}

/**
* @brief This function handles Memory management fault.
*/
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN MemoryManagement_IRQn 1 */

  /* USER CODE END MemoryManagement_IRQn 1 */
}

/**
* @brief This function handles Pre-fetch fault, memory access fault.
*/
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN BusFault_IRQn 1 */

  /* USER CODE END BusFault_IRQn 1 */
}

/**
* @brief This function handles Undefined instruction or illegal state.
*/
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
  }
  /* USER CODE BEGIN UsageFault_IRQn 1 */

  /* USER CODE END UsageFault_IRQn 1 */
}

/**
* @brief This function handles System service call via SWI instruction.
*/
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */
```

```
  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}

/**
* @brief This function handles Debug monitor.
*/
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}

/**
* @brief This function handles Pendable request for system service.
*/
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}

/**
* @brief This function handles System tick timer.
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}

/******************************************************************************/
/* STM32F4xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                    */
/******************************************************************************/

/**
* @brief This function handles ADC1 global interrupt.
*/
void ADC_IRQHandler(void)
{
  /* USER CODE BEGIN ADC_IRQn 0 */

  /* USER CODE END ADC_IRQn 0 */
  HAL_ADC_IRQHandler(&hadc1);
  /* USER CODE BEGIN ADC_IRQn 1 */

  /* USER CODE END ADC_IRQn 1 */
}

/**
* @brief This function handles TIM1 update interrupt.
*/
void TIM1_UP_IRQHandler(void) //interrumpe cada 10ms
```

```
//utilizado para tomar valores de la celda de carga, guadar valores en sd y graficar
{
    numeroint++;            // cuento numero de interrupciones
    k++;                    // cuento numero de muestras para el filtro digital
    registroadc = HAL_ADC_GetValue(&hadc1); // Tomo valor el adc
    HAL_ADC_Start(&hadc1);  // Vuelvo a iniciar adc para tomar nuevo valor en proxima
    interrupcion
    if (k < 100)            // Promedia el filtro
    {
        x4 = x3;
        x3 = x2;
        x2 = x1;
        x1 = x;
        y4 = y3;
        y3 = y2;
        y2 = Y1;
        Y1 = y;
        x = registroadc;
        y = 2.234256192678642e-05*x + 8.937024770714570e-05*x1 + 1.340553715607186e-04*x2 +
        8.937024770714570e-05*x3 + 2.234256192678642e-05*x4
                                  + 3.449986651647306*Y1 - 4.463402961204221*y2 +
                                  2.566446772846271*y3 - 0.553387944280184*y4;
    }
    else                                  // El filtro ya promedio, valor final
    {
        fuerza = (y/4095.0*3.3 - offset_Celda)/0.1091;
        mu = fuerza*9.8067/carganormal;     // calculo mu
        sprintf(mu_char,"%5.3f",mu);

        k = 0;
        y = 0;
        Y1 = 0;
        y2 = 0;
        y3 = 0;
        y4 = 0;
        x = 0;
        x1 = 0;
        x2 = 0;
        x3 = 0;
        x4 = 0;

        tiempoensayo = (numeroint/100.0);   // calculo cuanto va de ensayo
        sprintf(tiempoensayo_char,"%6.2f",tiempoensayo);
        distancia = tiempoensayo*(frecuencia/60)*pi*(diametro/1000);
        sprintf(distancia_char,"%6.2f",distancia);

        if (quiere_guardar==1)
        {
            fr = f_write(&fil, tiempoensayo_char, sizeof(tiempoensayo_char), &bw);
            fr = f_write(&fil, ";", 1, &bw);
            fr = f_write(&fil, distancia_char, sizeof(distancia_char), &bw);
            fr = f_write(&fil, ";", 1, &bw);
            fr = f_write(&fil, mu_char, sizeof(mu_char), &bw);
            fr = f_write(&fil, "\n", 2, &bw);
        }
    }
    resto = numeroint%intervalograficoint;  // evaluo este resto para saber si tengo que
    graficar o no un punto
    if (resto == 0)
    {
        //numeropuntos++;
        if (numeropuntos < 288) // grafico puntos hasta llegar a 288 puntos
        {
            numeropuntos++;
            xpan[numeropuntos] = 864.0 + 20*numeropuntos;
            ypan[numeropuntos] = 2608.0 - mu*2176.0; ;
            ypan[numeropuntos] = round(ypan[numeropuntos]);
        }
        if (numeropuntos == 1)
        {
            ypan[0] = ypan[1];
            xpan[0] = 864.0;
        }
```

```
        pantallaEnsayando();
    }
    valortag = ft800memRead8(REG_TOUCH_TAG);
    read_outran=HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin);
    HAL_TIM_IRQHandler(&htim1);
}

/**
* @brief This function handles TIM5 global interrupt.
*/
void TIM5_IRQHandler(void) //input capture
{ //se cuentan los pulsos del sensor inductivo
    if (pulsos < 50)
    {
        pulsos++;                   // Cuento la cantidad de pulsos hasta que llegue a 50
    }
    contador1 = contador2;
    contador2 = HAL_TIM_ReadCapturedValue(&htim1,TIM_CHANNEL_2);
    frecuenciasensor = (96000000.0/htim5.Init.Prescaler)/(contador2-contador1)*60.0;    //
    La velocidad expresada en RPM
    if (pulsos == 50)   // Espero 50 pulsos para que la velocidad se estabilice
    {
        j++;
        suma_sensor = frecuenciasensor + suma_sensor;
        if (j == 10)                // Hago una promediacion de 10 valores para eliminar
        algo de ruido en la medicion
        {
            frecuenciasensor = suma_sensor/j;
            controlDAC();           // Con la velocidad ya estabilizada, ajusto para reducir
            el error
            j = 0;
        }
    }
    if(HAL_GPIO_ReadPin(GPIOA,OUTRAN_Pin))
    {
        numero_pulsos--;
        if (numero_pulsos == 0) // fin del ensayo, paro
        {
            HAL_TIM_Base_Stop_IT(&htim1);   // Inhabilito interrupciones para graficar
            HAL_TIM_IC_Stop_IT(&htim5, TIM_CHANNEL_2);  // Inhabilito input capture para
            contar pulsos del sensor
            HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET);  // Paro el
            motor
            HAL_ADC_Stop(&hadc1);   // inhabilito las conversiones en el adc
            HAL_DAC_Stop(&hdac, DAC_CHANNEL_1);
            numeroint = 0;
            //numeropuntos = 0;
            termino_ensayo=1;
            if (quiere_guardar==1) //cierro la sd
            {
                fr=f_close(&fil);
                fr=f_mount(NULL, "", 0);
            }

        }
    }
  HAL_TIM_IRQHandler(&htim5);

}

/**
* @brief This function handles TIM6 global interrupt and DAC channel underrun error interrupt.
*/
void TIM6_DAC_IRQHandler(void)
{ //se esperan 4 minutos para tomar el offset de la celda
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */
    contador_calibracion++;
    if (contador_calibracion>80)      //interrumpe cada 3 seg
    {                                 //4min
        HAL_TIM_Base_Stop_IT(&htim6); //para interrupciones
        HAL_TIM_Base_Start_IT(&htim9);
        HAL_ADC_Start(&hadc1);
        //contador_calibracion = 0;
```

```
    }
    if (valortag==25)
    {
        PantallaEspera6();
    }
  /* USER CODE END TIM6_DAC_IRQn 0 */
  HAL_TIM_IRQHandler(&htim6);
  /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

  /* USER CODE END TIM6_DAC_IRQn 1 */
}

/**
* @brief This function handles TIM1 break interrupt and TIM9 global interrupt.
*/
void TIM1_BRK_TIM9_IRQHandler(void)
{
    //se calcula el offset
    k++;
    registroadc = HAL_ADC_GetValue(&hadc1); // Tomo valor el adc
    HAL_ADC_Start(&hadc1);  // Vuelvo a iniciar adc para tomar nuevo valor en proxima
    interrupcion
    contador_calibracion++;
    if (contador_calibracion <= 1000)
    {
        if (k <= 100)           // Promedia el filtro
        {
            x4 = x3;
            x3 = x2;
            x2 = x1;
            x1 = x;
            y4 = y3;
            y3 = y2;
            y2 = Y1;
            Y1 = y;
            x = registroadc;
            y = 2.234256192678642e-05*x + 8.937024770714570e-05*x1 +
            1.340553715607186e-04*x2 + 8.937024770714570e-05*x3 + 2.234256192678642e-05*x4
                        + 3.449986651647306*Y1 - 4.463402961204221*y2 +
                        2.566446772846271*y3 - 0.553387944280184*y4;
        }
        else
        {
            k = 0;
            //enviar_flotante(y);
            //fr = f_write(&fil, "\n", 2, &bw);
            suma = suma + y;
            y=0;
            Y1 = 0;
            y2 = 0;
            y3 = 0;
            y4 = 0;
            x = 0;
            x1 = 0;
            x2 = 0;
            x3 = 0;
            x4 = 0;
        }
    }
    else
    {
        k=0;
        offset_Celda = suma/contador_calibracion*100.0/4095.0*3.3;
        HAL_TIM_Base_Stop_IT(&htim9);
        //HAL_TIM_Base_Start_IT(&htim1);
    }

  HAL_TIM_IRQHandler(&htim1);
  HAL_TIM_IRQHandler(&htim9);
}

void controlDAC (void)
{
```

```
    if (ultimo_ajuste == 0)                                                 // Dejo de
    entrar a este if cuando tengo el valor optimo
    {
        if (primer_ajuste == 0)                                             // Si es el
        primer ajuste, no puedo evaluar la evolucion del error
        {
            primer_ajuste++;
            // Entro una sola vez en este if
            error = frecuencia*8.0 - frecuenciasensor;      // El error es la diferencia
            entre la velocidad esperada en el sensor y la medida
            error = fabs(error);                                                //
            Evaluo el error absoluto del error
            if (frecuenciasensor < frecuencia*8.0)              // Si la velocidad medida
            esta debajo de la referencia, aumento la tension del DAC
            {
                registrodac_anterior = registrodac;
                registrodac++;
                HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
            }

            else
                    // Si la velocidad medida esta por arriba de la referencia, bajo la
            tension del DAC
            {
                registrodac_anterior = registrodac;
                registrodac--;
                HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
            }
        }

        else
                // En el resto de los ajuste evaluo si el error aumenta o disminuye
        {
            erroranterior = error;
            error = frecuencia*8.0 - frecuenciasensor;      // El error es la diferencia
            entre la velocidad esperada en el sensor y la medida
            error = fabs(error);                                                //
            Evaluo el error absoluto del error
            if (error < erroranterior)                                  // Si el error
            baja, sigo modificando
            {
                if (frecuenciasensor < frecuencia*8.0)          // Si la velocidad medida
                esta debajo de la referencia, aumento la tension del DAC
                {
                    registrodac_anterior = registrodac;
                    registrodac++;
                    HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
                }

                else
                        // Si la velocidad medida esta por arriba de la referencia, bajo la
                tension del DAC
                {
                    registrodac_anterior = registrodac;
                    registrodac--;
                    HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
                }
            }

            else
                    // Si el error se agrando, vuelvo al valor anterior, que es el optimo
            {
                registrodac = registrodac_anterior;
                HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac);
                ultimo_ajuste++;
            }
        }
    }

}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    stm32f4xx_it.h
  * @brief   This file contains the headers of the interrupt handlers.
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions --------------------------------------------------------- */

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);
void ADC_IRQHandler(void);
void TIM1_UP_IRQHandler(void);
void TIM5_IRQHandler(void);

#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
 /**
  ******************************************************************************
  * File Name          : pantallas.c
  * Description        : los comandos que se envian al ft800 para graficar cada pantalla
  ******************************************************************************
 **/
/* Includes */
#include "main.h"
#include "ft_gpu.h"
#include "ft800.h"
#include "co_processor.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include "pantallas.h"


/* Variables */
extern DAC_HandleTypeDef hdac;
extern unsigned int cmdBufferRd, cmdBufferWr;
extern long numero_pulsos;
extern char txt[22], t1[10], t2[10], t3[10], t4[10], t5[10], t6[10], t7[10], t8[10], t9[10];
extern char frecuencia_char[10], diametro_char[10], distancia_char[10];
extern char duracion_char[10], carganormal_char[10];
extern int xpan[576], ypan[576], intervalograficoint, numeropuntos;
extern float graficocada, registrodac, intervalografico, deltat, frecuencia, duracion;
char toogle_teclado;
extern unsigned int contador_calibracion;

/*********************************************/

/* Funciones */
void calibrate (void)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR(1,1,1));
    cmd(COLOR_RGB(255,255,255));
    cmd_text(240,102,30, 1536, "Por favor toque ");
    cmd_text(240,148,30, 1536, "los puntos");
    cmd_calibrate();
    finpantalla();
}

void PantallaInicio (unsigned char pres)    // menu principal
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(1));
    if (pres==1)
    cmd_button(25,160, 200,50, 30,OPT_FLAT, "");
    else cmd_button(25,160, 200,50, 30,0, "Ensayo");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(2));
    if (pres==2) cmd_button(255,160, 200,50, 30,OPT_FLAT, "");
    else cmd_button(255,160, 200,50, 30,0, "Rectificacion");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,255));
    cmd_text(241,39,29, 1536, "Universidad Nacional de Mar del Plata");
```

```
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,255));
    cmd_text(239,79,29, 1536, "Facultad de Ingenieria");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void PantallaTipicos()
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,85,127));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(10));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(0,1088));
    cmd(VERTEX2F(7680,1088));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(10));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(0,2176));
    cmd(VERTEX2F(7680,2176));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(10));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(0,3264));
    cmd(VERTEX2F(7680,3264));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x00557f);
    cmd_gradcolor(0x00007f);
    cmd(TAG(3));
    cmd_button(129,4, 220,60, 29,256, "PROCEDIMIENTO A");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x00557f);
    cmd_gradcolor(0x00007f);
    cmd(TAG(4));
    cmd_button(130,72, 220,60, 29,256, "TRIBO LAB 5");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x00557f);
    cmd_gradcolor(0x00557f);
    cmd(TAG(5));
    cmd_button(130,140, 220,60, 29,256, "TRIBO LAB 10");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x00557f);
    cmd_gradcolor(0x00557f);
    cmd(TAG(6));
    cmd_button(130,208, 220,60, 29,256, "Volver");
    cmd(RESTORE_CONTEXT());
    finpantalla();

}


void PantallaSeteo (unsigned char pres) // seteo
{
```

```
iniciopantalla();
cmd_dlstart();
cmd(CLEAR_COLOR_RGB(0,0,0));
cmd(CLEAR(1,1,1));
cmd(SAVE_CONTEXT());
cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,255));
cmd_text(240,22,29, 1536, "Defina los parametros del ensayo");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x003870);
cmd_gradcolor(0xffffff);
cmd(TAG(15));
if (pres==15) cmd_button(270,223, 190,45, 30,OPT_FLAT, "");
else cmd_button(270,223, 190,45, 30,0, "Confirmar");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x003870);
cmd_gradcolor(0xffffff);
cmd(TAG(14));
if (pres==14) cmd_button(20,223, 190,45, 30,OPT_FLAT, "");
else cmd_button(20,223, 190,45, 30,0, "Volver");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x2e7f76);
cmd_gradcolor(0xffffff);
cmd(TAG(12));
if (pres==12) cmd_button(20,50, 100,40, 28,OPT_FLAT, "");
else cmd_button(20,50, 100,40, 28,0, "Distancia");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x2e7f76);
cmd_gradcolor(0xffffff);
cmd(TAG(7));
if (pres==7) cmd_button(20,110, 100,40, 28,OPT_FLAT, "");
else cmd_button(20,110, 100,40, 28,0, "Velocidad");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x2e7f76);
cmd_gradcolor(0xffffff);
cmd(TAG(13));
if (pres==13) cmd_button(140,170, 100,40, 28,OPT_FLAT, "");
else cmd_button(140,170, 100,40, 28,0, "Carga");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd(LINE_WIDTH(16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(1920,800));
cmd(VERTEX2F(3520,1440));
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd(LINE_WIDTH(16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(1920,1760));
cmd(VERTEX2F(3520,2400));
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd(LINE_WIDTH(16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(3840,2720));
cmd(VERTEX2F(5440,3360));
```

```
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x2e7f76);
cmd_gradcolor(0xffffff);
cmd(TAG(10));
if (pres==10) cmd_button(260,50, 100,40, 28,OPT_FLAT, "");
else cmd_button(260,50, 100,40, 28,0, "Duracion");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x2e7f76);
cmd_gradcolor(0xffffff);
cmd(TAG(11));
if (pres==11) cmd_button(260,110, 100,40, 28,OPT_FLAT, "");
else cmd_button(260,110, 100,40, 28,0, "Diametro");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd(LINE_WIDTH(16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(5760,800));
cmd(VERTEX2F(7360,1440));
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd(LINE_WIDTH(16));
cmd(BEGIN(RECTS));
cmd(VERTEX2F(5760,1760));
cmd(VERTEX2F(7360,2400));
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(170,70,28, 1536, distancia_char);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(170,130,28, 1536, frecuencia_char);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(410,70,28, 1536, duracion_char);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(410,130,28, 1536, diametro_char);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(290,190,28, 1536, carganormal_char);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(211,85,27, 1536, "m");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(447,85,27, 1536, "min");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(205,146,26, 1536, "RPM");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(448,143,27, 1536, "mm");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(0,0,0));
cmd_text(331,204,27, 1536, "N");
```

```
    cmd(RESTORE_CONTEXT());
    finpantalla();

}

void ErrorSeteo()   // error de configuracion
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(170,0,0));
    cmd_text(240,25,31, 1536, "Error de configuracion");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(240,65,29, 1536, "Verifique el rango de validez");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(240,86,29, 1536, "de los parametros:");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,126,27, 1536, "50 RPM < Velocidad < 350 RPM");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,145,27, 1536, "200 mm < Diametro  <  300 mm");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,164,27, 1536, "5 min. < Duracion < 120 min.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,184,27, 1536, "160 m < Distancia < 17236 m");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,203,27, 1536, "0 N <   Fuerza   < 294 N");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(16));
    cmd_button(180,221, 120,39, 31,0, "OK");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void SiDeseaGuardar(unsigned char pres)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(18));
    if (pres==18) cmd_button(270,140, 160,60,31,OPT_FLAT, "");
    else cmd_button(270,140, 160,60, 31,0, "SI");
    cmd(RESTORE_CONTEXT());
```

```
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(17));
    if (pres==17) cmd_button(50,140, 160,60, 31,OPT_FLAT, "");
    else cmd_button(50,140, 160,60, 31,0, "NO");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,255));
    cmd_text(243,62,29, 1536, "Desea guardar los datos del ensayo");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void NoIngresoSD(unsigned char pres)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,255));
    cmd_text(240,58,30, 1536, "No ha ingresado la tarjeta SD");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(19));
    if (pres==19) cmd_button(160,140, 160,60, 31,OPT_FLAT, "OK");
    else cmd_button(160,140, 160,60, 31,0, "OK");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void TecladoABC (unsigned char pres)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(131,131,131));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    if (pres==125)
    {
        if (toogle_teclado==0) toogle_teclado=1;
        else toogle_teclado=0;
    }
    if (toogle_teclado==0)
    {
    cmd_keys(0,61,480,45, 30,pres, "QWERTYUIOP");
    cmd_keys(24,115,432,45, 30,pres, "ASDFGHJKL");
    cmd_keys(48,167,384,45, 30,pres, "ZXCVBNM");
    }
    else
    {
    cmd_keys(0,61,480,45, 30,pres, "qwertyuiop");
    cmd_keys(24,115,432,45, 30,pres, "asdfghjkl");
    cmd_keys(48,167,384,45, 30,pres, "zxcvbnm");
    }
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(22));
    if (pres==22)
```

```
        {
        cmd_fgcolor(0x00007f);
        cmd_button(0,220, 157,45, 29,OPT_FLAT, "");
        }
    else cmd_button(0,220, 157,45, 29,0, "123");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd_keys(163,220,154,45, 31,pres, "_");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,255,0));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(23));
    if (pres==23)
        {
        cmd_fgcolor(0x00007f);
        cmd_button(323,220, 157,45, 30,OPT_FLAT, "Enter");
        }
    else cmd_button(323,220, 157,45, 30,0, "Enter");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(21));
    if (pres==21)
        {
        cmd_fgcolor(0x00007f);
        cmd_button(396,6, 80,45, 31,OPT_FLAT, "<-");
        }
    else cmd_button(396,6, 80,45, 31,0, "<-");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd(LINE_WIDTH(16));
    cmd(BEGIN(RECTS));
    cmd(VERTEX2F(1600,96));
    cmd(VERTEX2F(5760,816));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(354,26,29, 3072, txt);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0xffffff);
    cmd(TAG(125));
    if (pres==125)
    {
        cmd_fgcolor(0x00007f);
        cmd_button(12,6, 72,45, 23,OPT_FLAT, "shift");
    }
    else cmd_button(12,6, 72,45, 23,0, "shift");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void Teclado123 (unsigned char pres)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(131,131,131));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
```

```
    cmd_keys(120,61,180,45, 30,pres, "789");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd_keys(120,114,240,45, 30,pres, "456-");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd_keys(120,167,240,45, 30,pres, "1230");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(24));
    if (pres==24)
        {
        cmd_fgcolor(0x00007f);
        cmd_button(0,220, 157,45, 29,OPT_FLAT, "Abc");
        }
    else cmd_button(0,220, 157,45, 29,0, "Abc");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd_keys(163,220,154,45, 31,pres, "_");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,255,0));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(23));
    if (pres==23)
        {
        cmd_fgcolor(0x00007f);
        cmd_button(323,220, 157,45, 30,OPT_FLAT, "Enter");
        }
    else cmd_button(323,220, 157,45, 30,0, "Enter");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x703800);
    cmd_gradcolor(0x703800);
    cmd(TAG(21));
    if (pres==21)
        {
        cmd_fgcolor(0x00007f);
        cmd_button(396,6, 80,45, 31,OPT_FLAT, "");
        }
    else cmd_button(396,6, 80,45, 31,0, "<-");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd(LINE_WIDTH(16));
    cmd(BEGIN(RECTS));
    cmd(VERTEX2F(1600,96));
    cmd(VERTEX2F(5760,816));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(354,26,29, 3072, txt);
    cmd(RESTORE_CONTEXT());
    finpantalla();

}
void TecladoVariables (char variable_char[10], unsigned char pres)  // teclado
{
```

```
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(85,0,255));
    cmd(LINE_WIDTH(16));
    cmd(BEGIN(RECTS));
    cmd(VERTEX2F(2064,320));
    cmd(VERTEX2F(5616,4032));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x5500ff);
    cmd_gradcolor(0x000000);
    cmd_keys(133,111,160,45, 30,pres, "456");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x5500ff);
    cmd_gradcolor(0x000000);
    cmd_keys(133,158,160,45, 30,pres, "123");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x5500ff);
    cmd_gradcolor(0x000000);
    cmd_keys(133,205,108,45, 30,pres, "0.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x5500ff);
    cmd_gradcolor(0x000000);
    cmd_keys(133,64,160,45, 30,pres, "789");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x00b890);
    cmd_gradcolor(0x00ff00);
    cmd(TAG(9));
    cmd_button(244,205, 103,45, 28,0, "Enter");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0xffc870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(8));
    if (pres==8) cmd_button(294,111, 53,45, 28,OPT_FLAT, "");
    else cmd_button(294,111, 53,45, 28,0, "Clr");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0xffc870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(101));
    if (pres==101) cmd_button(294,158, 53,45, 28,OPT_FLAT, "");
    else cmd_button(294,158, 53,45, 28,0, "Del");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd(LINE_WIDTH(16));
    cmd(BEGIN(RECTS));
    cmd(VERTEX2F(2128,384));
    cmd(VERTEX2F(5552,960));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(345,42,30, 3072, variable_char);
    cmd(RESTORE_CONTEXT());
    finpantalla();

}
```

```
void Inicio_ensayo(unsigned char tag,unsigned char pres)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,2880));
    cmd(VERTEX2F(6944,2880));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,160));
    cmd(VERTEX2F(6944,160));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,2880));
    cmd(VERTEX2F(544,160));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(6944,2880));
    cmd(VERTEX2F(6944,160));
    cmd(END());
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(387,209,27, 1536, "Tiempo [min]");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(16,20,31, 1536, "u");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,160,16, 1536, "0.000");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,143,16, 1536, "0.125");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,126,16, 1536, "0.250");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,109,16, 1536, "0.375");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,92,16, 1536, "0.500");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(458,75,16, 1536, "0.625");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
```

```
cmd_text(458,58,16, 1536, "0.750");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,41,16, 1536, "0.875");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,24,16, 1536, "1.000");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(54,191,16, 1536, "0.0");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(94,191,16, 1536, t1);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(134,191,16, 1536, t2);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(174,191,16, 1536, t3);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(214,191,16, 1536, t4);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(254,191,16, 1536, t5);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(294,191,16, 1536, t6);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(334,191,16, 1536, t7);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(374,191,16, 1536, t8);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(414,191,16, 1536, t9);
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
if (tag!=25)
{
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x003870);
cmd_gradcolor(0xffffff);
cmd(TAG(25));
if (pres==25) cmd_button(340,229, 120,34, 27,OPT_FLAT, "");
else cmd_button(340,229, 120,34, 27,0, "Arrancar");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(LINE_WIDTH(31));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(144,224));
cmd(VERTEX2F(160,704));
cmd(END());
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_fgcolor(0x003870);
cmd_gradcolor(0xffffff);
cmd(TAG(26));
```

```
    if (pres==26) cmd_button(195,229, 120,34, 27,OPT_FLAT, "");
    else cmd_button(195,229, 120,34, 27,0, "Volver");
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,255,0));
    cmd_text(230,77,27, 1536, "Presione Arrancar y luego coloque la carga");
    }
    else
    {
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(27));
    if (pres==27) cmd_button(340,229, 120,34, 27,0, "");
    else cmd_button(340,229, 120,34, 27,0, "Detener");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(230,77,27, 1536, "El ensayo empieza una vez colocada la carga");
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_spinner(230,128,0,0);
    }
    cmd(RESTORE_CONTEXT());
    finpantalla();
}
void PantallaErrorSD()
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(170,0,0));
    cmd_text(240,45,31, 1536, "Error");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(240,90,28, 1536, "Ha ocurrido un error con la tarjeta SD,");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(40,160,27, 1024, "* Presione Continuar si desea proceder con el ensayo.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(240,113,28, 1536, "los datos no se guardaran.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(40,184,27, 1024, "* Presione Salir para volver al inicio.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(125));
    cmd_button(50,212, 140,50, 29,0, "Salir");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(126));
    cmd_button(290,212, 140,50, 29,0, "Continuar");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}
void PantallaEspera6()
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
```

```
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(113,170,165));
    cmd_bgcolor(0x000000);
    cmd_progress(140,180,200,30,256,contador_calibracion,80);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,170,0));
    cmd_text(240,60,29, 1536, "Por favor espere hasta que ");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,170,0));
    cmd_text(240,95,29, 1536, "se calibre el sistema");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void pantallaEnsayando ()   // grafico
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,2880));
    cmd(VERTEX2F(6944,2880));
    cmd(END());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,160));
    cmd(VERTEX2F(6944,160));
    cmd(END());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,2880));
    cmd(VERTEX2F(544,160));
    cmd(END());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(6944,2880));
    cmd(VERTEX2F(6944,160));
    cmd(END());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(387,209,27, 1536, "Tiempo [min]");
    cmd(RESTORE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(16,20,31, 1536, "u");
    cmd(LINE_WIDTH(4));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,432));
    cmd(VERTEX2F(6944,432));
    cmd(END());
    cmd(LINE_WIDTH(4));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,704));
    cmd(VERTEX2F(6944,704));
    cmd(END());
    cmd(LINE_WIDTH(4));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(544,976));
    cmd(VERTEX2F(6944,976));
    cmd(END());
```

```
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,1248));
cmd(VERTEX2F(6944,1248));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,1520));
cmd(VERTEX2F(6944,1520));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,1792));
cmd(VERTEX2F(6944,1792));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,2064));
cmd(VERTEX2F(6944,2064));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,2336));
cmd(VERTEX2F(6944,2336));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(544,2608));
cmd(VERTEX2F(6944,2608));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(864,160));
cmd(VERTEX2F(864,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(1184,160));
cmd(VERTEX2F(1184,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(1504,160));
cmd(VERTEX2F(1504,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(1824,160));
cmd(VERTEX2F(1824,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(2144,160));
cmd(VERTEX2F(2144,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(2784,160));
cmd(VERTEX2F(2784,2880));
cmd(END());
cmd(LINE_WIDTH(4));
```

```
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(2464,160));
cmd(VERTEX2F(2464,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(3104,160));
cmd(VERTEX2F(3104,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(3424,160));
cmd(VERTEX2F(3424,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(3744,160));
cmd(VERTEX2F(3744,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(4064,160));
cmd(VERTEX2F(4064,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(4384,160));
cmd(VERTEX2F(4384,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(4704,160));
cmd(VERTEX2F(4704,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(5024,160));
cmd(VERTEX2F(5024,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(5344,160));
cmd(VERTEX2F(5344,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(5664,160));
cmd(VERTEX2F(5664,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(5984,160));
cmd(VERTEX2F(5984,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
cmd(BEGIN(LINES));
cmd(VERTEX2F(6304,160));
cmd(VERTEX2F(6304,2880));
cmd(END());
cmd(LINE_WIDTH(4));
cmd(COLOR_RGB(255,255,255));
```

```
cmd(BEGIN(LINES));
cmd(VERTEX2F(6624,160));
cmd(VERTEX2F(6624,2880));
cmd(END());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,160,16, 1536, "0.000");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,143,16, 1536, "0.125");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,126,16, 1536, "0.250");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,109,16, 1536, "0.375");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,92,16, 1536, "0.500");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,75,16, 1536, "0.625");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,58,16, 1536, "0.750");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,41,16, 1536, "0.875");
cmd(RESTORE_CONTEXT());
cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(458,24,16, 1536, "1.000");
cmd(RESTORE_CONTEXT());
//cmd(SAVE_CONTEXT());
cmd(COLOR_RGB(255,255,255));
cmd_text(54,191,16, 1536, "0.0");
cmd(COLOR_RGB(255,255,255));
cmd_text(94,191,16, 1536, t1);
cmd(COLOR_RGB(255,255,255));
cmd_text(134,191,16, 1536, t2);
cmd(COLOR_RGB(255,255,255));
cmd_text(174,191,16, 1536, t3);
cmd(COLOR_RGB(255,255,255));
cmd_text(214,191,16, 1536, t4);
cmd(COLOR_RGB(255,255,255));
cmd_text(254,191,16, 1536, t5);
cmd(COLOR_RGB(255,255,255));
cmd_text(294,191,16, 1536, t6);
cmd(COLOR_RGB(255,255,255));
cmd_text(334,191,16, 1536, t7);
cmd(COLOR_RGB(255,255,255));
cmd_text(374,191,16, 1536, t8);
cmd(COLOR_RGB(255,255,255));
cmd_text(414,191,16, 1536, t9);
if (numero_pulsos==0)
{
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(29));
    cmd_button(340,229, 120,34, 27,0, "OK");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,0,0));
    cmd_text(160,240,29,1536, "El ensayo ha finalizado");
```

```
    }
    else
    {
        cmd(COLOR_RGB(255,255,255));
        cmd_fgcolor(0x003870);
        cmd_gradcolor(0xffffff);
        cmd(TAG(28));
        cmd_button(340,229, 120,34, 27,0, "Detener");
    }
    cmd(RESTORE_CONTEXT());
    cmd(LINE_WIDTH(31));
    cmd(COLOR_RGB(255,255,255));
    cmd(BEGIN(LINES));
    cmd(VERTEX2F(144,224));
    cmd(VERTEX2F(160,704));
    cmd(END());
    cmd(SAVE_CONTEXT());
    cmd(LINE_WIDTH(16));
    cmd(COLOR_RGB(0,255,0));
    cmd(BEGIN(LINES));
    if (numeropuntos > 0)
    {
        for (int j = 1; j<=numeropuntos; j++)
        {
            cmd(VERTEX2F(xpan[j-1],ypan[j-1]));
            cmd(VERTEX2F(xpan[j],ypan[j]));
        }
    }
    cmd(END());
    cmd(RESTORE_CONTEXT());
    finpantalla();
}


void pantallaStandby(void)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,0,0));
    cmd_text(240,64,29, 1536, "El ensayo se ha detenido");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(240,105,28, 1536, "Coloque nuevamente la carga para continuar");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(240,141,28, 1536, "Presione Salir para finalizar el ensayo");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x550000);
    cmd_gradcolor(0xffffff);
    cmd(TAG(34));
    cmd_button(160,178, 160,60, 31,0, "Salir");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}

void Ensayando_detenido(void)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(170,0,0));
    cmd_text(240,65,30, 1536, "Ha detenido el ensayo");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
```

```
    cmd(COLOR_RGB(255,255,255));
    cmd_text(50,140,27, 1024, "*Presione Continuar para reanudar.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_text(50,165,27, 1024, "*Presione Salir para terminar el ensayo.");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(130));
    cmd_button(40,200, 160,50, 29,0, "Salir");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(131));
    cmd_button(280,200, 160,50, 28,0, "Continuar");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}
void pantallaRectificacion(unsigned char tag)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,255));
    cmd_text(240,45,29, 1536, "Rectificacion");
    cmd(RESTORE_CONTEXT());
    if (tag!=31)
    {
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(255,255,255));
        cmd_fgcolor(0x2e7f76);
        cmd_gradcolor(0xffffff);
        cmd(TAG(30));
        if (tag==30) cmd_button(130,80, 110,55, 28,OPT_FLAT, "");
        else cmd_button(130,80, 110,55, 28,0, "Frecuencia");
        cmd(RESTORE_CONTEXT());
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(255,255,255));
        cmd(LINE_WIDTH(16));
        cmd(BEGIN(RECTS));
        cmd(VERTEX2F(3840,1280));
        cmd(VERTEX2F(5600,2160));
        cmd(END());
        cmd(RESTORE_CONTEXT());
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(255,255,255));
        cmd_fgcolor(0x003870);
        cmd_gradcolor(0xffffff);
        cmd(TAG(33));
        if (tag==33) cmd_button(50,190, 160,60, 29,OPT_FLAT, "");
        else cmd_button(50,190, 160,60, 29,0, "Volver");
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(255,255,255));
        cmd_fgcolor(0x003870);
        cmd_gradcolor(0xffffff);
        cmd(TAG(31));
        if (tag==31)cmd_button(270,190, 160,60, 29,OPT_FLAT, "");
        else cmd_button(270,190, 160,60, 29,0, "Arrancar");
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(0,0,0));
        cmd_text(295,107,28, 1536, frecuencia_char);
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(0,0,0));
```

```
        cmd_text(333,128,27, 1536, "RPM");
        cmd(RESTORE_CONTEXT());
     }
    else
    {
        HAL_Delay(100);
        cmd(SAVE_CONTEXT());
        cmd(COLOR_RGB(255,255,255));
        cmd_fgcolor(0x003870);
        cmd_gradcolor(0xffffff);
        cmd(TAG(32));
        if (tag==32)cmd_button(160,190, 160,60, 29,OPT_FLAT, "");
        else cmd_button(160,190, 160,60, 29,0, "Detener");
        cmd(RESTORE_CONTEXT());
        cmd(COLOR_RGB(46,127,118));
        cmd_spinner(240,125,0,0);

    }

    cmd(RESTORE_CONTEXT());
    finpantalla();

}
void ErrorRectificacion(void)
{
    iniciopantalla();
    cmd_dlstart();
    cmd(CLEAR_COLOR_RGB(0,0,0));
    cmd(CLEAR(1,1,1));
    cmd(SAVE_CONTEXT());
    cmd_gradient(239,65, 0xb3b3b3, 240,226, 0xffffff);
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(170,0,0));
    cmd_text(240,50,30, 1536, "Error de Configuracion");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,0));
    cmd_text(240,120,28, 1536, "*Rango de velocidad permitido:");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(0,0,127));
    cmd_text(240,155,28, 1536, "50 RPM < Velocidad < 350 RPM");
    cmd(RESTORE_CONTEXT());
    cmd(SAVE_CONTEXT());
    cmd(COLOR_RGB(255,255,255));
    cmd_fgcolor(0x003870);
    cmd_gradcolor(0xffffff);
    cmd(TAG(140));
    cmd_button(160,190, 160,50, 30,0, "OK");
    cmd(RESTORE_CONTEXT());
    finpantalla();
}
void habilitacion_dac_grafico  (void)
{
    HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
    registrodac = -372.5 + 12.41*frecuencia;        // Esta funcion permite setear una
    velocidad desde 50 RPM a 350RPM
    registrodac = round(registrodac);
    HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1,DAC_ALIGN_12B_R,registrodac); // seteo referencia
    graficocada = (60.0*duracion)/288.0;    // cada cuanto tiempo tengo que graficar
    intervalografico = graficocada*100.0;       // cada cuantas interrupciones grafico
    intervalograficoint = round(intervalografico);  // redondeo intervalografico
    deltat = duracion/18.0; // divido el eje temporal
    sprintf(t1, "%.1f",deltat*2);
    sprintf(t2, "%.1f",deltat*4);
    sprintf(t3, "%.1f",deltat*6);
    sprintf(t4, "%.1f",deltat*8);
    sprintf(t5, "%.1f",deltat*10);
    sprintf(t6, "%.1f",deltat*12);
    sprintf(t7, "%.1f",deltat*14);
    sprintf(t8, "%.1f",deltat*16);
    sprintf(t9, "%.1f",deltat*18);
```

```
}

void iniciopantalla (void)
{
    do
    {
        cmdBufferRd = ft800memRead16(REG_CMD_READ);     // Read the graphics processor read
        pointer
        cmdBufferWr = ft800memRead16(REG_CMD_WRITE);    // Read the graphics processor write
        pointer
    }while (cmdBufferWr != cmdBufferRd);                // Wait until the two registers match


}

void finpantalla (void)
{
        cmd(DISPLAY());
        cmd_swap();
        ft800memWrite16(REG_CMD_WRITE, cli);            // Update the ring buffer pointer so
        the graphics processor starts executing
}

//****************************************END
FILE*************************************************//
```

```
/**
  ******************************************************************************
  * File Name          : pantallas.h
  * Description        : pantallas utilizadas
  ******************************************************************************
  */
void calibrate (void);  // calibracion
void PantallaInicio (unsigned char pres);   // menu principal
void PantallaTipicos(void);
void PantallaSeteo (unsigned char pres);    // seteo
void TecladoVariables (char variable_char[10], unsigned char pres); // teclado
void pantallaEnsayando ();  // grafico
void ErrorSeteo (void); // error de configuracion
float FuncionFloat (char variable_char[10], float variable);    // funcion para leer el
teclado
void habilitacion_dac_grafico (void);
void iniciopantalla (void);
void finpantalla (void);
void TecladoABC (unsigned char pres);
void Teclado123 (unsigned char pres);
void SiDeseaGuardar(unsigned char pres);
void NoIngresoSD(unsigned char pres);
void Inicio_ensayo(unsigned char tag,unsigned char pres);
void pantallaRectificacion(unsigned char tag);
void pantallaStandby(void);
void PantallaSeteoFlash(unsigned char pres);
void PantallaEspera6(void);
void PantallaErrorSD(void);
```

```
 /**
  ************************************************************************
  * File Name          : Funciones propias.c
  * Description        : carga los valores ingresados en el teclado en la variable
  correspondiente
  ************************************************************************
**/
#include "pantallas.h"
#include "Funciones propias.h"
#include "main.h"
#include "global.h"
#include "ft_gpu.h"
#include "ft800.h"
#include "co_processor.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

/*declaracion de funciones-----------------------------------------------------*/
float CargoValor (char tipo_char[10],float tipo,unsigned char tag);

/*variables--------------------------------------------------------------------*/
char tecla_var[1];
extern int  longitudstring, comparostring;

/*-----------------------------------------------------------------------------*/


float CargoValor (char tipo_char[10],float tipo,unsigned char tag)
{

            if (tag==101)
            {
                tipo_char[strlen(tipo_char)-1]='\0';  //borro un caracter Del
            }
            if (tag==8)
            {
                strcpy(tipo_char,"0");  // Si toco clear borro frecuencia Clr
            }
            if (tag>45 && tag<58)
            {
                sprintf(tecla_var, "%c",tag); //a tecla le paso el valor tag con formato
                longitudstring = strlen(tipo_char);
                if (longitudstring < 9) // Si frecuencia tiene menos de 9 caracteres sigo
                escribiendo
                {
                 comparostring = strcmp(tipo_char, "0");
                    if (comparostring == 0) // Si frecuencia es 0 reemplazo el 0 por el
                    numero que se ingreso, no concateno
                        strcpy(tipo_char,tecla_var);
                    else
                      strcat(tipo_char , tecla_var);
                }
             }
        tipo = atof(tipo_char);
        return tipo;
}


//****END FILE****//
```

```
/**
  ******************************************************************************
  * File Name          : Funciones propias.h
  * Description        : funciones para cargar en variables
  ******************************************************************************
  */

float CargoValor (char tipo_char[10],float tipo,unsigned char tag);

/*sd-----------------------------------*/
void enviar_flotante(float f);
void SDCard_Config(void);
void Inicio_SD(void);


void cerrar_SD(void);

/*-------------------------------------*/

void restablecer_variables(void);
```

```c
/**
  ******************************************************************************
  * File Name          : ADC.c
  * Description        : This file provides code for the configuration
  *                      of the ADC instances.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "adc.h"
#include "gpio.h"

ADC_HandleTypeDef hadc1;

/* ADC1 init function */
void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and
    number of conversion)
    */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV8;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

    /**Configure for the selected ADC regular channel its corresponding rank in the
    sequencer and its sample time.
    */
  sConfig.Channel = ADC_CHANNEL_7;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

void HAL_ADC_MspInit(ADC_HandleTypeDef* adcHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspInit 0 */

  /* USER CODE END ADC1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();

    /**ADC1 GPIO Configuration
    PA7     ------> ADC1_IN7
    */
    GPIO_InitStruct.Pin = GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* Peripheral interrupt init */
```

```
    HAL_NVIC_SetPriority(ADC_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(ADC_IRQn);
  /* USER CODE BEGIN ADC1_MspInit 1 */

  /* USER CODE END ADC1_MspInit 1 */
  }
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* adcHandle)
{

  if(adcHandle->Instance==ADC1)
  {
  /* USER CODE BEGIN ADC1_MspDeInit 0 */

  /* USER CODE END ADC1_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();

    /**ADC1 GPIO Configuration
    PA7     ------> ADC1_IN7
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_7);

    /* Peripheral interrupt Deinit*/
    HAL_NVIC_DisableIRQ(ADC_IRQn);

  }

}
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : ADC.h
  * Description        : This file provides code for the configuration
  *                      of the ADC instances.
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __adc_H
#define __adc_H
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "main.h"


extern ADC_HandleTypeDef hadc1;
extern void Error_Handler(void);

void MX_ADC1_Init(void);



#ifdef __cplusplus
}
#endif
#endif /*__ adc_H */



/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : co_processor.c
  * Description        : ft800 funciones para graficar los widget de una.
  ******************************************************************************
  **/
#include "co_processor.h"
#include "ft_gpu.h"
#include "ft800.h"

#include <string.h>

unsigned int cli;   // Habia una variable mas, dli, que estaba al pedo para mi

void write_string(const char *s)
{                                                       // write string

    uint16_t length, n;
    length = strlen(s) + 1;

    for(n = 0; n < length; n++) {

        ft800memWrite8(RAM_CMD + cli + n, s[n]);
    }

    ft800memWrite8(RAM_CMD + cli + n, 0);
    //cli += ((length + 3) & ~3);
    cli = incCMDOffset(cli, ((length + 3) & ~3));
}

void cmd(uint32_t cmd) {
// write 32bit command to co-processor engine FIFO RAM_CMD

    ft800memWrite32(RAM_CMD + cli, cmd);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_dlstart(void) {
// start a new display list

    ft800memWrite32(RAM_CMD + cli, CMD_DLSTART);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_swap(void) {
// swap the current display list

    ft800memWrite32(RAM_CMD + cli, CMD_SWAP);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_coldstart(void) {
// set co-processor engine state to default values

    ft800memWrite32(RAM_CMD + cli, CMD_COLDSTART);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_interrupt(uint32_t ms) {
// trigger interrupt INT_CMDFLAG

    ft800memWrite32(RAM_CMD + cli, CMD_INTERRUPT);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ms);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}
```

```
void cmd_append(uint32_t ptr, uint32_t num) {
// append memory to display list

    ft800memWrite32(RAM_CMD + cli, CMD_APPEND);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_regread(uint32_t ptr, uint32_t result) {
// read a register value

    ft800memWrite32(RAM_CMD + cli, CMD_REGREAD);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, result);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_memwrite(uint32_t ptr, uint32_t num) {
// write bytes into memory

    ft800memWrite32(RAM_CMD + cli, CMD_MEMWRITE);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
}

void cmd_inflate(uint32_t ptr) {
// decompress data into memory

    ft800memWrite32(RAM_CMD + cli, CMD_INFLATE);
    //cli += 4;
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
}

void cmd_loadimage(const uint8_t* img, uint32_t ptr, uint32_t length, uint32_t opt) {
//load a JPEG image

    uint32_t i;

    ft800memWrite32(RAM_CMD + cli, CMD_LOADIMAGE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, opt);
    cli = incCMDOffset(cli, 4);

    for (i = 0; i < length; i++)
    {
        ft800memWrite8(RAM_CMD + cli, *(img + i));
        cli++;
    }
}

void cmd_memcrc(uint32_t ptr, uint32_t num, uint32_t result) {
//compute a CRC-32 for memory
```

```
    ft800memWrite32(RAM_CMD + cli, CMD_MEMCRC);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, result);
    cli = incCMDOffset(cli, 4);

}

void cmd_memzero(uint32_t ptr, uint32_t num) {
//write zero to a block of memory

    ft800memWrite32(RAM_CMD + cli, CMD_MEMZERO);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    cli = incCMDOffset(cli, 4);
}

void cmd_memset(uint32_t ptr, uint32_t value, uint32_t num) {
//fill memory with a byte value

    ft800memWrite32(RAM_CMD + cli, CMD_MEMSET);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, value);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    cli = incCMDOffset(cli, 4);
}

void cmd_memcpy(uint32_t dest, uint32_t src, uint32_t num) {
//copy a block of memory

    ft800memWrite32(RAM_CMD + cli, CMD_MEMCPY);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, dest);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, src);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, num);
    cli = incCMDOffset(cli, 4);
}

void cmd_button(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, int16_t opt, const
char *s) {             //draw a button

    ft800memWrite32(RAM_CMD + cli, CMD_BUTTON);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | font));
    cli = incCMDOffset(cli, 4);
    write_string(s);
}

void cmd_clock(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t h, uint16_t m,
uint16_t s, uint16_t ms) {    //draw an analog clock

    ft800memWrite32(RAM_CMD + cli, CMD_CLOCK);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, (y << 16) | x);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | r));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((m << 16) | h));
```

```
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((ms << 16) | s));
    cli = incCMDOffset(cli, 4);
}

void cmd_fgcolor(uint32_t color) {
// set the foreground color

    ft800memWrite32(RAM_CMD + cli, CMD_FGCOLOR);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, color);
    cli = incCMDOffset(cli, 4);
}

void cmd_bgcolor(uint32_t color) {
// set the backgroupd color

    ft800memWrite32(RAM_CMD + cli, CMD_BGCOLOR);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, color);
    cli = incCMDOffset(cli, 4);
}

void cmd_gradcolor(uint32_t color) {
// set the 3D button highlight color

    ft800memWrite32(RAM_CMD + cli, CMD_GRADCOLOR);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, color);
    cli = incCMDOffset(cli, 4);
}

void cmd_gauge(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t major, uint16_t
minor, uint16_t val, uint16_t range) {   // draw a gauge

    ft800memWrite32(RAM_CMD + cli, CMD_GAUGE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, (y << 16) | x);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | r));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((minor << 16) | major));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((range << 16) | val));
    cli = incCMDOffset(cli, 4);

}

void cmd_gradient(int16_t x0, int16_t y0, uint32_t rgb0, int16_t x1, int16_t y1, uint32_t
rgb1) {       // draw a smooth color gradient

    ft800memWrite32(RAM_CMD + cli, CMD_GRADIENT);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y0 << 16) | x0));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, rgb0);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y1 << 16) | x1));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, rgb1);
    cli = incCMDOffset(cli, 4);
}

void cmd_keys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, uint16_t opt, const
char* s) {      // draw a row of keys

    ft800memWrite32(RAM_CMD + cli, CMD_KEYS);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | font));
```

```
    cli = incCMDOffset(cli, 4);
    write_string(s);
}

void cmd_progress(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t range) {     // draw a progress bar

    ft800memWrite32(RAM_CMD + cli, CMD_PROGRESS);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((val << 16) | opt));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, range);
    cli = incCMDOffset(cli, 4);
}

void cmd_scrollbar(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t size, uint16_t range) { // draw a scroll bar

    ft800memWrite32(RAM_CMD + cli, CMD_SCROLLBAR);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((val << 16) | opt));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((range << 16) | size));;
    cli = incCMDOffset(cli, 4);
}

void cmd_slider(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t range) {                    // draw a slider

    ft800memWrite32(RAM_CMD + cli, CMD_SLIDER);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((val << 16) | opt));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, range);
    cli = incCMDOffset(cli, 4);
}

void cmd_dial(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t val)
{                                                 // draw a rotary dial control

    ft800memWrite32(RAM_CMD + cli, CMD_DIAL);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | r));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, val);
    cli = incCMDOffset(cli, 4);
}

void cmd_toggle(int16_t x, int16_t y, int16_t w, int16_t font, uint16_t opt, uint16_t state,
const char* s) {                // draw a toggle switch

    ft800memWrite32(RAM_CMD + cli, CMD_TOGGLE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((font << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((state << 16) | opt));
    cli = incCMDOffset(cli, 4);
```

```
    write_string(s);
}

void cmd_text(int16_t x, int16_t y, int16_t font, uint16_t opt, const char* s)
{                                               // draw text

    ft800memWrite32(RAM_CMD + cli, CMD_TEXT);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | font));
    cli = incCMDOffset(cli, 4);
    write_string(s);
}

void cmd_number(int16_t x, int16_t y, int16_t font, uint16_t opt, int32_t n)
{                                                // draw a decimal number

    ft800memWrite32(RAM_CMD + cli, CMD_NUMBER);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((opt << 16) | font));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, n);
    cli = incCMDOffset(cli, 4);
}

void cmd_loadidentity(void)
{
    // set the current matrix to the identity matrix

    ft800memWrite32(RAM_CMD + cli, CMD_LOADIDENTITY);
    cli = incCMDOffset(cli, 4);
}

void cmd_setmatrix(void)
{
        // write the current matrix to the display list

    ft800memWrite32(RAM_CMD + cli, CMD_SETMATRIX);
    cli = incCMDOffset(cli, 4);
}

void cmd_getmatrix(int32_t a, int32_t b, int32_t c, int32_t d, int32_t e, int32_t f)
{                                          // retrieves the current matrix coefficients

    ft800memWrite32(RAM_CMD + cli, CMD_GETMATRIX);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, a);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, b);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, c);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, d);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, e);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, f);
    cli = incCMDOffset(cli, 4);
}

void cmd_getptr(uint32_t result)
{
// get the end memory address of inflated data

    ft800memWrite32(RAM_CMD + cli, CMD_GETPTR);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, result);
    cli = incCMDOffset(cli, 4);
}
```

```
void cmd_getprops(uint32_t ptr, uint32_t width, uint32_t height)
{                                                       // get the image properties
decompressed by CMD_LOADIMAGE

    ft800memWrite32(RAM_CMD + cli, CMD_GETPROPS);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, width);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, height);
    cli = incCMDOffset(cli, 4);
}

void cmd_scale(int32_t sx, int32_t sy)
{                                                                                   //
apply a scale to the current matrix

    ft800memWrite32(RAM_CMD + cli, CMD_SCALE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, sx);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, sy);
    cli = incCMDOffset(cli, 4);
}

void cmd_rotate(int32_t a)
{
    // apply a rotation to the current matrix

    ft800memWrite32(RAM_CMD + cli, CMD_ROTATE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, a);
    cli = incCMDOffset(cli, 4);
}

void cmd_translate(int32_t tx, int32_t ty)
{                                                                               // apply a
translation to the current matrix

    ft800memWrite32(RAM_CMD + cli, CMD_TRANSLATE);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, tx);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ty);
    cli = incCMDOffset(cli, 4);
}

void cmd_calibrate(void)
{                                                                                    //
execute the screen calibration routine

    ft800memWrite32(RAM_CMD + cli, CMD_CALIBRATE);
    cli = incCMDOffset(cli, 4);
}

void cmd_spinner(int16_t x, int16_t y, uint16_t style, uint16_t scale)
{                                       // start an animated spinner

    ft800memWrite32(RAM_CMD + cli, CMD_SPINNER);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((scale << 16) | style));
    cli = incCMDOffset(cli, 4);
}

void cmd_screensaver(void)
{                                                                                   //
start an animated screensaver

    ft800memWrite32(RAM_CMD + cli, CMD_SCREENSAVER);
    cli = incCMDOffset(cli, 4);
```

```
}

void cmd_sketch(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t ptr, uint16_t format)
{                    // start continuous sketch update

    ft800memWrite32(RAM_CMD + cli, CMD_SKETCH);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, format);
    cli = incCMDOffset(cli, 4);
}

void cmd_stop(void)
{
    // stop any of spinner, screensaver or sketch

    ft800memWrite32(RAM_CMD + cli, CMD_STOP);
    cli = incCMDOffset(cli, 4);
}

void cmd_setfont(uint32_t font, uint32_t ptr)
{                                                                    // set up a custom font

    ft800memWrite32(RAM_CMD + cli, CMD_SETFONT);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
}

void cmd_track(int16_t x, int16_t y, int16_t w, int16_t h, int16_t tag)
{                                          // track touches for a graphics object

    ft800memWrite32(RAM_CMD + cli, CMD_TRACK);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((y << 16) | x));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ((h << 16) | w));
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, tag);
    cli = incCMDOffset(cli, 4);
}

void cmd_snapshot(uint32_t ptr)
{                                                                                  // take
a snapshot of the current screen

    ft800memWrite32(RAM_CMD + cli, CMD_SNAPSHOT);
    cli = incCMDOffset(cli, 4);
    ft800memWrite32(RAM_CMD + cli, ptr);
    cli = incCMDOffset(cli, 4);
}

void cmd_logo(void)
{
    // play FTDI logo animation

    ft800memWrite32(RAM_CMD + cli, CMD_LOGO);
    cli = incCMDOffset(cli, 4);
}

/***END FILE***/
```

```
/**
  ******************************************************************************
  * File Name          : co_processor.h
  * Description        : funciones de co_processor
  ******************************************************************************
  */
#ifndef __co_processor_H
#define __eve_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "ft_gpu.h"
#include "ft800.h"

extern unsigned int dli, cli;

void write_string(const char *s);
void cmd(uint32_t cmd);
void cmd_dlstart(void);
void cmd_swap(void);
void cmd_coldstart(void);
void cmd_interrupt(uint32_t ms);
void cmd_append(uint32_t ptr, uint32_t num);
void cmd_regread(uint32_t ptr, uint32_t result);
void cmd_memwrite(uint32_t ptr, uint32_t num);
void cmd_inflate(uint32_t ptr);
void cmd_loadimage(const uint8_t* img, uint32_t ptr, uint32_t length, uint32_t opt);
void cmd_memcrc(uint32_t ptr, uint32_t num, uint32_t result);
void cmd_memzero(uint32_t ptr, uint32_t num);
void cmd_memset(uint32_t ptr, uint32_t value, uint32_t num);
void cmd_memcpy(uint32_t dest, uint32_t src, uint32_t num);
void cmd_button(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, int16_t opt, const
char*s);
void cmd_clock(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t h, uint16_t m, uint16_t
s, uint16_t ms);
void cmd_fgcolor(uint32_t color);
void cmd_bgcolor(uint32_t color);
void cmd_gradcolor(uint32_t color);
void cmd_gauge(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t major, uint16_t minor,
uint16_t val, uint16_t range);
void cmd_gradient(int16_t x0, int16_t y0, uint32_t rgb0, int16_t x1, int16_t y1, uint32_t rgb1);
void cmd_keys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font, uint16_t opt, const
char* s);
void cmd_progress(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t range);
void cmd_scrollbar(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t size, uint16_t range);
void cmd_slider(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t opt, uint16_t val,
uint16_t range);
void cmd_dial(int16_t x, int16_t y, int16_t r, uint16_t opt, uint16_t val);
void cmd_toggle(int16_t x, int16_t y, int16_t w, int16_t font, uint16_t opt, uint16_t state,
const char* s);
void cmd_text(int16_t x, int16_t y, int16_t font, uint16_t opt, const char* s);
void cmd_number(int16_t x, int16_t y, int16_t font, uint16_t opt, int32_t n);
void cmd_loadidentity(void);
void cmd_setmatrix(void);
void cmd_getmatrix(int32_t a, int32_t b, int32_t c, int32_t d, int32_t e, int32_t f);
void cmd_getptr(uint32_t result);
void cmd_getprops(uint32_t ptr, uint32_t width, uint32_t height);
void cmd_scale(int32_t sx, int32_t sy);
void cmd_rotate(int32_t a);
void cmd_translate(int32_t tx, int32_t ty);
void cmd_calibrate(void);
void cmd_spinner(int16_t x, int16_t y, uint16_t style, uint16_t scale);
void cmd_screensaver(void);
void cmd_sketch(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t ptr, uint16_t format);
void cmd_stop(void);
void cmd_setfont(uint32_t font, uint32_t ptr);
void cmd_track(int16_t x, int16_t y, int16_t w, int16_t h, int16_t tag);
```

```
void cmd_snapshot(uint32_t ptr);
void cmd_logo(void);

#endif /*__co_processor_H */
```

```c
/**
  ******************************************************************************
  * File Name          : DAC.c
  * Description        : This file provides code for the configuration
  *                      of the DAC instances.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "dac.h"
#include "gpio.h"

DAC_HandleTypeDef hdac;

/* DAC init function */
void MX_DAC_Init(void)
{
  DAC_ChannelConfTypeDef sConfig;

    /**DAC Initialization
    */
  hdac.Instance = DAC;
  if (HAL_DAC_Init(&hdac) != HAL_OK)
  {
    Error_Handler();
  }

    /**DAC channel OUT1 config
    */
  sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }

}

void HAL_DAC_MspInit(DAC_HandleTypeDef* dacHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(dacHandle->Instance==DAC)
  {
    /* Peripheral clock enable */
    __HAL_RCC_DAC_CLK_ENABLE();

    /**DAC GPIO Configuration
    PA5     ------> DAC_OUT1
    */
    GPIO_InitStruct.Pin = GPIO_PIN_5;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  }
}

void HAL_DAC_MspDeInit(DAC_HandleTypeDef* dacHandle)
{

  if(dacHandle->Instance==DAC)
  {
    /* Peripheral clock disable */
    __HAL_RCC_DAC_CLK_DISABLE();

    /**DAC GPIO Configuration
    PA5     ------> DAC_OUT1
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_5);
  }

}
```

```
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : DAC.h
  * Description        : This file provides code for the configuration
  *                      of the DAC instances.
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __dac_H
#define __dac_H
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "main.h"

extern DAC_HandleTypeDef hdac;

extern void Error_Handler(void);

void MX_DAC_Init(void);


#ifdef __cplusplus
}
#endif
#endif /*__ dac_H */


/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : ft800.c
  * Description        : configuración del ft800
  ******************************************************************************
  **/
#include "spi.h"
#include "gpio.h"
#include "ft_gpu.h"
#include "ft800.h"

unsigned int lcdWidth;      // Active width of LCD display
unsigned int lcdHeight;     // Active height of LCD display
unsigned int lcdHcycle;     // Total number of clocks per line
unsigned int lcdHoffset;    // Start of active line
unsigned int lcdHsync0;     // Start of horizontal sync pulse
unsigned int lcdHsync1;     // End of horizontal sync pulse
unsigned int lcdVcycle;     // Total number of lines per screen
unsigned int lcdVoffset;    // Start of active screen
unsigned int lcdVsync0;     // Start of vertical sync pulse
unsigned int lcdVsync1;     // End of vertical sync pulse
unsigned char lcdPclk;      // Pixel Clock
unsigned char lcdSwizzle;   // Define RGB output pins
unsigned char lcdPclkpol;   // Define active edge of PCLK

unsigned long ramDisplayList = RAM_DL;      // Set beginning of display list memory
unsigned long ramCommandBuffer = RAM_CMD;   // Set beginning of graphics command memory

unsigned char ft800Gpio;    // Used for FT800 GPIO register

void ft800cmdWrite(unsigned char ftCommand)
{
    unsigned char cZero = 0x00;                                         // Filler value for
    command
    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low
    HAL_SPI_Transmit(&hspi5, &ftCommand, 1, 0);                         // Send command
    HAL_SPI_Transmit(&hspi5, &cZero, 1, 0);                             // Send first filler
    byte
    HAL_SPI_Transmit(&hspi5, &cZero, 1, 0);                             // Send second
    filler byte
    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high
}

void ft800memWrite8(unsigned long ftAddress, unsigned char ftData8)
{
    unsigned char cTempAddr[3];                                         // FT800 Memory
    Address

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_WRITE;                // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                             // middle byte
    cTempAddr[0] = (char) (ftAddress);                                  // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);                  // Send Memory Write
        plus high address byte
    }

    for (int j = 0; j < sizeof(ftData8); j++)
    {
        HAL_SPI_Transmit(&hspi5, &ftData8, 1, 0);                       // Send data byte
    }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high
}

void ft800memWrite16(unsigned long ftAddress, unsigned int ftData16)
{
```

```
    unsigned char cTempAddr[3];                                     // FT800 Memory
    Address
    unsigned char cTempData[2];                                     // 16-bit data to
    write

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_WRITE;            // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                         // middle byte
    cTempAddr[0] = (char) (ftAddress);                              // low byte

    cTempData[1] = (char) (ftData16 >> 8);                          // Compose data to
    be sent - high byte
    cTempData[0] = (char) (ftData16);                               // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);              // Send Memory Write
        plus high address byte
    }

    for (int j = 0; j < sizeof(cTempData); j++)                     // Start with least
    significant byte
    {
        HAL_SPI_Transmit(&hspi5, &cTempData[j], 1, 0);              // Send data byte
    }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high
}

void ft800memWrite32(unsigned long ftAddress, unsigned long ftData32)
{
    unsigned char cTempAddr[3];                                     // FT800 Memory
    Address
    unsigned char cTempData[4];                                     // 32-bit data to
    write

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_WRITE;            // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                         // middle byte
    cTempAddr[0] = (char) (ftAddress);                              // low byte

    cTempData[3] = (char) (ftData32 >> 24);                         // Compose data to
    be sent - high byte
    cTempData[2] = (char) (ftData32 >> 16);
    cTempData[1] = (char) (ftData32 >> 8);
    cTempData[0] = (char) (ftData32);                               // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);              // Send Memory Write
        plus high address byte
    }

    for (int j = 0; j < sizeof(cTempData); j++)                     // Start with least
    significant byte
    {
        HAL_SPI_Transmit(&hspi5, &cTempData[j], 1, 0);              // Send SPI byte
    }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high
}

unsigned char ft800memRead8(unsigned long ftAddress)
{
  unsigned char ftData8 = ZERO;// Place-holder for 8-bits being read
  unsigned char errorprimero = ZERO;
  unsigned char cTempAddr[3];                                       // FT800 Memory
```

```
  Address
    unsigned char cZeroFill = ZERO;                                 // Dummy byte

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_READ;              // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                         // middle byte
    cTempAddr[0] = (char) (ftAddress);                              // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);              // Send Memory Write
        plus high address byte

    }

  HAL_SPI_Transmit(&hspi5, &cZeroFill, 1, 0);                       // Send dummy byte
  HAL_SPI_Receive(&hspi5, &errorprimero, 1, 0);
    for (int j = 0; j < sizeof(ftData8); j++)                       // Start with least
    significant byte
        {
        HAL_SPI_Receive(&hspi5, &ftData8, 1, 0);                    // Receive data byte
        }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high

  return ftData8;                                                   // Return 8-bits
}

unsigned int ft800memRead16(unsigned long ftAddress)
{
  unsigned int ftData16;                                            // 16-bits to return
    unsigned char cTempAddr[3];                                     // FT800 Memory
    Address
    unsigned char cTempData[2];                                     // Place-holder for
    16-bits being read
    unsigned char cZeroFill;

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_READ;             // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                         // middle byte
    cTempAddr[0] = (char) (ftAddress);                              // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);              // Send Memory Write
        plus high address byte
    }

  HAL_SPI_Transmit(&hspi5, &cZeroFill, 1, 0);                       // Send dummy byte
  HAL_SPI_Receive(&hspi5, &cTempData[0], 1, 0);
    for (int j = 0; j < sizeof(cTempData); j++)                   // Start with least
    significant byte
    {
        HAL_SPI_Receive(&hspi5, &cTempData[j], 1, 0);               // Receive data byte
    }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high

    ftData16 = (cTempData[1]<< 8) |                                 // Compose value to
    return - high byte
                      (cTempData[0]);                               // low byte

  return ftData16;                                                  // Return 16-bits
}

unsigned long ft800memRead32(unsigned long ftAddress)
{
```

```
  unsigned long ftData32;                                          // 32-bits to return
    unsigned char cTempAddr[3];                                    // FT800 Memory
    Address
    unsigned char cTempData[4];                                    // Place holder for
    32-bits being read
    unsigned char cZeroFill;                                       // Dummy byte

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_READ;            // Compose the
    command and address to send
    cTempAddr[1] = (char) (ftAddress >> 8);                        // middle byte
    cTempAddr[0] = (char) (ftAddress);                             // low byte

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);    // Set chip select low

  for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi5, &cTempAddr[i], 1, 0);             // Send Memory Write
        plus high address byte
    }

  HAL_SPI_Transmit(&hspi5, &cZeroFill, 1, 0);                      // Send dummy byte

    for (int j = 0; j <= sizeof(cTempData); j++)                   // Start with least
    significatn byte
    {
        HAL_SPI_Receive(&hspi5, &cTempData[j], 1, 0);              // Receive data byte
    }

    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_SET);      // Set chip select
    high

    ftData32 = (cTempData[3]<< 24) |                               // Compose value to
    return - high byte
                        (cTempData[2]<< 16) |
                        (cTempData[1]<< 8) |
                        (cTempData[0]);                            // Low byte

  return ftData32;                                                 // Return 32-bits
}

unsigned int incCMDOffset(unsigned int currentOffset, unsigned char commandSize)
{
    unsigned int newOffset;                                        // Used to hold new
    offset
    newOffset = currentOffset + commandSize;                       // Calculate new
    offset
    if(newOffset > 4095)                                           // If new offset
    past boundary...
    {
        newOffset = (newOffset - 4096);                            // ... roll over
        pointer
    }
    return newOffset;                                              // Return new offset
}

void ft800_Init(void) {

#ifdef
LCD_WQVGA                                                                  //
WQVGA display parameters
    lcdWidth   =
    480;                                                                   // Active
    width of LCD display
    lcdHeight  =
    272;                                                                   // Active
    height of LCD display
    lcdHcycle  =
    548;                                                                   // Total
    number of clocks per line
    lcdHoffset =
    43;                                                                    // Start
    of active line
    lcdHsync0  =
```

```
    0;                                                                  // Start
    of horizontal sync pulse
    lcdHsync1  =
    41;                                                                 // End of
    horizontal sync pulse
    lcdVcycle  =
    292;                                                                // Total
    number of lines per screen
    lcdVoffset =
    12;                                                                 // Start
    of active screen
    lcdVsync0  =
    0;                                                                  // Start
    of vertical sync pulse
    lcdVsync1  =
    10;                                                                 // End of
    vertical sync pulse
    lcdPclk    =
    5;                                                                  // Pixel
    Clock
    lcdSwizzle =
    0;                                                                  // Define
    RGB output pins
    lcdPclkpol =
    1;                                                                  // Define
    active edge of PCLK
#endif

    HAL_GPIO_WritePin(PD_PAN_GPIO_Port, PD_PAN_Pin,
    GPIO_PIN_SET);                                  // Initial state of PD_N - high
    HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin,
    GPIO_PIN_SET);                                      // Initial state of SPI CS - high
    HAL_Delay(20);      // Estaba en
    40                                                                          //
    Wait 20ms
    HAL_GPIO_WritePin(PD_PAN_GPIO_Port, PD_PAN_Pin,
    GPIO_PIN_RESET);                                    // Reset FT800
    HAL_Delay(20);      // Estaba en
    40                                                                  // Wait 20ms
    HAL_GPIO_WritePin(PD_PAN_GPIO_Port, PD_PAN_Pin,
    GPIO_PIN_SET);                                  // FT800 is awake
    HAL_Delay(20);      // Estaba en
    40                                                                  // Wait 20ms -
    required


    ft800cmdWrite(FT800_ACTIVE);
        // Start FT800
    HAL_Delay(5);       // Estaba en
    10                                                                      // Give some
    time to process

    ft800cmdWrite(FT800_CLKEXT);
        // Set FT800 for external clock
    HAL_Delay(5);       // Estaba en
    10                                                                  // Give some time
    to process

    ft800cmdWrite(FT800_CLK48M);
        // Set FT800 for 48MHz PLL
    HAL_Delay(5);       // Estaba en 10
    // Give some time to process

    // Now FT800 can accept commands at up to 30MHz clock on SPI bus
    if (ft800memRead8(REG_ID) !=
    0x7C)                                                       // Read ID register - is
    it 0x7C?
    {

      while(1);
          // If we don't get 0x7C, the ineface isn't working - halt with infinite loop
    }
    ft800memWrite8(REG_PCLK,
```

```
ZERO);                                                          // Set PCLK to zero -
don't clock the LCD until later
ft800memWrite8(REG_PWM_DUTY,
ZERO);                                                  // Turn off backlight

// End of Wake-up FT800
ft800memWrite16(REG_HSIZE,
lcdWidth);                                              // active display width
ft800memWrite16(REG_HCYCLE,
lcdHcycle);                                             // total number of clocks
per line, incl front/back porch
ft800memWrite16(REG_HOFFSET,
lcdHoffset);                                            // start of active line
ft800memWrite16(REG_HSYNC0,
lcdHsync0);                                             // start of horizontal
sync pulse
ft800memWrite16(REG_HSYNC1,
lcdHsync1);                                             // end of horizontal sync
pulse
ft800memWrite16(REG_VSIZE,
lcdHeight);                                             // active display height
ft800memWrite16(REG_VCYCLE,
lcdVcycle);                                             // total number of lines
per screen, incl pre/post
ft800memWrite16(REG_VOFFSET,
lcdVoffset);                                            // start of active screen
ft800memWrite16(REG_VSYNC0,
lcdVsync0);                                             // start of vertical sync
pulse
ft800memWrite16(REG_VSYNC1,
lcdVsync1);                                             // end of vertical sync
pulse
ft800memWrite8(REG_SWIZZLE,
lcdSwizzle);                                            // FT800 output to LCD -
pin order
ft800memWrite8(REG_PCLK_POL,
lcdPclkpol);                                            // LCD data is clocked in
on this PCLK edge

// Don't set PCLK yet - wait for just after the first display list
ft800memWrite8(REG_TOUCH_MODE,
3);                                                     // Enable touch
ft800memWrite8(REG_TOUCH_ADC_MODE, 1);
ft800memWrite16(REG_TOUCH_CHARGE, 6000);
ft800memWrite8(REG_TOUCH_SETTLE, 3);
ft800memWrite8(REG_TOUCH_OVERSAMPLE, 15);
ft800memWrite16(REG_TOUCH_RZTHRESH, 0x5000);
//ft800memWrite8(REG_VOL_PB, 0xf);
ft800memWrite8(REG_VOL_SOUND,0xFF); //VOLUMEN SONIDO
ft800memWrite16(REG_SOUND,0x0057);  //CUAL SONIDO SUENA

ramDisplayList =
RAM_DL;                                                                 // start of
Display List
ft800memWrite32(ramDisplayList,
DL_CLEAR_RGB);                                          // Clear Color RGB
00000010 RRRRRRRR GGGGGGGG BBBBBBBB  (R/G/B = Colour values) default zero / black
ramDisplayList +=
4;                                                                      // point to
next location
ft800memWrite32(ramDisplayList, (DL_CLEAR | CLR_COL | CLR_STN |
CLR_TAG));                  // Clear 00100110 -------- -------- -----CST  (C/S/T define
which parameters to clear)
ramDisplayList +=
4;                                                                      // point to
next location
ft800memWrite32(ramDisplayList,
DL_DISPLAY);                                            // DISPLAY command 00000000
00000000 00000000 00000000 (end of display list)

ft800memWrite32(REG_DLSWAP,
DLSWAP_FRAME);                                           // 00000000 00000000
00000000 000000SS  (SS bits define when render occurs)
```

```
    // Nothing is being displayed yet... the pixel clock is still 0x00
    ramDisplayList =
    RAM_DL;                                                                 // Reset
    Display List pointer for next list

    ft800Gpio =
    ft800memRead8(REG_GPIO);                                                    // Read
    the FT800 GPIO register for a read/modify/write operation
    ft800Gpio = ft800Gpio |
    0x80;                                                                 // set bit 7 of
    FT800 GPIO register (DISP) - others are inputs
    ft800memWrite8(REG_GPIO, 0xff);      //PASAR A FF PARA QUE
    SUENE                                                 // Enable the DISP signal to the
    LCD panel
    ft800memWrite8(REG_GPIO_DIR, 0xFF);  //HABILITAR PARA QUE SUENE
    ft800memWrite8(REG_PCLK,
    lcdPclk);                                                             // Now start clocking
    data to the LCD panel
    for(int duty = 0; duty <= 128; duty++)
    {
      ft800memWrite8(REG_PWM_DUTY,
      duty);                                                        // Turn on backlight -
      ramp up slowly to full brighness
      HAL_Delay(10);
    }

    // End of Write Initial Display List & Enable Display
}
/****END FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : ft800.h
  * Description        : funciones para escribir en ft800
  ******************************************************************************
  */
#ifndef __ft800_H
#define __ft800_H

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "ft_gpu.h"

#define FT800_PD_N  GPIO_PIN_0
#define FT800_CS_N  GPIO_PIN_1
#define LCD_WQVGA

void ft800cmdWrite(unsigned char);
void ft800memWrite8(unsigned long, unsigned char);
void ft800memWrite16(unsigned long, unsigned int);
void ft800memWrite32(unsigned long, unsigned long);
unsigned char ft800memRead8(unsigned long);
unsigned int ft800memRead16(unsigned long);
unsigned long ft800memRead32(unsigned long);
unsigned int incCMDOffset(unsigned int, unsigned char);
void ft800_Init(void);
unsigned int incCMDOffset(unsigned int currentOffset, unsigned char commandSize);

#endif /*__ft800_H */
```

```
/**
  ******************************************************************************
  * File Name          : gpio.c
  * Description        : This file provides code for the configuration
  *                      of all used GPIO pins.
  ******************************************************************************
  **/

/* Includes ------------------------------------------------------------------*/
#include "gpio.h"

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(ARRANQUE_GPIO_Port, ARRANQUE_Pin, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(CS_SD_GPIO_Port, CS_SD_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(CS_PAN_GPIO_Port, CS_PAN_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(PD_PAN_GPIO_Port, PD_PAN_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SENTIDO_GPIO_Port, SENTIDO_Pin, GPIO_PIN_SET);

  /*Configure GPIO pins : PAPin PAPin */
  GPIO_InitStruct.Pin = ARRANQUE_Pin|PD_PAN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = CS_SD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(CS_SD_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = CD_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(CD_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PBPin PBPin */
  GPIO_InitStruct.Pin = CS_PAN_Pin|SENTIDO_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PtPin */
  GPIO_InitStruct.Pin = OUTRAN_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(OUTRAN_GPIO_Port, &GPIO_InitStruct);

}


/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : gpio.h
  * Description        : This file contains all the functions prototypes for
  *                      the gpio
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __gpio_H
#define __gpio_H
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "main.h"


void MX_GPIO_Init(void);

/* USER CODE BEGIN Prototypes */

/* USER CODE END Prototypes */

#ifdef __cplusplus
}
#endif
#endif /*__ pinoutConfig_H */



/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : SPI.c
  * Description        : This file provides code for the configuration
  *                      of the SPI instances.
  ******************************************************************************
  **/

/* Includes ------------------------------------------------------------------*/
#include "spi.h"
#include "gpio.h"

SPI_HandleTypeDef hspi2;
SPI_HandleTypeDef hspi5;

/* SPI2 init function */
void MX_SPI2_Init(void)
{

  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 15;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }

}
/* SPI5 init function */
void MX_SPI5_Init(void)
{

  hspi5.Instance = SPI5;
  hspi5.Init.Mode = SPI_MODE_MASTER;
  hspi5.Init.Direction = SPI_DIRECTION_2LINES;
  hspi5.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi5.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi5.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi5.Init.NSS = SPI_NSS_SOFT;
  hspi5.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_16;
  hspi5.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi5.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi5.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi5.Init.CRCPolynomial = 15;
  if (HAL_SPI_Init(&hspi5) != HAL_OK)
  {
    Error_Handler();
  }

}

void HAL_SPI_MspInit(SPI_HandleTypeDef* spiHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(spiHandle->Instance==SPI2)
  {
  /* USER CODE BEGIN SPI2_MspInit 0 */

  /* USER CODE END SPI2_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_SPI2_CLK_ENABLE();

    /**SPI2 GPIO Configuration
```

```
    PB13     ------> SPI2_SCK
    PB14     ------> SPI2_MISO
    PB15     ------> SPI2_MOSI
    */
    GPIO_InitStruct.Pin = SPI2_SCK_SD_Pin|SPI2_MISO_SD_Pin|SPI2_MOSI_SD_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* USER CODE BEGIN SPI2_MspInit 1 */

  /* USER CODE END SPI2_MspInit 1 */
  }
  else if(spiHandle->Instance==SPI5)
  {
  /* USER CODE BEGIN SPI5_MspInit 0 */

  /* USER CODE END SPI5_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_SPI5_CLK_ENABLE();

    /**SPI5 GPIO Configuration
    PB0     ------> SPI5_SCK
    PA12     ------> SPI5_MISO
    PB8     ------> SPI5_MOSI
    */
    GPIO_InitStruct.Pin = SPI5_SCK_PANT_Pin|SPI5_MOSI_PANT_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF6_SPI5;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = SPI5_MISO_PANT_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF6_SPI5;
    HAL_GPIO_Init(SPI5_MISO_PANT_GPIO_Port, &GPIO_InitStruct);

  /* USER CODE BEGIN SPI5_MspInit 1 */

  /* USER CODE END SPI5_MspInit 1 */
  }
}

void HAL_SPI_MspDeInit(SPI_HandleTypeDef* spiHandle)
{

  if(spiHandle->Instance==SPI2)
  {
  /* USER CODE BEGIN SPI2_MspDeInit 0 */

  /* USER CODE END SPI2_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_SPI2_CLK_DISABLE();

    /**SPI2 GPIO Configuration
    PB13     ------> SPI2_SCK
    PB14     ------> SPI2_MISO
    PB15     ------> SPI2_MOSI
    */
    HAL_GPIO_DeInit(GPIOB, SPI2_SCK_SD_Pin|SPI2_MISO_SD_Pin|SPI2_MOSI_SD_Pin);

  /* USER CODE BEGIN SPI2_MspDeInit 1 */

  /* USER CODE END SPI2_MspDeInit 1 */
  }
  else if(spiHandle->Instance==SPI5)
  {
  /* USER CODE BEGIN SPI5_MspDeInit 0 */
```

```
  /* USER CODE END SPI5_MspDeInit 0 */
    /* Peripheral clock disable */
    __HAL_RCC_SPI5_CLK_DISABLE();

    /**SPI5 GPIO Configuration
    PB0     ------> SPI5_SCK
    PA12     ------> SPI5_MISO
    PB8     ------> SPI5_MOSI
    */
    HAL_GPIO_DeInit(GPIOB, SPI5_SCK_PANT_Pin|SPI5_MOSI_PANT_Pin);

    HAL_GPIO_DeInit(SPI5_MISO_PANT_GPIO_Port, SPI5_MISO_PANT_Pin);

  /* USER CODE BEGIN SPI5_MspDeInit 1 */

  /* USER CODE END SPI5_MspDeInit 1 */
  }
}


/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : SPI.h
  * Description        : This file provides code for the configuration
  *                      of the SPI instances.
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __spi_H
#define __spi_H
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "main.h"

extern SPI_HandleTypeDef hspi2;
extern SPI_HandleTypeDef hspi5;

extern void Error_Handler(void);

void MX_SPI2_Init(void);
void MX_SPI5_Init(void);

#ifdef __cplusplus
}
#endif
#endif /*__ spi_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```c
/**
  ******************************************************************************
  * File Name          : TIM.c
  * Description        : This file provides code for the configuration
  *                      of the TIM instances.(configuracion de timers)
  ******************************************************************************
  **/

/* Includes ------------------------------------------------------------------*/
#include "tim.h"
#include "gpio.h"


TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim5;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim9;

/* TIM1 init function */
void MX_TIM1_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 96;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 10000;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}
/* TIM5 init function */
void MX_TIM5_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_IC_InitTypeDef sConfigIC;

  htim5.Instance = TIM5;
  htim5.Init.Prescaler = 960;
  htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim5.Init.Period = 4294967295;
  htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_IC_Init(&htim5) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
```

```
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
  sConfigIC.ICFilter = 0;
  if (HAL_TIM_IC_ConfigChannel(&htim5, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM6 init function */
void MX_TIM6_Init(void)
{
  TIM_MasterConfigTypeDef sMasterConfig;

  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 9600;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 30000;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

/* TIM9 init function */
void MX_TIM9_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig;

  htim9.Instance = TIM9;
  htim9.Init.Prescaler = 96;
  htim9.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim9.Init.Period = 10000;
  htim9.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim9) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim9, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

}

void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
{

  if(tim_baseHandle->Instance==TIM1)
  {
    /* Peripheral clock enable */
    __HAL_RCC_TIM1_CLK_ENABLE();

    /* Peripheral interrupt init */
    HAL_NVIC_SetPriority(TIM1_UP_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM1_UP_IRQn);
  }
  else if(tim_baseHandle->Instance==TIM6)
   {
     /* Peripheral clock enable */
     __HAL_RCC_TIM6_CLK_ENABLE();
```

```
    /* Peripheral interrupt init */
    HAL_NVIC_SetPriority(TIM6_DAC_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM6_DAC_IRQn);
   }
  else if(tim_baseHandle->Instance==TIM9)
   {
    /* Peripheral clock enable */
    __HAL_RCC_TIM9_CLK_ENABLE();

    /* Peripheral interrupt init */
    HAL_NVIC_SetPriority(TIM1_BRK_TIM9_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM1_BRK_TIM9_IRQn);
   }
}

void HAL_TIM_IC_MspInit(TIM_HandleTypeDef* tim_icHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(tim_icHandle->Instance==TIM5)
  {
    /* Peripheral clock enable */
    __HAL_RCC_TIM5_CLK_ENABLE();

    /**TIM5 GPIO Configuration
    PA1     ------> TIM5_CH2
    */
    GPIO_InitStruct.Pin = GPIO_PIN_1;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF2_TIM5;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* Peripheral interrupt init */
    HAL_NVIC_SetPriority(TIM5_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM5_IRQn);
  }
}


void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
{

  if(tim_baseHandle->Instance==TIM1)
  {
    /* Peripheral clock disable */
    __HAL_RCC_TIM1_CLK_DISABLE();

    /* Peripheral interrupt Deinit*/
    HAL_NVIC_DisableIRQ(TIM1_UP_IRQn);

  }

  else if(tim_baseHandle->Instance==TIM6)
    {
      /* Peripheral clock disable */
      __HAL_RCC_TIM6_CLK_DISABLE();

      /* Peripheral interrupt Deinit*/
      HAL_NVIC_DisableIRQ(TIM6_DAC_IRQn);

    }
  else if(tim_baseHandle->Instance==TIM9)
    {
      /* Peripheral clock disable */
      __HAL_RCC_TIM9_CLK_DISABLE();
    }
}

void HAL_TIM_IC_MspDeInit(TIM_HandleTypeDef* tim_icHandle)
{
```

```
  if(tim_icHandle->Instance==TIM5)
  {
    /* Peripheral clock disable */
    __HAL_RCC_TIM5_CLK_DISABLE();

    /**TIM5 GPIO Configuration
    PA1     ------> TIM5_CH2
    */
    HAL_GPIO_DeInit(GPIOA, GPIO_PIN_1);

    /* Peripheral interrupt Deinit*/
    HAL_NVIC_DisableIRQ(TIM5_IRQn);

  }
}
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * File Name          : TIM.h
  * Description        : This file provides code for the configuration
  *                      of the TIM instances.
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __tim_H
#define __tim_H
#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "main.h"

extern TIM_HandleTypeDef htim1;
extern TIM_HandleTypeDef htim5;
extern TIM_HandleTypeDef htim6;
extern TIM_HandleTypeDef htim9;

extern void Error_Handler(void);

void MX_TIM1_Init(void);
void MX_TIM5_Init(void);
void MX_TIM6_Init(void);
void MX_TIM9_Init(void);


#ifdef __cplusplus
}
#endif
#endif /*__ tim_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/*-----------------------------------------------------------------------*/
/* Low level disk I/O module skeleton for FatFs     (C)ChaN, 2014        */
/*                                                                       */
/*   Portions COPYRIGHT 2015 STMicroelectronics                          */
/*   Portions Copyright (C) 2014, ChaN, all right reserved               */
/*-----------------------------------------------------------------------*/
/* If a working storage control module is available, it should be        */
/* attached to the FatFs via a glue function rather than modifying it.   */
/* This is an example of glue functions to attach various exsisting      */
/* storage control modules to the FatFs module with a defined API.       */
/*-----------------------------------------------------------------------*/

/**
  ******************************************************************************
  * @file    diskio.c
  * @author  MCD Application Team
  * @version V1.3.0
  * @date    08-May-2015
  * @brief   FatFs low level disk I/O module.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "diskio.h"
#include "ff_gen_drv.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
extern Disk_drvTypeDef  disk;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/**
  * @brief  Gets Disk Status
  * @param  pdrv: Physical drive number (0..)
  * @retval DSTATUS: Operation status
  */
DSTATUS disk_status (
    BYTE pdrv       /* Physical drive nmuber to identify the drive */
)
{
  DSTATUS stat;

  stat = disk.drv[pdrv]->disk_status(disk.lun[pdrv]);
  return stat;
}

/**
  * @brief  Initializes a Drive
  * @param  pdrv: Physical drive number (0..)
  * @retval DSTATUS: Operation status
  */
DSTATUS disk_initialize (
    BYTE pdrv               /* Physical drive nmuber to identify the drive */
)
{
  DSTATUS stat = RES_OK;

  if(disk.is_initialized[pdrv] == 0)
  {
    disk.is_initialized[pdrv] = 1;
    stat = disk.drv[pdrv]->disk_initialize(disk.lun[pdrv]);
  }
  return stat;
}

/**
  * @brief  Reads Sector(s)
  * @param  pdrv: Physical drive number (0..)
  * @param  *buff: Data buffer to store read data
  * @param  sector: Sector address (LBA)
```

```
  * @param  count: Number of sectors to read (1..128)
  * @retval DRESULT: Operation result
  */
DRESULT disk_read (
    BYTE pdrv,      /* Physical drive nmuber to identify the drive */
    BYTE *buff,     /* Data buffer to store read data */
    DWORD sector,           /* Sector address in LBA */
    UINT count      /* Number of sectors to read */
)
{
  DRESULT res;

  res = disk.drv[pdrv]->disk_read(disk.lun[pdrv], buff, sector, count);
  return res;
}

/**
  * @brief  Writes Sector(s)
  * @param  pdrv: Physical drive number (0..)
  * @param  *buff: Data to be written
  * @param  sector: Sector address (LBA)
  * @param  count: Number of sectors to write (1..128)
  * @retval DRESULT: Operation result
  */
#if _USE_WRITE == 1
DRESULT disk_write (
    BYTE pdrv,      /* Physical drive nmuber to identify the drive */
    const BYTE *buff,   /* Data to be written */
    DWORD sector,       /* Sector address in LBA */
    UINT count          /* Number of sectors to write */
)
{
  DRESULT res;

  res = disk.drv[pdrv]->disk_write(disk.lun[pdrv], buff, sector, count);
  return res;
}
#endif /* _USE_WRITE == 1 */

/**
  * @brief  I/O control operation
  * @param  pdrv: Physical drive number (0..)
  * @param  cmd: Control code
  * @param  *buff: Buffer to send/receive control data
  * @retval DRESULT: Operation result
  */
#if _USE_IOCTL == 1
DRESULT disk_ioctl (
    BYTE pdrv,      /* Physical drive nmuber (0..) */
    BYTE cmd,       /* Control code */
    void *buff      /* Buffer to send/receive control data */
)
{
  DRESULT res;

  res = disk.drv[pdrv]->disk_ioctl(disk.lun[pdrv], cmd, buff);
  return res;
}
#endif /* _USE_IOCTL == 1 */

/**
  * @brief  Gets Time from RTC
  * @param  None
  * @retval Time in DWORD
  */
__weak DWORD get_fattime (void)
{
  return 0;
}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/*-----------------------------------------------------------------------/
/  Low level disk interface modlue include file   (C)ChaN, 2014          /
/-----------------------------------------------------------------------*/

#ifndef _DISKIO_DEFINED
#define _DISKIO_DEFINED

#ifdef __cplusplus
extern "C" {
#endif

#define _USE_WRITE  1   /* 1: Enable disk_write function */
#define _USE_IOCTL  1   /* 1: Enable disk_ioctl fucntion */

#include "integer.h"


/* Status of Disk Functions */
typedef BYTE    DSTATUS;

/* Results of Disk Functions */
typedef enum {
    RES_OK = 0,     /* 0: Successful */
    RES_ERROR,      /* 1: R/W Error */
    RES_WRPRT,      /* 2: Write Protected */
    RES_NOTRDY,     /* 3: Not Ready */
    RES_PARERR      /* 4: Invalid Parameter */
} DRESULT;


/*---------------------------------------*/
/* Prototypes for disk control functions */


DSTATUS disk_initialize (BYTE pdrv);
DSTATUS disk_status (BYTE pdrv);
DRESULT disk_read (BYTE pdrv, BYTE* buff, DWORD sector, UINT count);
DRESULT disk_write (BYTE pdrv, const BYTE* buff, DWORD sector, UINT count);
DRESULT disk_ioctl (BYTE pdrv, BYTE cmd, void* buff);
DWORD get_fattime (void);

/* Disk Status Bits (DSTATUS) */

#define STA_NOINIT      0x01    /* Drive not initialized */
#define STA_NODISK      0x02    /* No medium in the drive */
#define STA_PROTECT     0x04    /* Write protected */


/* Command code for disk_ioctrl fucntion */

/* Generic command (Used by FatFs) */
#define CTRL_SYNC       0   /* Complete pending write process (needed at _FS_READONLY == 0) */
#define GET_SECTOR_COUNT    1   /* Get media size (needed at _USE_MKFS == 1) */
#define GET_SECTOR_SIZE     2   /* Get sector size (needed at _MAX_SS != _MIN_SS) */
#define GET_BLOCK_SIZE      3   /* Get erase block size (needed at _USE_MKFS == 1) */
#define CTRL_TRIM       4   /* Inform device that the data on the block of sectors is no
longer used (needed at _USE_TRIM == 1) */

/* Generic command (Not used by FatFs) */
#define CTRL_POWER          5   /* Get/Set power status */
#define CTRL_LOCK           6   /* Lock/Unlock media removal */
#define CTRL_EJECT          7   /* Eject media */
#define CTRL_FORMAT         8   /* Create physical format on the media */

/* MMC/SDC specific ioctl command */
#define MMC_GET_TYPE        10  /* Get card type */
#define MMC_GET_CSD         11  /* Get CSD */
#define MMC_GET_CID         12  /* Get CID */
#define MMC_GET_OCR         13  /* Get OCR */
#define MMC_GET_SDSTAT      14  /* Get SD status */

/* ATA/CF specific ioctl command */
#define ATA_GET_REV         20  /* Get F/W revision */
```

```
#define ATA_GET_MODEL       21  /* Get model name */
#define ATA_GET_SN          22  /* Get serial number */

#ifdef __cplusplus
}
#endif

#endif
```

```
/*----------------------------------------------------------------------------/
/  FatFs - FAT file system module  R0.11                  (C)ChaN, 2015
/-----------------------------------------------------------------------------/
/ FatFs module is a free software that opened under license policy of
/ following conditions.
/
/ Copyright (C) 2015, ChaN, all right reserved.
/
/ 1. Redistributions of source code must retain the above copyright notice,
/    this condition and the following disclaimer.
/
/ This software is provided by the copyright holder and contributors "AS IS"
/ and any warranties related to this software are DISCLAIMED.
/ The copyright owner or contributors be NOT LIABLE for any damages caused
/ by use of this software.
/----------------------------------------------------------------------------/
/ Feb 26,'06 R0.00  Prototype.
/
/ Apr 29,'06 R0.01  First stable version.
/
/ Jun 01,'06 R0.02  Added FAT12 support.
/                   Removed unbuffered mode.
/                   Fixed a problem on small (<32M) partition.
/ Jun 10,'06 R0.02a Added a configuration option (_FS_MINIMUM).
/
/ Sep 22,'06 R0.03  Added f_rename().
/                   Changed option _FS_MINIMUM to _FS_MINIMIZE.
/ Dec 11,'06 R0.03a Improved cluster scan algorithm to write files fast.
/                   Fixed f_mkdir() creates incorrect directory on FAT32.
/
/ Feb 04,'07 R0.04  Supported multiple drive system.
/                   Changed some interfaces for multiple drive system.
/                   Changed f_mountdrv() to f_mount().
/                   Added f_mkfs().
/ Apr 01,'07 R0.04a Supported multiple partitions on a physical drive.
/                   Added a capability of extending file size to f_lseek().
/                   Added minimization level 3.
/                   Fixed an endian sensitive code in f_mkfs().
/ May 05,'07 R0.04b Added a configuration option _USE_NTFLAG.
/                   Added FSINFO support.
/                   Fixed DBCS name can result FR_INVALID_NAME.
/                   Fixed short seek (<= csize) collapses the file object.
/
/ Aug 25,'07 R0.05  Changed arguments of f_read(), f_write() and f_mkfs().
/                   Fixed f_mkfs() on FAT32 creates incorrect FSINFO.
/                   Fixed f_mkdir() on FAT32 creates incorrect directory.
/ Feb 03,'08 R0.05a Added f_truncate() and f_utime().
/                   Fixed off by one error at FAT sub-type determination.
/                   Fixed btr in f_read() can be mistruncated.
/                   Fixed cached sector is not flushed when create and close without write.
/
/ Apr 01,'08 R0.06  Added fputc(), fputs(), fprintf() and fgets().
/                   Improved performance of f_lseek() on moving to the same or following
cluster.
/
/ Apr 01,'09 R0.07  Merged Tiny-FatFs as a configuration option. (_FS_TINY)
/                   Added long file name feature.
/                   Added multiple code page feature.
/                   Added re-entrancy for multitask operation.
/                   Added auto cluster size selection to f_mkfs().
/                   Added rewind option to f_readdir().
/                   Changed result code of critical errors.
/                   Renamed string functions to avoid name collision.
/ Apr 14,'09 R0.07a Separated out OS dependent code on reentrant cfg.
/                   Added multiple sector size feature.
/ Jun 21,'09 R0.07c Fixed f_unlink() can return FR_OK on error.
/                   Fixed wrong cache control in f_lseek().
/                   Added relative path feature.
/                   Added f_chdir() and f_chdrive().
/                   Added proper case conversion to extended character.
/ Nov 03,'09 R0.07e Separated out configuration options from ff.h to ffconf.h.
/                   Fixed f_unlink() fails to remove a sub-directory on _FS_RPATH.
/                   Fixed name matching error on the 13 character boundary.
```

```
/                   Added a configuration option, _LFN_UNICODE.
/                   Changed f_readdir() to return the SFN with always upper case on non-LFN
cfg.
/
/ May 15,'10 R0.08  Added a memory configuration option. (_USE_LFN = 3)
/                   Added file lock feature. (_FS_SHARE)
/                   Added fast seek feature. (_USE_FASTSEEK)
/                   Changed some types on the API, XCHAR->TCHAR.
/                   Changed .fname in the FILINFO structure on Unicode cfg.
/                   String functions support UTF-8 encoding files on Unicode cfg.
/ Aug 16,'10 R0.08a Added f_getcwd().
/                   Added sector erase feature. (_USE_ERASE)
/                   Moved file lock semaphore table from fs object to the bss.
/                   Fixed a wrong directory entry is created on non-LFN cfg when the given
name contains ';'.
/                   Fixed f_mkfs() creates wrong FAT32 volume.
/ Jan 15,'11 R0.08b Fast seek feature is also applied to f_read() and f_write().
/                   f_lseek() reports required table size on creating CLMP.
/                   Extended format syntax of f_printf().
/                   Ignores duplicated directory separators in given path name.
/
/ Sep 06,'11 R0.09  f_mkfs() supports multiple partition to complete the multiple partition
feature.
/                   Added f_fdisk().
/ Aug 27,'12 R0.09a Changed f_open() and f_opendir() reject null object pointer to avoid
crash.
/                   Changed option name _FS_SHARE to _FS_LOCK.
/                   Fixed assertion failure due to OS/2 EA on FAT12/16 volume.
/ Jan 24,'13 R0.09b Added f_setlabel() and f_getlabel().
/
/ Oct 02,'13 R0.10  Added selection of character encoding on the file. (_STRF_ENCODE)
/                   Added f_closedir().
/                   Added forced full FAT scan for f_getfree(). (_FS_NOFSINFO)
/                   Added forced mount feature with changes of f_mount().
/                   Improved behavior of volume auto detection.
/                   Improved write throughput of f_puts() and f_printf().
/                   Changed argument of f_chdrive(), f_mkfs(), disk_read() and disk_write().
/                   Fixed f_write() can be truncated when the file size is close to 4GB.
/                   Fixed f_open(), f_mkdir() and f_setlabel() can return incorrect error
code.
/ Jan 15,'14 R0.10a Added arbitrary strings as drive number in the path name. (_STR_VOLUME_ID)
/                   Added a configuration option of minimum sector size. (_MIN_SS)
/                   2nd argument of f_rename() can have a drive number and it will be ignored.
/                   Fixed f_mount() with forced mount fails when drive number is >= 1.
/                   Fixed f_close() invalidates the file object without volume lock.
/                   Fixed f_closedir() returns but the volume lock is left acquired.
/                   Fixed creation of an entry with LFN fails on too many SFN collisions.
/ May 19,'14 R0.10b Fixed a hard error in the disk I/O layer can collapse the directory entry.
/                   Fixed LFN entry is not deleted on delete/rename an object with lossy
converted SFN.
/ Nov  9,'14 R0.10c Added a configuration option for the platforms without RTC. (_FS_NORTC)
/                   Fixed volume label created by Mac OS X cannot be retrieved with
f_getlabel(). (appeared at R0.09b)
/                   Fixed a potential problem of FAT access that can appear on disk error.
/                   Fixed null pointer dereference on attempting to delete the root
direcotry. (appeared at R0.08)
/ Feb 02,'15 R0.11  Added f_findfirst() and f_findnext(). (_USE_FIND)
/                   Fixed f_unlink() does not remove cluster chain of the file. (appeared at
R0.10c)
/                   Fixed _FS_NORTC option does not work properly. (appeared at R0.10c)
/---------------------------------------------------------------------------*/

#include "ff.h"         /* Declarations of FatFs API */
#include "diskio.h"     /* Declarations of disk I/O functions */


/*--------------------------------------------------------------------------

   Module Private Definitions

---------------------------------------------------------------------------*/

#if _FATFS != 32020 /* Revision ID */
```

```
#error Wrong include file (ff.h).
#endif


/* Reentrancy related */
#if _FS_REENTRANT
#if _USE_LFN == 1
#error Static LFN work area cannot be used at thread-safe configuration
#endif
#define ENTER_FF(fs)        { if (!lock_fs(fs)) return FR_TIMEOUT; }
#define LEAVE_FF(fs, res)   { unlock_fs(fs, res); return res; }
#else
#define ENTER_FF(fs)
#define LEAVE_FF(fs, res)   return res
#endif

#define ABORT(fs, res)      { fp->err = (BYTE)(res); LEAVE_FF(fs, res); }


/* Definitions of sector size */
#if (_MAX_SS < _MIN_SS) || (_MAX_SS != 512 && _MAX_SS != 1024 && _MAX_SS != 2048 && _MAX_SS
!= 4096) || (_MIN_SS != 512 && _MIN_SS != 1024 && _MIN_SS != 2048 && _MIN_SS != 4096)
#error Wrong sector size configuration
#endif
#if _MAX_SS == _MIN_SS
#define SS(fs)  ((UINT)_MAX_SS) /* Fixed sector size */
#else
#define SS(fs)  ((fs)->ssize)   /* Variable sector size */
#endif


/* Timestamp feature */
#if _FS_NORTC == 1
#if _NORTC_YEAR < 1980 || _NORTC_YEAR > 2107 || _NORTC_MON < 1 || _NORTC_MON > 12 ||
_NORTC_MDAY < 1 || _NORTC_MDAY > 31
#error Invalid _FS_NORTC settings
#endif
#define GET_FATTIME()   ((DWORD)(_NORTC_YEAR - 1980) << 25 | (DWORD)_NORTC_MON << 21 |
(DWORD)_NORTC_MDAY << 16)
#else
#define GET_FATTIME()   get_fattime()
#endif


/* File access control feature */
#if _FS_LOCK
#if _FS_READONLY
#error _FS_LOCK must be 0 at read-only configuration
#endif
typedef struct {
    FATFS *fs;      /* Object ID 1, volume (NULL:blank entry) */
    DWORD clu;      /* Object ID 2, directory (0:root) */
    WORD idx;       /* Object ID 3, directory index */
    WORD ctr;       /* Object open counter, 0:none, 0x01..0xFF:read mode open count,
    0x100:write mode */
} FILESEM;
#endif



/* DBCS code ranges and SBCS extend character conversion table */

#if _CODE_PAGE == 932   /* Japanese Shift-JIS */
#define _DF1S   0x81    /* DBC 1st byte range 1 start */
#define _DF1E   0x9F    /* DBC 1st byte range 1 end */
#define _DF2S   0xE0    /* DBC 1st byte range 2 start */
#define _DF2E   0xFC    /* DBC 1st byte range 2 end */
#define _DS1S   0x40    /* DBC 2nd byte range 1 start */
#define _DS1E   0x7E    /* DBC 2nd byte range 1 end */
#define _DS2S   0x80    /* DBC 2nd byte range 2 start */
#define _DS2E   0xFC    /* DBC 2nd byte range 2 end */

#elif _CODE_PAGE == 936 /* Simplified Chinese GBK */
```

```
#define _DF1S   0x81
#define _DF1E   0xFE
#define _DS1S   0x40
#define _DS1E   0x7E
#define _DS2S   0x80
#define _DS2E   0xFE

#elif _CODE_PAGE == 949 /* Korean */
#define _DF1S   0x81
#define _DF1E   0xFE
#define _DS1S   0x41
#define _DS1E   0x5A
#define _DS2S   0x61
#define _DS2E   0x7A
#define _DS3S   0x81
#define _DS3E   0xFE

#elif _CODE_PAGE == 950 /* Traditional Chinese Big5 */
#define _DF1S   0x81
#define _DF1E   0xFE
#define _DS1S   0x40
#define _DS1E   0x7E
#define _DS2S   0xA1
#define _DS2E   0xFE

#elif _CODE_PAGE == 437 /* U.S. (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x90,0x41,0x8E,0x41,0x8F,0x80,0x45,0x45,0x45,0x49,0x49,0x49,0x8E,0x8F,0x90,0x92,0x9
2,0x4F,0x99,0x4F,0x55,0x55,0x59,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 720 /* Arabic (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x45,0x41,0x84,0x41,0x86,0x43,0x45,0x45,0x45,0x49,0x49,0x8D,0x8E,0x8F,0x90,0x92,0x9
2,0x93,0x94,0x95,0x49,0x49,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 737 /* Greek (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x92,0x9
2,0x93,0x94,0x95,0x96,0x97,0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87, \

                0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0xAA,0x92,0x93,0x94,0x95,0x9
                6,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
```

```
                xDF, \

                0x97,0xEA,0xEB,0xEC,0xE4,0xED,0xEE,0xE7,0xE8,0xF1,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 775 /* Baltic (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x91,0xA0,0x8E,0x95,0x8F,0x80,0xAD,0xED,0x8A,0x8A,0xA1,0x8D,0x8E,0x8F,0x90,0x92,0x9
2,0xE2,0x99,0x95,0x96,0x97,0x97,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xE0,0xA3,0xA3,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xB5,0xB6,0xB7,0xB8,0xBD,0xBE,0xC6,0xC7,0xA5,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE3,0xE8,0xE8,0xEA,0xEA,0xEE,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 850 /* Multilingual Latin 1 (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xDE,0x8E,0x8F,0x90,0x92,0x9
2,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \

                0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xED,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 852 /* Latin 2 (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xDE,0x8F,0x80,0x9D,0xD3,0x8A,0x8A,0xD7,0x8D,0x8E,0x8F,0x90,0x91,0x9
1,0xE2,0x99,0x95,0x95,0x97,0x97,0x99,0x9A,0x9B,0x9B,0x9D,0x9E,0x9F, \

                0xB5,0xD6,0xE0,0xE9,0xA4,0xA4,0xA6,0xA6,0xA8,0xA8,0xAA,0x8D,0xAC,0xB8,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBD,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC6,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD1,0xD1,0xD2,0xD3,0xD2,0xD5,0xD6,0xD7,0xB7,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE3,0xD5,0xE6,0xE6,0xE8,0xE9,0xE8,0xEB,0xED,0xED,0xDD,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xEB,0xFC,0xFC,0xFE,0
                xFF}

#elif _CODE_PAGE == 855 /* Cyrillic (OEM) */
#define _DF1S   0
#define _EXCVT
{0x81,0x81,0x83,0x83,0x85,0x85,0x87,0x87,0x89,0x89,0x8B,0x8B,0x8D,0x8D,0x8F,0x8F,0x91,0x91,0x9
3,0x93,0x95,0x95,0x97,0x97,0x99,0x99,0x9B,0x9B,0x9D,0x9D,0x9F,0x9F, \

                0xA1,0xA1,0xA3,0xA3,0xA5,0xA5,0xA7,0xA7,0xA9,0xA9,0xAB,0xAB,0xAD,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB6,0xB6,0xB8,0xB8,0xB9,0xBA,0xBB,0xBC,0xBE,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD1,0xD1,0xD3,0xD3,0xD5,0xD5,0xD7,0xD7,0xDD,0xD9,0xDA,0xDB,0xDC,0xDD,0xE0,0
                xDF, \
```

```
                0xE0,0xE2,0xE2,0xE4,0xE4,0xE6,0xE6,0xE8,0xE8,0xEA,0xEA,0xEC,0xEC,0xEE,0xEE,0xE
                F,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF8,0xFA,0xFA,0xFC,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 857 /* Turkish (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0x98,0x8E,0x8F,0x90,0x92,0x9
2,0xE2,0x99,0xE3,0xEA,0xEB,0x98,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9E, \

                0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA6,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xDE,0x59,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 858 /* Multilingual Latin 1 + Euro (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x9A,0x90,0xB6,0x8E,0xB7,0x8F,0x80,0xD2,0xD3,0xD4,0xD8,0xD7,0xDE,0x8E,0x8F,0x90,0x92,0x9
2,0xE2,0x99,0xE3,0xEA,0xEB,0x59,0x99,0x9A,0x9D,0x9C,0x9D,0x9E,0x9F, \

                0xB5,0xD6,0xE0,0xE9,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC7,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD1,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE5,0xE5,0xE6,0xE7,0xE7,0xE9,0xEA,0xEB,0xED,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 862 /* Hebrew (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0x41,0x49,0x4F,0x55,0xA5,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0x21,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 866 /* Russian (OEM) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \
```

```
                0x90,0x91,0x92,0x93,0x9d,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9
                F,0xF0,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 874 /* Thai (OEM, Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 1250 /* Central Europe (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x8D,0x8E,0x8F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xA3,0xB4,0xB5,0xB6,0xB7,0xB8,0xA5,0xAA,0xBB,0xBC,0xBD,0xBC,0
                xAF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xFF}

#elif _CODE_PAGE == 1251 /* Cyrillic (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x82,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x80,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x8D,0x8E,0x8F, \

                0xA0,0xA2,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB2,0xA5,0xB5,0xB6,0xB7,0xA8,0xB9,0xAA,0xBB,0xA3,0xBD,0xBD,0
                xAF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF}

#elif _CODE_PAGE == 1252 /* Latin 1 (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0xAd,0x9B,0x8C,0x9D,0xAE,0x9F, \

                0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
```

```
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                x9F}

#elif _CODE_PAGE == 1253 /* Greek (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xA2,0xB8,0xB9,0
                xBA, \

                0xE0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xFB,0xBC,0xFD,0xBF,0
                xFF}

#elif _CODE_PAGE == 1254 /* Turkish (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x8A,0x9B,0x8C,0x9D,0x9E,0x9F, \

                0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                x9F}

#elif _CODE_PAGE == 1255 /* Hebrew (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xE0,0xE1,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,0xEE,0xE
                F,0xF0,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0
                xFF}

#elif _CODE_PAGE == 1256 /* Arabic (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x8C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0x41,0xE1,0x41,0xE3,0xE4,0xE5,0xE6,0x43,0x45,0x45,0x45,0x45,0xEC,0xED,0x49,0x4
                9,0xF0,0xF1,0xF2,0xF3,0x4F,0xF5,0xF6,0xF7,0xF8,0x55,0xFA,0x55,0x55,0xFD,0xFE,0
```

```
                xFF}

#elif _CODE_PAGE == 1257 /* Baltic (Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F, \

                0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xA8,0xB9,0xAA,0xBB,0xBC,0xBD,0xBE,0
                xAF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xFF}

#elif _CODE_PAGE == 1258 /* Vietnam (OEM, Windows) */
#define _DF1S   0
#define _EXCVT
{0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x9
2,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0xAC,0x9D,0x9E,0x9F, \

                0xA0,0x21,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xA
                F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0
                xBF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0
                xDF, \

                0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xEC,0xCD,0xCE,0xC
                F,0xD0,0xD1,0xF2,0xD3,0xD4,0xD5,0xD6,0xF7,0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xFE,0
                x9F}

#elif _CODE_PAGE == 1   /* ASCII (for only non-LFN cfg) */
#if _USE_LFN
#error Cannot use LFN feature without valid code page.
#endif
#define _DF1S   0

#else
#error Unknown code page

#endif


/* Character code support macros */
#define IsUpper(c)  (((c)>='A')&&((c)<='Z'))
#define IsLower(c)  (((c)>='a')&&((c)<='z'))
#define IsDigit(c)  (((c)>='0')&&((c)<='9'))

#if _DF1S       /* Code page is DBCS */

#ifdef _DF2S    /* Two 1st byte areas */
#define IsDBCS1(c)  (((BYTE)(c) >= _DF1S && (BYTE)(c) <= _DF1E) || ((BYTE)(c) >= _DF2S &&
(BYTE)(c) <= _DF2E))
#else           /* One 1st byte area */
#define IsDBCS1(c)  ((BYTE)(c) >= _DF1S && (BYTE)(c) <= _DF1E)
#endif

#ifdef _DS3S    /* Three 2nd byte areas */
#define IsDBCS2(c)  (((BYTE)(c) >= _DS1S && (BYTE)(c) <= _DS1E) || ((BYTE)(c) >= _DS2S &&
(BYTE)(c) <= _DS2E) || ((BYTE)(c) >= _DS3S && (BYTE)(c) <= _DS3E))
#else           /* Two 2nd byte areas */
#define IsDBCS2(c)  (((BYTE)(c) >= _DS1S && (BYTE)(c) <= _DS1E) || ((BYTE)(c) >= _DS2S &&
(BYTE)(c) <= _DS2E))
#endif

#else           /* Code page is SBCS */
```

```
#define IsDBCS1(c)  0
#define IsDBCS2(c)  0

#endif /* _DF1S */


/* Name status flags */
#define NSFLAG      11      /* Index of name status byte in fn[] */
#define NS_LOSS     0x01    /* Out of 8.3 format */
#define NS_LFN      0x02    /* Force to create LFN entry */
#define NS_LAST     0x04    /* Last segment */
#define NS_BODY     0x08    /* Lower case flag (body) */
#define NS_EXT      0x10    /* Lower case flag (ext) */
#define NS_DOT      0x20    /* Dot entry */


/* FAT sub-type boundaries (Differ from specs but correct for real DOS/Windows) */
#define MIN_FAT16   4086U   /* Minimum number of clusters as FAT16 */
#define MIN_FAT32   65526U  /* Minimum number of clusters as FAT32 */


/* FatFs refers the members in the FAT structures as byte array instead of
/ structure member because the structure is not binary compatible between
/ different platforms */

#define BS_jmpBoot          0       /* x86 jump instruction (3) */
#define BS_OEMName          3       /* OEM name (8) */
#define BPB_BytsPerSec      11      /* Sector size [byte] (2) */
#define BPB_SecPerClus      13      /* Cluster size [sector] (1) */
#define BPB_RsvdSecCnt      14      /* Size of reserved area [sector] (2) */
#define BPB_NumFATs         16      /* Number of FAT copies (1) */
#define BPB_RootEntCnt      17      /* Number of root directory entries for FAT12/16 (2) */
#define BPB_TotSec16        19      /* Volume size [sector] (2) */
#define BPB_Media           21      /* Media descriptor (1) */
#define BPB_FATSz16         22      /* FAT size [sector] (2) */
#define BPB_SecPerTrk       24      /* Track size [sector] (2) */
#define BPB_NumHeads        26      /* Number of heads (2) */
#define BPB_HiddSec         28      /* Number of special hidden sectors (4) */
#define BPB_TotSec32        32      /* Volume size [sector] (4) */
#define BS_DrvNum           36      /* Physical drive number (2) */
#define BS_BootSig          38      /* Extended boot signature (1) */
#define BS_VolID            39      /* Volume serial number (4) */
#define BS_VolLab           43      /* Volume label (8) */
#define BS_FilSysType       54      /* File system type (1) */
#define BPB_FATSz32         36      /* FAT size [sector] (4) */
#define BPB_ExtFlags        40      /* Extended flags (2) */
#define BPB_FSVer           42      /* File system version (2) */
#define BPB_RootClus        44      /* Root directory first cluster (4) */
#define BPB_FSInfo          48      /* Offset of FSINFO sector (2) */
#define BPB_BkBootSec       50      /* Offset of backup boot sector (2) */
#define BS_DrvNum32         64      /* Physical drive number (2) */
#define BS_BootSig32        66      /* Extended boot signature (1) */
#define BS_VolID32          67      /* Volume serial number (4) */
#define BS_VolLab32         71      /* Volume label (8) */
#define BS_FilSysType32     82      /* File system type (1) */
#define FSI_LeadSig         0       /* FSI: Leading signature (4) */
#define FSI_StrucSig        484     /* FSI: Structure signature (4) */
#define FSI_Free_Count      488     /* FSI: Number of free clusters (4) */
#define FSI_Nxt_Free        492     /* FSI: Last allocated cluster (4) */
#define MBR_Table           446     /* MBR: Partition table offset (2) */
#define SZ_PTE              16      /* MBR: Size of a partition table entry */
#define BS_55AA             510     /* Signature word (2) */

#define DIR_Name            0       /* Short file name (11) */
#define DIR_Attr            11      /* Attribute (1) */
#define DIR_NTres           12      /* Lower case flag (1) */
#define DIR_CrtTimeTenth    13      /* Created time sub-second (1) */
#define DIR_CrtTime         14      /* Created time (2) */
#define DIR_CrtDate         16      /* Created date (2) */
#define DIR_LstAccDate      18      /* Last accessed date (2) */
#define DIR_FstClusHI       20      /* Higher 16-bit of first cluster (2) */
#define DIR_WrtTime         22      /* Modified time (2) */
```

```
#define DIR_WrtDate         24      /* Modified date (2) */
#define DIR_FstClusLO       26      /* Lower 16-bit of first cluster (2) */
#define DIR_FileSize        28      /* File size (4) */
#define LDIR_Ord            0       /* LFN entry order and LLE flag (1) */
#define LDIR_Attr           11      /* LFN attribute (1) */
#define LDIR_Type           12      /* LFN type (1) */
#define LDIR_Chksum         13      /* Sum of corresponding SFN entry */
#define LDIR_FstClusLO      26      /* Must be zero (0) */
#define SZ_DIRE             32      /* Size of a directory entry */
#define LLEF                0x40    /* Last long entry flag in LDIR_Ord */
#define DDEM                0xE5    /* Deleted directory entry mark at DIR_Name[0] */
#define RDDEM               0x05    /* Replacement of the character collides with DDEM */



/*--------------------------------------------------------------*/
/* Module private work area                                     */
/*--------------------------------------------------------------*/
/* Remark: Uninitialized variables with static duration are
/  guaranteed zero/null at start-up. If not, either the linker
/  or start-up routine being used is out of ANSI-C standard.
*/

#if _VOLUMES < 1 || _VOLUMES > 9
#error Wrong _VOLUMES setting
#endif
static FATFS *FatFs[_VOLUMES];  /* Pointer to the file system objects (logical drives) */
static WORD Fsid;               /* File system mount ID */

#if _FS_RPATH && _VOLUMES >= 2
static BYTE CurrVol;            /* Current drive */
#endif

#if _FS_LOCK
static FILESEM Files[_FS_LOCK]; /* Open object lock semaphores */
#endif

#if _USE_LFN == 0           /* Non LFN feature */
#define DEFINE_NAMEBUF      BYTE sfn[12]
#define INIT_BUF(dobj)      (dobj).fn = sfn
#define FREE_BUF()
#else
#if _MAX_LFN < 12 || _MAX_LFN > 255
#error Wrong _MAX_LFN setting
#endif
#if _USE_LFN == 1           /* LFN feature with static working buffer */
static WCHAR LfnBuf[_MAX_LFN + 1];
#define DEFINE_NAMEBUF      BYTE sfn[12]
#define INIT_BUF(dobj)      { (dobj).fn = sfn; (dobj).lfn = LfnBuf; }
#define FREE_BUF()
#elif _USE_LFN == 2         /* LFN feature with dynamic working buffer on the stack */
#define DEFINE_NAMEBUF      BYTE sfn[12]; WCHAR lbuf[_MAX_LFN + 1]
#define INIT_BUF(dobj)      { (dobj).fn = sfn; (dobj).lfn = lbuf; }
#define FREE_BUF()
#elif _USE_LFN == 3         /* LFN feature with dynamic working buffer on the heap */
#define DEFINE_NAMEBUF      BYTE sfn[12]; WCHAR *lfn
#define INIT_BUF(dobj)      { lfn = ff_memalloc((_MAX_LFN + 1) * 2); if (!lfn)
LEAVE_FF((dobj).fs, FR_NOT_ENOUGH_CORE); (dobj).lfn = lfn; (dobj).fn = sfn; }
#define FREE_BUF()          ff_memfree(lfn)
#else
#error Wrong _USE_LFN setting
#endif
#endif

#ifdef _EXCVT
static const BYTE ExCvt[] = _EXCVT; /* Upper conversion table for extended characters */
#endif
```

```
/*-----------------------------------------------------------------------

   Module Private Functions

-----------------------------------------------------------------------*/
DWORD clust2sect (FATFS* fs, DWORD clst);
DWORD get_fat (FATFS* fs,   DWORD clst);

#if !_FS_READONLY
FRESULT put_fat (FATFS* fs, DWORD clst, DWORD val);
#endif /* !_FS_READONLY */

#if _USE_LFN
static void gen_numname (BYTE* dst, const BYTE* src, const WCHAR* lfn, UINT seq);
#endif /* !_USE_LFN */



/*-----------------------------------------------------------------------*/
/* String functions                                                      */
/*-----------------------------------------------------------------------*/

/* Copy memory to memory */
static
void mem_cpy (void* dst, const void* src, UINT cnt) {
    BYTE *d = (BYTE*)dst;
    const BYTE *s = (const BYTE*)src;

#if _WORD_ACCESS == 1
    while (cnt >= sizeof (int)) {
        *(int*)d = *(int*)s;
        d += sizeof (int); s += sizeof (int);
        cnt -= sizeof (int);
    }
#endif
    while (cnt--)
        *d++ = *s++;
}

/* Fill memory */
static
void mem_set (void* dst, int val, UINT cnt) {
    BYTE *d = (BYTE*)dst;

    while (cnt--)
        *d++ = (BYTE)val;
}

/* Compare memory to memory */
static
int mem_cmp (const void* dst, const void* src, UINT cnt) {
    const BYTE *d = (const BYTE *)dst, *s = (const BYTE *)src;
    int r = 0;

    while (cnt-- && (r = *d++ - *s++) == 0) ;
    return r;
}

/* Check if chr is contained in the string */
static
int chk_chr (const char* str, int chr) {
    while (*str && *str != chr) str++;
    return *str;
}



/*-----------------------------------------------------------------------*/
/* Request/Release grant to access the volume                            */
/*-----------------------------------------------------------------------*/
#if _FS_REENTRANT
```

```
static
int lock_fs (
    FATFS* fs       /* File system object */
)
{
    return ff_req_grant(fs->sobj);
}


static
void unlock_fs (
    FATFS* fs,      /* File system object */
    FRESULT res     /* Result code to be returned */
)
{
    if (fs &&
        res != FR_NOT_ENABLED &&
        res != FR_INVALID_DRIVE &&
        res != FR_INVALID_OBJECT &&
        res != FR_TIMEOUT) {
        ff_rel_grant(fs->sobj);
    }
}
#endif




/*----------------------------------------------------------------------------*/
/* File lock control functions                                                */
/*----------------------------------------------------------------------------*/
#if _FS_LOCK

static
FRESULT chk_lock (  /* Check if the file can be accessed */
    DIR* dp,        /* Directory object pointing the file to be checked */
    int acc         /* Desired access type (0:Read, 1:Write, 2:Delete/Rename) */
)
{
    UINT i, be;

    /* Search file semaphore table */
    for (i = be = 0; i < _FS_LOCK; i++) {
        if (Files[i].fs) {  /* Existing entry */
            if (Files[i].fs == dp->fs &&        /* Check if the object matched with an open
            object */
                Files[i].clu == dp->sclust &&
                Files[i].idx == dp->index) break;
        } else {            /* Blank entry */
            be = 1;
        }
    }
    if (i == _FS_LOCK)  /* The object is not opened */
        return (be || acc == 2) ? FR_OK : FR_TOO_MANY_OPEN_FILES;   /* Is there a blank
        entry for new object? */

    /* The object has been opened. Reject any open against writing file and all write mode
    open */
    return (acc || Files[i].ctr == 0x100) ? FR_LOCKED : FR_OK;
}


static
int enq_lock (void) /* Check if an entry is available for a new object */
{
    UINT i;

    for (i = 0; i < _FS_LOCK && Files[i].fs; i++) ;
    return (i == _FS_LOCK) ? 0 : 1;
}


static
```

```
UINT inc_lock (	/* Increment object open counter and returns its index (0:Internal error) */
	DIR* dp,	/* Directory object pointing the file to register or increment */
	int acc		/* Desired access (0:Read, 1:Write, 2:Delete/Rename) */
)
{
	UINT i;


	for (i = 0; i < _FS_LOCK; i++) {	/* Find the object */
		if (Files[i].fs == dp->fs &&
			Files[i].clu == dp->sclust &&
			Files[i].idx == dp->index) break;
	}

	if (i == _FS_LOCK) {				/* Not opened. Register it as new. */
		for (i = 0; i < _FS_LOCK && Files[i].fs; i++) ;
		if (i == _FS_LOCK) return 0;	/* No free entry to register (int err) */
		Files[i].fs = dp->fs;
		Files[i].clu = dp->sclust;
		Files[i].idx = dp->index;
		Files[i].ctr = 0;
	}

	if (acc && Files[i].ctr) return 0;	/* Access violation (int err) */

	Files[i].ctr = acc ? 0x100 : Files[i].ctr + 1;	/* Set semaphore value */

	return i + 1;
}


static
FRESULT dec_lock (	/* Decrement object open counter */
	UINT i			/* Semaphore index (1..) */
)
{
	WORD n;
	FRESULT res;


	if (--i < _FS_LOCK) {	/* Shift index number origin from 0 */
		n = Files[i].ctr;
		if (n == 0x100) n = 0;		/* If write mode open, delete the entry */
		if (n) n--;					/* Decrement read mode open count */
		Files[i].ctr = n;
		if (!n) Files[i].fs = 0;	/* Delete the entry if open count gets zero */
		res = FR_OK;
	} else {
		res = FR_INT_ERR;			/* Invalid index nunber */
	}
	return res;
}


static
void clear_lock (	/* Clear lock entries of the volume */
	FATFS *fs
)
{
	UINT i;

	for (i = 0; i < _FS_LOCK; i++) {
		if (Files[i].fs == fs) Files[i].fs = 0;
	}
}
#endif




/*-----------------------------------------------------------------------*/
/* Move/Flush disk access window in the file system object               */
/*-----------------------------------------------------------------------*/
```

```
#if !_FS_READONLY
static
FRESULT sync_window (
    FATFS* fs       /* File system object */
)
{
    DWORD wsect;
    UINT nf;
    FRESULT res = FR_OK;


    if (fs->wflag) {    /* Write back the sector if it is dirty */
        wsect = fs->winsect;    /* Current sector number */
        if (disk_write(fs->drv, fs->win.d8, wsect, 1) != RES_OK) {
            res = FR_DISK_ERR;
        } else {
            fs->wflag = 0;
            if (wsect - fs->fatbase < fs->fsize) {      /* Is it in the FAT area? */
                for (nf = fs->n_fats; nf >= 2; nf--) {  /* Reflect the change to all FAT
                copies */
                    wsect += fs->fsize;
                    disk_write(fs->drv, fs->win.d8, wsect, 1);
                }
            }
        }
    }
    return res;
}
#endif


static
FRESULT move_window (
    FATFS* fs,      /* File system object */
    DWORD sector    /* Sector number to make appearance in the fs->win[].d8 */
)
{
    FRESULT res = FR_OK;


    if (sector != fs->winsect) {    /* Window offset changed? */
#if !_FS_READONLY
        res = sync_window(fs);      /* Write-back changes */
#endif
        if (res == FR_OK) {         /* Fill sector window with new data */
            if (disk_read(fs->drv, fs->win.d8, sector, 1) != RES_OK) {
                sector = 0xFFFFFFFF;    /* Invalidate window if data is not reliable */
                res = FR_DISK_ERR;
            }
            fs->winsect = sector;
        }
    }
    return res;
}




/*-----------------------------------------------------------------------*/
/* Synchronize file system and strage device                             */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT sync_fs (   /* FR_OK: successful, FR_DISK_ERR: failed */
    FATFS* fs       /* File system object */
)
{
    FRESULT res;


    res = sync_window(fs);
    if (res == FR_OK) {
        /* Update FSINFO sector if needed */
```

```
        if (fs->fs_type == FS_FAT32 && fs->fsi_flag == 1) {
            /* Create FSINFO structure */
            mem_set(fs->win.d8, 0, SS(fs));
            ST_WORD(fs->win.d8 + BS_55AA, 0xAA55);
            ST_DWORD(fs->win.d8 + FSI_LeadSig, 0x41615252);
            ST_DWORD(fs->win.d8 + FSI_StrucSig, 0x61417272);
            ST_DWORD(fs->win.d8 + FSI_Free_Count, fs->free_clust);
            ST_DWORD(fs->win.d8 + FSI_Nxt_Free, fs->last_clust);
            /* Write it into the FSINFO sector */
            fs->winsect = fs->volbase + 1;
            disk_write(fs->drv, fs->win.d8, fs->winsect, 1);
            fs->fsi_flag = 0;
        }
        /* Make sure that no pending write process in the physical drive */
        if (disk_ioctl(fs->drv, CTRL_SYNC, 0) != RES_OK)
            res = FR_DISK_ERR;
    }

    return res;
}
#endif




/*-----------------------------------------------------------------------*/
/* Get sector# from cluster#                                             */
/*-----------------------------------------------------------------------*/
/* Hidden API for hacks and disk tools */

DWORD clust2sect (  /* !=0: Sector number, 0: Failed - invalid cluster# */
    FATFS* fs,      /* File system object */
    DWORD clst      /* Cluster# to be converted */
)
{
    clst -= 2;
    if (clst >= fs->n_fatent - 2) return 0;     /* Invalid cluster# */
    return clst * fs->csize + fs->database;
}




/*-----------------------------------------------------------------------*/
/* FAT access - Read value of a FAT entry                                */
/*-----------------------------------------------------------------------*/
/* Hidden API for hacks and disk tools */

DWORD get_fat ( /* 0xFFFFFFFF:Disk error, 1:Internal error, 2..0x0FFFFFFF:Cluster status */
    FATFS* fs,  /* File system object */
    DWORD clst  /* FAT index number (cluster number) to get the value */
)
{
    UINT wc, bc;
    BYTE *p;
    DWORD val;


    if (clst < 2 || clst >= fs->n_fatent) { /* Check range */
        val = 1;    /* Internal error */

    } else {
        val = 0xFFFFFFFF;   /* Default value falls on disk error */

        switch (fs->fs_type) {
        case FS_FAT12 :
            bc = (UINT)clst; bc += bc / 2;
            if (move_window(fs, fs->fatbase + (bc / SS(fs))) != FR_OK) break;
            wc = fs->win.d8[bc++ % SS(fs)];
            if (move_window(fs, fs->fatbase + (bc / SS(fs))) != FR_OK) break;
            wc |= fs->win.d8[bc % SS(fs)] << 8;
            val = clst & 1 ? wc >> 4 : (wc & 0xFFF);
            break;
```

```
        case FS_FAT16 :
            if (move_window(fs, fs->fatbase + (clst / (SS(fs) / 2))) != FR_OK) break;
            p = &fs->win.d8[clst * 2 % SS(fs)];
            val = LD_WORD(p);
            break;

        case FS_FAT32 :
            if (move_window(fs, fs->fatbase + (clst / (SS(fs) / 4))) != FR_OK) break;
            p = &fs->win.d8[clst * 4 % SS(fs)];
            val = LD_DWORD(p) & 0x0FFFFFFF;
            break;

        default:
            val = 1;    /* Internal error */
        }
    }

    return val;
}




/*-----------------------------------------------------------------------*/
/* FAT access - Change value of a FAT entry                              */
/*-----------------------------------------------------------------------*/
/* Hidden API for hacks and disk tools */

#if !_FS_READONLY
FRESULT put_fat (
    FATFS* fs,  /* File system object */
    DWORD clst, /* FAT index number (cluster number) to be changed */
    DWORD val   /* New value to be set to the entry */
)
{
    UINT bc;
    BYTE *p;
    FRESULT res;


    if (clst < 2 || clst >= fs->n_fatent) { /* Check range */
        res = FR_INT_ERR;

    } else {
        switch (fs->fs_type) {
        case FS_FAT12 :
            bc = (UINT)clst; bc += bc / 2;
            res = move_window(fs, fs->fatbase + (bc / SS(fs)));
            if (res != FR_OK) break;
            p = &fs->win.d8[bc++ % SS(fs)];
            *p = (clst & 1) ? ((*p & 0x0F) | ((BYTE)val << 4)) : (BYTE)val;
            fs->wflag = 1;
            res = move_window(fs, fs->fatbase + (bc / SS(fs)));
            if (res != FR_OK) break;
            p = &fs->win.d8[bc % SS(fs)];
            *p = (clst & 1) ? (BYTE)(val >> 4) : ((*p & 0xF0) | ((BYTE)(val >> 8) & 0x0F));
            fs->wflag = 1;
            break;

        case FS_FAT16 :
            res = move_window(fs, fs->fatbase + (clst / (SS(fs) / 2)));
            if (res != FR_OK) break;
            p = &fs->win.d8[clst * 2 % SS(fs)];
            ST_WORD(p, (WORD)val);
            fs->wflag = 1;
            break;

        case FS_FAT32 :
            res = move_window(fs, fs->fatbase + (clst / (SS(fs) / 4)));
            if (res != FR_OK) break;
            p = &fs->win.d8[clst * 4 % SS(fs)];
            val |= LD_DWORD(p) & 0xF0000000;
```

```
            ST_DWORD(p, val);
            fs->wflag = 1;
            break;

        default :
            res = FR_INT_ERR;
        }
    }

    return res;
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* FAT handling - Remove a cluster chain                                 */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT remove_chain (
    FATFS* fs,          /* File system object */
    DWORD clst          /* Cluster# to remove a chain from */
)
{
    FRESULT res;
    DWORD nxt;
#if _USE_TRIM
    DWORD scl = clst, ecl = clst, rt[2];
#endif

    if (clst < 2 || clst >= fs->n_fatent) { /* Check range */
        res = FR_INT_ERR;

    } else {
        res = FR_OK;
        while (clst < fs->n_fatent) {           /* Not a last link? */
            nxt = get_fat(fs, clst);            /* Get cluster status */
            if (nxt == 0) break;                /* Empty cluster? */
            if (nxt == 1) { res = FR_INT_ERR; break; }  /* Internal error? */
            if (nxt == 0xFFFFFFFF) { res = FR_DISK_ERR; break; }    /* Disk error? */
            res = put_fat(fs, clst, 0);         /* Mark the cluster "empty" */
            if (res != FR_OK) break;
            if (fs->free_clust != 0xFFFFFFFF) { /* Update FSINFO */
                fs->free_clust++;
                fs->fsi_flag |= 1;
            }
#if _USE_TRIM
            if (ecl + 1 == nxt) {   /* Is next cluster contiguous? */
                ecl = nxt;
            } else {                /* End of contiguous clusters */
                rt[0] = clust2sect(fs, scl);                    /* Start sector */
                rt[1] = clust2sect(fs, ecl) + fs->csize - 1;    /* End sector */
                disk_ioctl(fs->drv, CTRL_TRIM, rt);             /* Erase the block */
                scl = ecl = nxt;
            }
#endif
            clst = nxt; /* Next cluster */
        }
    }

    return res;
}
#endif




/*-----------------------------------------------------------------------*/
/* FAT handling - Stretch or Create a cluster chain                      */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
```

```
static
DWORD create_chain (    /* 0:No free cluster, 1:Internal error, 0xFFFFFFFF:Disk error,
>=2:New cluster# */
    FATFS* fs,          /* File system object */
    DWORD clst          /* Cluster# to stretch. 0 means create a new chain. */
)
{
    DWORD cs, ncl, scl;
    FRESULT res;


    if (clst == 0) {        /* Create a new chain */
        scl = fs->last_clust;           /* Get suggested start point */
        if (!scl || scl >= fs->n_fatent) scl = 1;
    }
    else {                  /* Stretch the current chain */
        cs = get_fat(fs, clst);         /* Check the cluster status */
        if (cs < 2) return 1;           /* Invalid value */
        if (cs == 0xFFFFFFFF) return cs;    /* A disk error occurred */
        if (cs < fs->n_fatent) return cs;   /* It is already followed by next cluster */
        scl = clst;
    }

    ncl = scl;              /* Start cluster */
    for (;;) {
        ncl++;                          /* Next cluster */
        if (ncl >= fs->n_fatent) {      /* Check wrap around */
            ncl = 2;
            if (ncl > scl) return 0;    /* No free cluster */
        }
        cs = get_fat(fs, ncl);          /* Get the cluster status */
        if (cs == 0) break;             /* Found a free cluster */
        if (cs == 0xFFFFFFFF || cs == 1)/* An error occurred */
            return cs;
        if (ncl == scl) return 0;       /* No free cluster */
    }

    res = put_fat(fs, ncl, 0x0FFFFFFF); /* Mark the new cluster "last link" */
    if (res == FR_OK && clst != 0) {
        res = put_fat(fs, clst, ncl);   /* Link it to the previous one if needed */
    }
    if (res == FR_OK) {
        fs->last_clust = ncl;           /* Update FSINFO */
        if (fs->free_clust != 0xFFFFFFFF) {
            fs->free_clust--;
            fs->fsi_flag |= 1;
        }
    } else {
        ncl = (res == FR_DISK_ERR) ? 0xFFFFFFFF : 1;
    }

    return ncl;     /* Return new cluster number or error code */
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* FAT handling - Convert offset into cluster with link map table        */
/*-----------------------------------------------------------------------*/

#if _USE_FASTSEEK
static
DWORD clmt_clust (  /* <2:Error, >=2:Cluster number */
    FIL* fp,        /* Pointer to the file object */
    DWORD ofs       /* File offset to be converted to cluster# */
)
{
    DWORD cl, ncl, *tbl;


    tbl = fp->cltbl + 1;    /* Top of CLMT */
```

```
    cl = ofs / SS(fp->fs) / fp->fs->csize;  /* Cluster order from top of the file */
    for (;;) {
        ncl = *tbl++;           /* Number of cluters in the fragment */
        if (!ncl) return 0;     /* End of table? (error) */
        if (cl < ncl) break;    /* In this fragment? */
        cl -= ncl; tbl++;       /* Next fragment */
    }
    return cl + *tbl;   /* Return the cluster number */
}
#endif  /* _USE_FASTSEEK */



/*-----------------------------------------------------------------------*/
/* Directory handling - Set directory index                              */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_sdi (
    DIR* dp,        /* Pointer to directory object */
    UINT idx        /* Index of directory table */
)
{
    DWORD clst, sect;
    UINT ic;


    dp->index = (WORD)idx;  /* Current index */
    clst = dp->sclust;      /* Table start cluster (0:root) */
    if (clst == 1 || clst >= dp->fs->n_fatent)  /* Check start cluster range */
        return FR_INT_ERR;
    if (!clst && dp->fs->fs_type == FS_FAT32)   /* Replace cluster# 0 with root cluster# if
    in FAT32 */
        clst = dp->fs->dirbase;

    if (clst == 0) {    /* Static table (root-directory in FAT12/16) */
        if (idx >= dp->fs->n_rootdir)   /* Is index out of range? */
            return FR_INT_ERR;
        sect = dp->fs->dirbase;
    }
    else {              /* Dynamic table (root-directory in FAT32 or sub-directory) */
        ic = SS(dp->fs) / SZ_DIRE * dp->fs->csize;  /* Entries per cluster */
        while (idx >= ic) { /* Follow cluster chain */
            clst = get_fat(dp->fs, clst);               /* Get next cluster */
            if (clst == 0xFFFFFFFF) return FR_DISK_ERR; /* Disk error */
            if (clst < 2 || clst >= dp->fs->n_fatent)   /* Reached to end of table or
            internal error */
                return FR_INT_ERR;
            idx -= ic;
        }
        sect = clust2sect(dp->fs, clst);
    }
    dp->clust = clst;   /* Current cluster# */
    if (!sect) return FR_INT_ERR;
    dp->sect = sect + idx / (SS(dp->fs) / SZ_DIRE);                 /* Sector# of the
    directory entry */
    dp->dir = dp->fs->win.d8 + (idx % (SS(dp->fs) / SZ_DIRE)) * SZ_DIRE;    /* Ptr to the
    entry in the sector */

    return FR_OK;
}



/*-----------------------------------------------------------------------*/
/* Directory handling - Move directory table index next                  */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_next (  /* FR_OK:Succeeded, FR_NO_FILE:End of table, FR_DENIED:Could not stretch
*/
```

```
    DIR* dp,        /* Pointer to the directory object */
    int stretch     /* 0: Do not stretch table, 1: Stretch table if needed */
)
{
    DWORD clst;
    UINT i;
#if !_FS_READONLY
    UINT c;
#endif


    i = dp->index + 1;
    if (!(i & 0xFFFF) || !dp->sect) /* Report EOT when index has reached 65535 */
        return FR_NO_FILE;

    if (!(i % (SS(dp->fs) / SZ_DIRE))) {    /* Sector changed? */
        dp->sect++;                 /* Next sector */

        if (!dp->clust) {       /* Static table */
            if (i >= dp->fs->n_rootdir) /* Report EOT if it reached end of static table */
                return FR_NO_FILE;
        }
        else {                  /* Dynamic table */
            if (((i / (SS(dp->fs) / SZ_DIRE)) & (dp->fs->csize - 1)) == 0) {    /* Cluster
            changed? */
                clst = get_fat(dp->fs, dp->clust);              /* Get next cluster */
                if (clst <= 1) return FR_INT_ERR;
                if (clst == 0xFFFFFFFF) return FR_DISK_ERR;
                if (clst >= dp->fs->n_fatent) {                 /* If it reached end of
                dynamic table, */
#if !_FS_READONLY
                    if (!stretch) return FR_NO_FILE;            /* If do not stretch, report
                    EOT */
                    clst = create_chain(dp->fs, dp->clust);     /* Stretch cluster chain */
                    if (clst == 0) return FR_DENIED;            /* No free cluster */
                    if (clst == 1) return FR_INT_ERR;
                    if (clst == 0xFFFFFFFF) return FR_DISK_ERR;
                    /* Clean-up stretched table */
                    if (sync_window(dp->fs)) return FR_DISK_ERR;/* Flush disk access window */
                    mem_set(dp->fs->win.d8, 0, SS(dp->fs));     /* Clear window buffer */
                    dp->fs->winsect = clust2sect(dp->fs, clst); /* Cluster start sector */
                    for (c = 0; c < dp->fs->csize; c++) {       /* Fill the new cluster with
                    0 */
                        dp->fs->wflag = 1;
                        if (sync_window(dp->fs)) return FR_DISK_ERR;
                        dp->fs->winsect++;
                    }
                    dp->fs->winsect -= c;                       /* Rewind window offset */
#else
                    if (!stretch) return FR_NO_FILE;            /* If do not stretch, report
                    EOT (this is to suppress warning) */
                    return FR_NO_FILE;                          /* Report EOT */
#endif
                }
                dp->clust = clst;               /* Initialize data for new cluster */
                dp->sect = clust2sect(dp->fs, clst);
            }
        }
    }

    dp->index = (WORD)i;    /* Current index */
    dp->dir = dp->fs->win.d8 + (i % (SS(dp->fs) / SZ_DIRE)) * SZ_DIRE;  /* Current entry in
    the window */

    return FR_OK;
}




/*-----------------------------------------------------------------------*/
/* Directory handling - Reserve directory entry                          */
/*-----------------------------------------------------------------------*/
```

```
#if !_FS_READONLY
static
FRESULT dir_alloc (
    DIR* dp,    /* Pointer to the directory object */
    UINT nent   /* Number of contiguous entries to allocate (1-21) */
)
{
    FRESULT res;
    UINT n;


    res = dir_sdi(dp, 0);
    if (res == FR_OK) {
        n = 0;
        do {
            res = move_window(dp->fs, dp->sect);
            if (res != FR_OK) break;
            if (dp->dir[0] == DDEM || dp->dir[0] == 0) {    /* Is it a free entry? */
                if (++n == nent) break; /* A block of contiguous free entries is found */
            } else {
                n = 0;                  /* Not a blank entry. Restart to search */
            }
            res = dir_next(dp, 1);      /* Next entry with table stretch enabled */
        } while (res == FR_OK);
    }
    if (res == FR_NO_FILE) res = FR_DENIED; /* No directory entry to allocate */
    return res;
}
#endif



/*-----------------------------------------------------------------------*/
/* Directory handling - Load/Store start cluster number                  */
/*-----------------------------------------------------------------------*/

static
DWORD ld_clust (
    FATFS* fs,  /* Pointer to the fs object */
    BYTE* dir   /* Pointer to the directory entry */
)
{
    DWORD cl;

    cl = LD_WORD(dir + DIR_FstClusLO);
    if (fs->fs_type == FS_FAT32)
        cl |= (DWORD)LD_WORD(dir + DIR_FstClusHI) << 16;

    return cl;
}


#if !_FS_READONLY
static
void st_clust (
    BYTE* dir,  /* Pointer to the directory entry */
    DWORD cl    /* Value to be set */
)
{
    ST_WORD(dir + DIR_FstClusLO, cl);
    ST_WORD(dir + DIR_FstClusHI, cl >> 16);
}
#endif



/*-----------------------------------------------------------------------*/
/* LFN handling - Test/Pick/Fit an LFN segment from/to directory entry   */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
```

```
static
const BYTE LfnOfs[] = {1,3,5,7,9,14,16,18,20,22,24,28,30};  /* Offset of LFN characters in
the directory entry */


static
int cmp_lfn (           /* 1:Matched, 0:Not matched */
    WCHAR* lfnbuf,      /* Pointer to the LFN to be compared */
    BYTE* dir           /* Pointer to the directory entry containing a part of LFN */
)
{
    UINT i, s;
    WCHAR wc, uc;


    i = ((dir[LDIR_Ord] & ~LLEF) - 1) * 13; /* Get offset in the LFN buffer */
    s = 0; wc = 1;
    do {
        uc = LD_WORD(dir + LfnOfs[s]);  /* Pick an LFN character from the entry */
        if (wc) {   /* Last character has not been processed */
            wc = ff_wtoupper(uc);       /* Convert it to upper case */
            if (i >= _MAX_LFN || wc != ff_wtoupper(lfnbuf[i++]))    /* Compare it */
                return 0;               /* Not matched */
        } else {
            if (uc != 0xFFFF) return 0; /* Check filler */
        }
    } while (++s < 13);             /* Repeat until all characters in the entry are checked */

    if ((dir[LDIR_Ord] & LLEF) && wc && lfnbuf[i])  /* Last segment matched but different
    length */
        return 0;

    return 1;                       /* The part of LFN matched */
}



static
int pick_lfn (          /* 1:Succeeded, 0:Buffer overflow */
    WCHAR* lfnbuf,      /* Pointer to the Unicode-LFN buffer */
    BYTE* dir           /* Pointer to the directory entry */
)
{
    UINT i, s;
    WCHAR wc, uc;


    i = ((dir[LDIR_Ord] & 0x3F) - 1) * 13;  /* Offset in the LFN buffer */

    s = 0; wc = 1;
    do {
        uc = LD_WORD(dir + LfnOfs[s]);      /* Pick an LFN character from the entry */
        if (wc) {   /* Last character has not been processed */
            if (i >= _MAX_LFN) return 0;    /* Buffer overflow? */
            lfnbuf[i++] = wc = uc;          /* Store it */
        } else {
            if (uc != 0xFFFF) return 0;     /* Check filler */
        }
    } while (++s < 13);                     /* Read all character in the entry */

    if (dir[LDIR_Ord] & LLEF) {             /* Put terminator if it is the last LFN part */
        if (i >= _MAX_LFN) return 0;        /* Buffer overflow? */
        lfnbuf[i] = 0;
    }

    return 1;
}


#if !_FS_READONLY
static
void fit_lfn (
    const WCHAR* lfnbuf,    /* Pointer to the LFN buffer */
```

```
    BYTE* dir,              /* Pointer to the directory entry */
    BYTE ord,               /* LFN order (1-20) */
    BYTE sum                /* SFN sum */
)
{
    UINT i, s;
    WCHAR wc;


    dir[LDIR_Chksum] = sum;         /* Set check sum */
    dir[LDIR_Attr] = AM_LFN;        /* Set attribute. LFN entry */
    dir[LDIR_Type] = 0;
    ST_WORD(dir + LDIR_FstClusLO, 0);

    i = (ord - 1) * 13;             /* Get offset in the LFN buffer */
    s = wc = 0;
    do {
        if (wc != 0xFFFF) wc = lfnbuf[i++]; /* Get an effective character */
        ST_WORD(dir+LfnOfs[s], wc); /* Put it */
        if (!wc) wc = 0xFFFF;       /* Padding characters following last character */
    } while (++s < 13);
    if (wc == 0xFFFF || !lfnbuf[i]) ord |= LLEF;    /* Bottom LFN part is the start of LFN
    sequence */
    dir[LDIR_Ord] = ord;            /* Set the LFN order */
}

#endif
#endif




/*-----------------------------------------------------------------------*/
/* Create numbered name                                                  */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
static
void gen_numname (
    BYTE* dst,          /* Pointer to the buffer to store numbered SFN */
    const BYTE* src,    /* Pointer to SFN */
    const WCHAR* lfn,   /* Pointer to LFN */
    UINT seq            /* Sequence number */
)
{
    BYTE ns[8], c;
    UINT i, j;
    WCHAR wc;
    DWORD sr;


    mem_cpy(dst, src, 11);

    if (seq > 5) {  /* On many collisions, generate a hash number instead of sequential
    number */
        sr = seq;
        while (*lfn) {  /* Create a CRC */
            wc = *lfn++;
            for (i = 0; i < 16; i++) {
                sr = (sr << 1) + (wc & 1);
                wc >>= 1;
                if (sr & 0x10000) sr ^= 0x11021;
            }
        }
        seq = (UINT)sr;
    }

    /* itoa (hexdecimal) */
    i = 7;
    do {
        c = (seq % 16) + '0';
        if (c > '9') c += 7;
        ns[i--] = c;
        seq /= 16;
```

```
    } while (seq);
    ns[i] = '~';

    /* Append the number */
    for (j = 0; j < i && dst[j] != ' '; j++) {
        if (IsDBCS1(dst[j])) {
            if (j == i - 1) break;
            j++;
        }
    }
    do {
        dst[j++] = (i < 8) ? ns[i++] : ' ';
    } while (j < 8);
}
#endif




/*-----------------------------------------------------------------------*/
/* Calculate sum of an SFN                                               */
/*-----------------------------------------------------------------------*/
#if _USE_LFN
static
BYTE sum_sfn (
    const BYTE* dir     /* Pointer to the SFN entry */
)
{
    BYTE sum = 0;
    UINT n = 11;

    do sum = (sum >> 1) + (sum << 7) + *dir++; while (--n);
    return sum;
}
#endif




/*-----------------------------------------------------------------------*/
/* Directory handling - Find an object in the directory                  */
/*-----------------------------------------------------------------------*/

static
FRESULT dir_find (
    DIR* dp         /* Pointer to the directory object linked to the file name */
)
{
    FRESULT res;
    BYTE c, *dir;
#if _USE_LFN
    BYTE a, ord, sum;
#endif

    res = dir_sdi(dp, 0);           /* Rewind directory object */
    if (res != FR_OK) return res;

#if _USE_LFN
    ord = sum = 0xFF; dp->lfn_idx = 0xFFFF; /* Reset LFN sequence */
#endif
    do {
        res = move_window(dp->fs, dp->sect);
        if (res != FR_OK) break;
        dir = dp->dir;                  /* Ptr to the directory entry of current index */
        c = dir[DIR_Name];
        if (c == 0) { res = FR_NO_FILE; break; }    /* Reached to end of table */
#if _USE_LFN    /* LFN configuration */
        a = dir[DIR_Attr] & AM_MASK;
        if (c == DDEM || ((a & AM_VOL) && a != AM_LFN)) {   /* An entry without valid data */
            ord = 0xFF; dp->lfn_idx = 0xFFFF;   /* Reset LFN sequence */
        } else {
            if (a == AM_LFN) {          /* An LFN entry is found */
                if (dp->lfn) {
```

```
                    if (c & LLEF) {     /* Is it start of LFN sequence? */
                        sum = dir[LDIR_Chksum];
                        c &= ~LLEF; ord = c;    /* LFN start order */
                        dp->lfn_idx = dp->index;    /* Start index of LFN */
                    }
                    /* Check validity of the LFN entry and compare it with given name */
                    ord = (c == ord && sum == dir[LDIR_Chksum] && cmp_lfn(dp->lfn, dir)) ?
                    ord - 1 : 0xFF;
                }
            } else {                    /* An SFN entry is found */
                if (!ord && sum == sum_sfn(dir)) break; /* LFN matched? */
                if (!(dp->fn[NSFLAG] & NS_LOSS) && !mem_cmp(dir, dp->fn, 11)) break;    /*
                SFN matched? */
                ord = 0xFF; dp->lfn_idx = 0xFFFF;   /* Reset LFN sequence */
            }
        }
#else       /* Non LFN configuration */
        if (!(dir[DIR_Attr] & AM_VOL) && !mem_cmp(dir, dp->fn, 11)) /* Is it a valid entry? */
            break;
#endif
        res = dir_next(dp, 0);      /* Next entry */
    } while (res == FR_OK);

    return res;
}




/*-----------------------------------------------------------------------*/
/* Read an object from the directory                                     */
/*-----------------------------------------------------------------------*/
#if _FS_MINIMIZE <= 1 || _USE_LABEL || _FS_RPATH >= 2
static
FRESULT dir_read (
    DIR* dp,        /* Pointer to the directory object */
    int vol         /* Filtered by 0:file/directory or 1:volume label */
)
{
    FRESULT res;
    BYTE a, c, *dir;
#if _USE_LFN
    BYTE ord = 0xFF, sum = 0xFF;
#endif

    res = FR_NO_FILE;
    while (dp->sect) {
        res = move_window(dp->fs, dp->sect);
        if (res != FR_OK) break;
        dir = dp->dir;                  /* Ptr to the directory entry of current index */
        c = dir[DIR_Name];
        if (c == 0) { res = FR_NO_FILE; break; }    /* Reached to end of table */
        a = dir[DIR_Attr] & AM_MASK;
#if _USE_LFN    /* LFN configuration */
        if (c == DDEM || (!_FS_RPATH && c == '.') || (int)((a & ~AM_ARC) == AM_VOL) != vol)
        {   /* An entry without valid data */
            ord = 0xFF;
        } else {
            if (a == AM_LFN) {          /* An LFN entry is found */
                if (c & LLEF) {         /* Is it start of LFN sequence? */
                    sum = dir[LDIR_Chksum];
                    c &= ~LLEF; ord = c;
                    dp->lfn_idx = dp->index;
                }
                /* Check LFN validity and capture it */
                ord = (c == ord && sum == dir[LDIR_Chksum] && pick_lfn(dp->lfn, dir)) ? ord
                - 1 : 0xFF;
            } else {                    /* An SFN entry is found */
                if (ord || sum != sum_sfn(dir)) /* Is there a valid LFN? */
                    dp->lfn_idx = 0xFFFF;       /* It has no LFN. */
                break;
            }
        }
```

```
#else       /* Non LFN configuration */
        if (c != DDEM && (_FS_RPATH || c != '.') && a != AM_LFN && (int)((a & ~AM_ARC) ==
        AM_VOL) == vol)    /* Is it a valid entry? */
            break;
#endif
        res = dir_next(dp, 0);              /* Next entry */
        if (res != FR_OK) break;
    }

    if (res != FR_OK) dp->sect = 0;

    return res;
}
#endif  /* _FS_MINIMIZE <= 1 || _USE_LABEL || _FS_RPATH >= 2 */




/*-----------------------------------------------------------------------*/
/* Register an object to the directory                                   */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY
static
FRESULT dir_register (  /* FR_OK:Successful, FR_DENIED:No free entry or too many SFN
collision, FR_DISK_ERR:Disk error */
    DIR* dp             /* Target directory with object name to be created */
)
{
    FRESULT res;
#if _USE_LFN    /* LFN configuration */
    UINT n, nent;
    BYTE sn[12], *fn, sum;
    WCHAR *lfn;


    fn = dp->fn; lfn = dp->lfn;
    mem_cpy(sn, fn, 12);

    if (_FS_RPATH && (sn[NSFLAG] & NS_DOT))     /* Cannot create dot entry */
        return FR_INVALID_NAME;

    if (sn[NSFLAG] & NS_LOSS) {         /* When LFN is out of 8.3 format, generate a
    numbered name */
        fn[NSFLAG] = 0; dp->lfn = 0;            /* Find only SFN */
        for (n = 1; n < 100; n++) {
            gen_numname(fn, sn, lfn, n);    /* Generate a numbered name */
            res = dir_find(dp);             /* Check if the name collides with existing SFN */
            if (res != FR_OK) break;
        }
        if (n == 100) return FR_DENIED;     /* Abort if too many collisions */
        if (res != FR_NO_FILE) return res;  /* Abort if the result is other than 'not
        collided' */
        fn[NSFLAG] = sn[NSFLAG]; dp->lfn = lfn;
    }

    if (sn[NSFLAG] & NS_LFN) {          /* When LFN is to be created, allocate entries for
    an SFN + LFNs. */
        for (n = 0; lfn[n]; n++) ;
        nent = (n + 25) / 13;
    } else {                        /* Otherwise allocate an entry for an SFN  */
        nent = 1;
    }
    res = dir_alloc(dp, nent);      /* Allocate entries */

    if (res == FR_OK && --nent) {   /* Set LFN entry if needed */
        res = dir_sdi(dp, dp->index - nent);
        if (res == FR_OK) {
            sum = sum_sfn(dp->fn);  /* Sum value of the SFN tied to the LFN */
            do {                    /* Store LFN entries in bottom first */
                res = move_window(dp->fs, dp->sect);
                if (res != FR_OK) break;
                fit_lfn(dp->lfn, dp->dir, (BYTE)nent, sum);
                dp->fs->wflag = 1;
```

```
                res = dir_next(dp, 0);  /* Next entry */
            } while (res == FR_OK && --nent);
        }
    }
#else   /* Non LFN configuration */
    res = dir_alloc(dp, 1);     /* Allocate an entry for SFN */
#endif

    if (res == FR_OK) {             /* Set SFN entry */
        res = move_window(dp->fs, dp->sect);
        if (res == FR_OK) {
            mem_set(dp->dir, 0, SZ_DIRE);   /* Clean the entry */
            mem_cpy(dp->dir, dp->fn, 11);   /* Put SFN */
#if _USE_LFN
            dp->dir[DIR_NTres] = dp->fn[NSFLAG] & (NS_BODY | NS_EXT);   /* Put NT flag */
#endif
            dp->fs->wflag = 1;
        }
    }

    return res;
}
#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* Remove an object from the directory                                   */
/*-----------------------------------------------------------------------*/
#if !_FS_READONLY && !_FS_MINIMIZE
static
FRESULT dir_remove (    /* FR_OK: Successful, FR_DISK_ERR: A disk error */
    DIR* dp             /* Directory object pointing the entry to be removed */
)
{
    FRESULT res;
#if _USE_LFN    /* LFN configuration */
    UINT i;

    i = dp->index;  /* SFN index */
    res = dir_sdi(dp, (dp->lfn_idx == 0xFFFF) ? i : dp->lfn_idx);   /* Goto the SFN or top
    of the LFN entries */
    if (res == FR_OK) {
        do {
            res = move_window(dp->fs, dp->sect);
            if (res != FR_OK) break;
            mem_set(dp->dir, 0, SZ_DIRE);   /* Clear and mark the entry "deleted" */
            *dp->dir = DDEM;
            dp->fs->wflag = 1;
            if (dp->index >= i) break;  /* When reached SFN, all entries of the object has
            been deleted. */
            res = dir_next(dp, 0);      /* Next entry */
        } while (res == FR_OK);
        if (res == FR_NO_FILE) res = FR_INT_ERR;
    }

#else           /* Non LFN configuration */
    res = dir_sdi(dp, dp->index);
    if (res == FR_OK) {
        res = move_window(dp->fs, dp->sect);
        if (res == FR_OK) {
            mem_set(dp->dir, 0, SZ_DIRE);   /* Clear and mark the entry "deleted" */
            *dp->dir = DDEM;
            dp->fs->wflag = 1;
        }
    }
#endif

    return res;
}
#endif /* !_FS_READONLY */
```

```
/*-----------------------------------------------------------------------*/
/* Get file information from directory entry                             */
/*-----------------------------------------------------------------------*/
#if _FS_MINIMIZE <= 1 || _FS_RPATH >= 2
static
void get_fileinfo (     /* No return code */
    DIR* dp,            /* Pointer to the directory object */
    FILINFO* fno        /* Pointer to the file information to be filled */
)
{
    UINT i;
    TCHAR *p, c;
    BYTE *dir;
#if _USE_LFN
    WCHAR w, *lfn;
#endif

    p = fno->fname;
    if (dp->sect) {     /* Get SFN */
        dir = dp->dir;
        i = 0;
        while (i < 11) {        /* Copy name body and extension */
            c = (TCHAR)dir[i++];
            if (c == ' ') continue;             /* Skip padding spaces */
            if (c == RDDEM) c = (TCHAR)DDEM;    /* Restore replaced DDEM character */
            if (i == 9) *p++ = '.';             /* Insert a . if extension is exist */
#if _USE_LFN
            if (IsUpper(c) && (dir[DIR_NTres] & (i >= 9 ? NS_EXT : NS_BODY)))
                c += 0x20;          /* To lower */
#if _LFN_UNICODE
            if (IsDBCS1(c) && i != 8 && i != 11 && IsDBCS2(dir[i]))
                c = c << 8 | dir[i++];
            c = ff_convert(c, 1);   /* OEM -> Unicode */
            if (!c) c = '?';
#endif
#endif
            *p++ = c;
        }
        fno->fattrib = dir[DIR_Attr];               /* Attribute */
        fno->fsize = LD_DWORD(dir + DIR_FileSize);  /* Size */
        fno->fdate = LD_WORD(dir + DIR_WrtDate);    /* Date */
        fno->ftime = LD_WORD(dir + DIR_WrtTime);    /* Time */
    }
    *p = 0;     /* Terminate SFN string by a \0 */

#if _USE_LFN
    if (fno->lfname) {
        i = 0; p = fno->lfname;
        if (dp->sect && fno->lfsize && dp->lfn_idx != 0xFFFF) { /* Get LFN if available */
            lfn = dp->lfn;
            while ((w = *lfn++) != 0) {     /* Get an LFN character */
#if !_LFN_UNICODE
                w = ff_convert(w, 0);       /* Unicode -> OEM */
                if (!w) { i = 0; break; }   /* No LFN if it could not be converted */
                if (_DF1S && w >= 0x100)    /* Put 1st byte if it is a DBC (always false on
                SBCS cfg) */
                    p[i++] = (TCHAR)(w >> 8);
#endif
                if (i >= fno->lfsize - 1) { i = 0; break; } /* No LFN if buffer overflow */
                p[i++] = (TCHAR)w;
            }
        }
        p[i] = 0;   /* Terminate LFN string by a \0 */
    }
#endif
}
#endif /* _FS_MINIMIZE <= 1 || _FS_RPATH >= 2 */
```

```
/*-----------------------------------------------------------------------*/
/* Pattern matching                                                      */
/*-----------------------------------------------------------------------*/
#if _USE_FIND && _FS_MINIMIZE <= 1
static
WCHAR get_achar (       /* Get a character and advances ptr 1 or 2 */
    const TCHAR** ptr   /* Pointer to pointer to the SBCS/DBCS/Unicode string */
)
{
    WCHAR chr;

#if !_LFN_UNICODE
    chr = (BYTE)*(*ptr)++;                  /* Get a byte */
    if (IsLower(chr)) chr -= 0x20;          /* To upper ASCII char */
    if (IsDBCS1(chr) && IsDBCS2(**ptr))     /* Get DBC 2nd byte if needed */
        chr = chr << 8 | (BYTE)*(*ptr)++;
#ifdef _EXCVT
    if (chr >= 0x80) chr = ExCvt[chr - 0x80];   /* To upper SBCS extended char */
#endif
#else
    chr = ff_wtoupper(*(*ptr)++);           /* Get a word and to upper */
#endif
    return chr;
}


static
int pattern_matching (  /* Return value: 0:mismatched, 1:matched */
    const TCHAR* pat,   /* Matching pattern */
    const TCHAR* nam,   /* String to be tested */
    int skip,           /* Number of pre-skip chars (number of ?s) */
    int inf             /* Infinite search (* specified) */
)
{
    const TCHAR *pp, *np;
    WCHAR pc, nc;
    int nm, nx;


    while (skip--) {                /* Pre-skip name chars */
        if (!get_achar(&nam)) return 0; /* Branch mismatched if less name chars */
    }
    if (!*pat && inf) return 1;     /* (short circuit) */

    do {
        pp = pat; np = nam;         /* Top of pattern and name to match */
        for (;;) {
            if (*pp == '?' || *pp == '*') { /* Wildcard? */
                nm = nx = 0;
                do {                /* Analyze the wildcard chars */
                    if (*pp++ == '?') nm++; else nx = 1;
                } while (*pp == '?' || *pp == '*');
                if (pattern_matching(pp, np, nm, nx)) return 1; /* Test new branch (recurs
                upto number of wildcard blocks in the pattern) */
                nc = *np; break;    /* Branch mismatched */
            }
            pc = get_achar(&pp);    /* Get a pattern char */
            nc = get_achar(&np);    /* Get a name char */
            if (pc != nc) break;    /* Branch mismatched? */
            if (!pc) return 1;      /* Branch matched? (matched at end of both strings) */
        }
        get_achar(&nam);            /* nam++ */
    } while (inf && nc);            /* Retry until end of name if infinite search is
    specified */

    return 0;
}
#endif /* _USE_FIND && _FS_MINIMIZE <= 1 */
```

```
/*-----------------------------------------------------------------------*/
/* Pick a segment and create the object name in directory form           */
/*-----------------------------------------------------------------------*/

static
FRESULT create_name (
    DIR* dp,            /* Pointer to the directory object */
    const TCHAR** path  /* Pointer to pointer to the segment in the path string */
)
{
#if _USE_LFN    /* LFN configuration */
    BYTE b, cf;
    WCHAR w, *lfn;
    UINT i, ni, si, di;
    const TCHAR *p;

    /* Create LFN in Unicode */
    for (p = *path; *p == '/' || *p == '\\'; p++) ; /* Strip duplicated separator */
    lfn = dp->lfn;
    si = di = 0;
    for (;;) {
        w = p[si++];                    /* Get a character */
        if (w < ' ' || w == '/' || w == '\\') break;    /* Break on end of segment */
        if (di >= _MAX_LFN)             /* Reject too long name */
            return FR_INVALID_NAME;
#if !_LFN_UNICODE
        w &= 0xFF;
        if (IsDBCS1(w)) {               /* Check if it is a DBC 1st byte (always false on
        SBCS cfg) */
#if _DF1S
            b = (BYTE)p[si++];          /* Get 2nd byte */
            w = (w << 8) + b;           /* Create a DBC */
            if (!IsDBCS2(b))
                return FR_INVALID_NAME; /* Reject invalid sequence */
#endif
        }
        w = ff_convert(w, 1);           /* Convert ANSI/OEM to Unicode */
        if (!w) return FR_INVALID_NAME; /* Reject invalid code */
#endif
        if (w < 0x80 && chk_chr("\"*:<>\?|\x7F", w)) /* Reject illegal characters for LFN */
            return FR_INVALID_NAME;
        lfn[di++] = w;                  /* Store the Unicode character */
    }
    *path = &p[si];                     /* Return pointer to the next segment */
    cf = (w < ' ') ? NS_LAST : 0;       /* Set last segment flag if end of path */
#if _FS_RPATH
    if ((di == 1 && lfn[di - 1] == '.') || /* Is this a dot entry? */
        (di == 2 && lfn[di - 1] == '.' && lfn[di - 2] == '.')) {
        lfn[di] = 0;
        for (i = 0; i < 11; i++)
            dp->fn[i] = (i < di) ? '.' : ' ';
        dp->fn[i] = cf | NS_DOT;        /* This is a dot entry */
        return FR_OK;
    }
#endif
    while (di) {                        /* Strip trailing spaces and dots */
        w = lfn[di - 1];
        if (w != ' ' && w != '.') break;
        di--;
    }
    if (!di) return FR_INVALID_NAME;    /* Reject nul string */

    lfn[di] = 0;                        /* LFN is created */

    /* Create SFN in directory form */
    mem_set(dp->fn, ' ', 11);
    for (si = 0; lfn[si] == ' ' || lfn[si] == '.'; si++) ;  /* Strip leading spaces and dots
    */
    if (si) cf |= NS_LOSS | NS_LFN;
    while (di && lfn[di - 1] != '.') di--;  /* Find extension (di<=si: no extension) */

    b = i = 0; ni = 8;
    for (;;) {
```

```
        w = lfn[si++];                  /* Get an LFN character */
        if (!w) break;                  /* Break on end of the LFN */
        if (w == ' ' || (w == '.' && si != di)) {   /* Remove spaces and dots */
            cf |= NS_LOSS | NS_LFN; continue;
        }

        if (i >= ni || si == di) {      /* Extension or end of SFN */
            if (ni == 11) {             /* Long extension */
                cf |= NS_LOSS | NS_LFN; break;
            }
            if (si != di) cf |= NS_LOSS | NS_LFN;   /* Out of 8.3 format */
            if (si > di) break;         /* No extension */
            si = di; i = 8; ni = 11;    /* Enter extension section */
            b <<= 2; continue;
        }

        if (w >= 0x80) {                /* Non ASCII character */
#ifdef _EXCVT
            w = ff_convert(w, 0);       /* Unicode -> OEM code */
            if (w) w = ExCvt[w - 0x80]; /* Convert extended character to upper (SBCS) */
#else
            w = ff_convert(ff_wtoupper(w), 0);  /* Upper converted Unicode -> OEM code */
#endif
            cf |= NS_LFN;               /* Force create LFN entry */
        }

        if (_DF1S && w >= 0x100) {      /* DBC (always false at SBCS cfg) */
            if (i >= ni - 1) {
                cf |= NS_LOSS | NS_LFN; i = ni; continue;
            }
            dp->fn[i++] = (BYTE)(w >> 8);
        } else {                        /* SBC */
            if (!w || chk_chr("+,;=[]", w)) {   /* Replace illegal characters for SFN */
                w = '_'; cf |= NS_LOSS | NS_LFN;/* Lossy conversion */
            } else {
                if (IsUpper(w)) {       /* ASCII large capital */
                    b |= 2;
                } else {
                    if (IsLower(w)) {   /* ASCII small capital */
                        b |= 1; w -= 0x20;
                    }
                }
            }
        }
        dp->fn[i++] = (BYTE)w;
    }

    if (dp->fn[0] == DDEM) dp->fn[0] = RDDEM;   /* If the first character collides with
    deleted mark, replace it with RDDEM */

    if (ni == 8) b <<= 2;
    if ((b & 0x0C) == 0x0C || (b & 0x03) == 0x03)   /* Create LFN entry when there are
    composite capitals */
        cf |= NS_LFN;
    if (!(cf & NS_LFN)) {                           /* When LFN is in 8.3 format without
    extended character, NT flags are created */
        if ((b & 0x03) == 0x01) cf |= NS_EXT;   /* NT flag (Extension has only small
        capital) */
        if ((b & 0x0C) == 0x04) cf |= NS_BODY;  /* NT flag (Filename has only small capital)
        */
    }

    dp->fn[NSFLAG] = cf;    /* SFN is created */

    return FR_OK;


#else   /* Non-LFN configuration */
    BYTE b, c, d, *sfn;
    UINT ni, si, i;
    const char *p;

    /* Create file name in directory form */
```

```
    for (p = *path; *p == '/' || *p == '\\'; p++) ; /* Strip duplicated separator */
    sfn = dp->fn;
    mem_set(sfn, ' ', 11);
    si = i = b = 0; ni = 8;
#if _FS_RPATH
    if (p[si] == '.') { /* Is this a dot entry? */
        for (;;) {
            c = (BYTE)p[si++];
            if (c != '.' || si >= 3) break;
            sfn[i++] = c;
        }
        if (c != '/' && c != '\\' && c > ' ') return FR_INVALID_NAME;
        *path = &p[si];                                     /* Return pointer to the next
        segment */
        sfn[NSFLAG] = (c <= ' ') ? NS_LAST | NS_DOT : NS_DOT;   /* Set last segment flag if
        end of path */
        return FR_OK;
    }
#endif
    for (;;) {
        c = (BYTE)p[si++];
        if (c <= ' ' || c == '/' || c == '\\') break;   /* Break on end of segment */
        if (c == '.' || i >= ni) {
            if (ni != 8 || c != '.') return FR_INVALID_NAME;
            i = 8; ni = 11;
            b <<= 2; continue;
        }
        if (c >= 0x80) {                /* Extended character? */
            b |= 3;                     /* Eliminate NT flag */
#ifdef _EXCVT
            c = ExCvt[c - 0x80];        /* To upper extended characters (SBCS cfg) */
#else
#if !_DF1S
            return FR_INVALID_NAME;     /* Reject extended characters (ASCII cfg) */
#endif
#endif
        }
        if (IsDBCS1(c)) {               /* Check if it is a DBC 1st byte (always false on
        SBCS cfg) */
            d = (BYTE)p[si++];          /* Get 2nd byte */
            if (!IsDBCS2(d) || i >= ni - 1) /* Reject invalid DBC */
                return FR_INVALID_NAME;
            sfn[i++] = c;
            sfn[i++] = d;
        } else {                        /* SBC */
            if (chk_chr("\"*+,:;<=>\?[]|\x7F", c))  /* Reject illegal chrs for SFN */
                return FR_INVALID_NAME;
            if (IsUpper(c)) {           /* ASCII large capital? */
                b |= 2;
            } else {
                if (IsLower(c)) {       /* ASCII small capital? */
                    b |= 1; c -= 0x20;
                }
            }
            sfn[i++] = c;
        }
    }
    *path = &p[si];                     /* Return pointer to the next segment */
    c = (c <= ' ') ? NS_LAST : 0;       /* Set last segment flag if end of path */

    if (!i) return FR_INVALID_NAME;     /* Reject nul string */
    if (sfn[0] == DDEM) sfn[0] = RDDEM; /* When first character collides with DDEM, replace
    it with RDDEM */

    if (ni == 8) b <<= 2;
    if ((b & 0x03) == 0x01) c |= NS_EXT;    /* NT flag (Name extension has only small
    capital) */
    if ((b & 0x0C) == 0x04) c |= NS_BODY;   /* NT flag (Name body has only small capital) */

    sfn[NSFLAG] = c;        /* Store NT flag, File name is created */

    return FR_OK;
#endif
```

```
}



/*-----------------------------------------------------------------------*/
/* Follow a file path                                                    */
/*-----------------------------------------------------------------------*/

static
FRESULT follow_path (   /* FR_OK(0): successful, !=0: error code */
    DIR* dp,            /* Directory object to return last directory and found object */
    const TCHAR* path   /* Full-path string to find a file or directory */
)
{
    FRESULT res;
    BYTE *dir, ns;


#if _FS_RPATH
    if (*path == '/' || *path == '\\') {    /* There is a heading separator */
        path++; dp->sclust = 0;             /* Strip it and start from the root directory */
    } else {                                /* No heading separator */
        dp->sclust = dp->fs->cdir;          /* Start from the current directory */
    }
#else
    if (*path == '/' || *path == '\\')      /* Strip heading separator if exist */
        path++;
    dp->sclust = 0;                         /* Always start from the root directory */
#endif

    if ((UINT)*path < ' ') {                /* Null path name is the origin directory itself
    */
        res = dir_sdi(dp, 0);
        dp->dir = 0;
    } else {                                /* Follow path */
        for (;;) {
            res = create_name(dp, &path);   /* Get a segment name of the path */
            if (res != FR_OK) break;
            res = dir_find(dp);             /* Find an object with the sagment name */
            ns = dp->fn[NSFLAG];
            if (res != FR_OK) {             /* Failed to find the object */
                if (res == FR_NO_FILE) {    /* Object is not found */
                    if (_FS_RPATH && (ns & NS_DOT)) {   /* If dot entry is not exist, */
                        dp->sclust = 0; dp->dir = 0;    /* it is the root directory and stay
                        there */
                        if (!(ns & NS_LAST)) continue;  /* Continue to follow if not last
                        segment */
                        res = FR_OK;                    /* Ended at the root directroy.
                        Function completed. */
                    } else {                            /* Could not find the object */
                        if (!(ns & NS_LAST)) res = FR_NO_PATH;  /* Adjust error code if not
                        last segment */
                    }
                }
                break;
            }
            if (ns & NS_LAST) break;            /* Last segment matched. Function completed.
            */
            dir = dp->dir;                      /* Follow the sub-directory */
            if (!(dir[DIR_Attr] & AM_DIR)) {    /* It is not a sub-directory and cannot
            follow */
                res = FR_NO_PATH; break;
            }
            dp->sclust = ld_clust(dp->fs, dir);
        }
    }

    return res;
}
```

```
/*-----------------------------------------------------------------------*/
/* Get logical drive number from path name                               */
/*-----------------------------------------------------------------------*/

static
int get_ldnumber (      /* Returns logical drive number (-1:invalid drive) */
    const TCHAR** path  /* Pointer to pointer to the path name */
)
{
    const TCHAR *tp, *tt;
    UINT i;
    int vol = -1;
#if _STR_VOLUME_ID      /* Find string drive id */
    static const char* const str[] = {_VOLUME_STRS};
    const char *sp;
    char c;
    TCHAR tc;
#endif


    if (*path) {    /* If the pointer is not a null */
        for (tt = *path; (UINT)*tt >= (_USE_LFN ? ' ' : '!') && *tt != ':'; tt++) ; /* Find
        ':' in the path */
        if (*tt == ':') {   /* If a ':' is exist in the path name */
            tp = *path;
            i = *tp++ - '0';
            if (i < 10 && tp == tt) {   /* Is there a numeric drive id? */
                if (i < _VOLUMES) { /* If a drive id is found, get the value and strip it */
                    vol = (int)i;
                    *path = ++tt;
                }
            }
#if _STR_VOLUME_ID
             else { /* No numeric drive number, find string drive id */
                i = 0; tt++;
                do {
                    sp = str[i]; tp = *path;
                    do {    /* Compare a string drive id with path name */
                        c = *sp++; tc = *tp++;
                        if (IsLower(tc)) tc -= 0x20;
                    } while (c && (TCHAR)c == tc);
                } while ((c || tp != tt) && ++i < _VOLUMES);    /* Repeat for each id until
                pattern match */
                if (i < _VOLUMES) { /* If a drive id is found, get the value and strip it */
                    vol = (int)i;
                    *path = tt;
                }
            }
#endif
            return vol;
        }
#if _FS_RPATH && _VOLUMES >= 2
        vol = CurrVol;  /* Current drive */
#else
        vol = 0;        /* Drive 0 */
#endif
    }
    return vol;
}




/*-----------------------------------------------------------------------*/
/* Load a sector and check if it is an FAT boot sector                   */
/*-----------------------------------------------------------------------*/

static
BYTE check_fs ( /* 0:FAT boor sector, 1:Valid boor sector but not FAT, 2:Not a boot sector,
3:Disk error */
    FATFS* fs,  /* File system object */
    DWORD sect  /* Sector# (lba) to check if it is an FAT boot record or not */
```

```
)
{
    fs->wflag = 0; fs->winsect = 0xFFFFFFFF;    /* Invaidate window */
    if (move_window(fs, sect) != FR_OK)         /* Load boot record */
        return 3;

    if (LD_WORD(&fs->win.d8[BS_55AA]) != 0xAA55)    /* Check boot record signature (always
    placed at offset 510 even if the sector size is >512) */
        return 2;

    if ((LD_DWORD(&fs->win.d8[BS_FilSysType]) & 0xFFFFFF) == 0x544146)      /* Check "FAT"
    string */
        return 0;
    if ((LD_DWORD(&fs->win.d8[BS_FilSysType32]) & 0xFFFFFF) == 0x544146)    /* Check "FAT"
    string */
        return 0;

    return 1;
}




/*-----------------------------------------------------------------------*/
/* Find logical drive and check if the volume is mounted                 */
/*-----------------------------------------------------------------------*/

static
FRESULT find_volume (   /* FR_OK(0): successful, !=0: any error occurred */
    FATFS** rfs,        /* Pointer to pointer to the found file system object */
    const TCHAR** path, /* Pointer to pointer to the path name (drive number) */
    BYTE wmode          /* !=0: Check write protection for write access */
)
{
    BYTE fmt, *pt;
    int vol;
    DSTATUS stat;
    DWORD bsect, fasize, tsect, sysect, nclst, szbfat, br[4];
    WORD nrsv;
    FATFS *fs;
    UINT i;


    /* Get logical drive number from the path name */
    *rfs = 0;
    vol = get_ldnumber(path);
    if (vol < 0) return FR_INVALID_DRIVE;

    /* Check if the file system object is valid or not */
    fs = FatFs[vol];                    /* Get pointer to the file system object */
    if (!fs) return FR_NOT_ENABLED;     /* Is the file system object available? */

    ENTER_FF(fs);                       /* Lock the volume */
    *rfs = fs;                          /* Return pointer to the file system object */

    if (fs->fs_type) {                  /* If the volume has been mounted */
        stat = disk_status(fs->drv);
        if (!(stat & STA_NOINIT)) {     /* and the physical drive is kept initialized */
            if (!_FS_READONLY && wmode && (stat & STA_PROTECT)) /* Check write protection if
            needed */
                return FR_WRITE_PROTECTED;
            return FR_OK;               /* The file system object is valid */
        }
    }

    /* The file system object is not valid. */
    /* Following code attempts to mount the volume. (analyze BPB and initialize the fs
    object) */

    fs->fs_type = 0;                    /* Clear the file system object */
    fs->drv = LD2PD(vol);               /* Bind the logical drive and a physical drive */
    stat = disk_initialize(fs->drv);    /* Initialize the physical drive */
    if (stat & STA_NOINIT)              /* Check if the initialization succeeded */
```

```
        return FR_NOT_READY;            /* Failed to initialize due to no medium or hard
        error */
    if (!_FS_READONLY && wmode && (stat & STA_PROTECT)) /* Check disk write protection if
    needed */
        return FR_WRITE_PROTECTED;
#if _MAX_SS != _MIN_SS                  /* Get sector size (multiple sector size cfg
only) */
    if (disk_ioctl(fs->drv, GET_SECTOR_SIZE, &SS(fs)) != RES_OK
        || SS(fs) < _MIN_SS || SS(fs) > _MAX_SS) return FR_DISK_ERR;
#endif
    /* Find an FAT partition on the drive. Supports only generic partitioning, FDISK and
    SFD. */
    bsect = 0;
    fmt = check_fs(fs, bsect);                  /* Load sector 0 and check if it is an FAT
    boot sector as SFD */
    if (fmt == 1 || (!fmt && (LD2PT(vol)))) {   /* Not an FAT boot sector or forced
    partition number */
        for (i = 0; i < 4; i++) {           /* Get partition offset */
            pt = fs->win.d8 + MBR_Table + i * SZ_PTE;
            br[i] = pt[4] ? LD_DWORD(&pt[8]) : 0;
        }
        i = LD2PT(vol);                     /* Partition number: 0:auto, 1-4:forced */
        if (i) i--;
        do {                                /* Find an FAT volume */
            bsect = br[i];
            fmt = bsect ? check_fs(fs, bsect) : 2;  /* Check the partition */
        } while (!LD2PT(vol) && fmt && ++i < 4);
    }
    if (fmt == 3) return FR_DISK_ERR;       /* An error occured in the disk I/O layer */
    if (fmt) return FR_NO_FILESYSTEM;       /* No FAT volume is found */

    /* An FAT volume is found. Following code initializes the file system object */

    if (LD_WORD(fs->win.d8 + BPB_BytsPerSec) != SS(fs)) /* (BPB_BytsPerSec must be equal to
    the physical sector size) */
        return FR_NO_FILESYSTEM;

    fasize = LD_WORD(fs->win.d8 + BPB_FATSz16);         /* Number of sectors per FAT */
    if (!fasize) fasize = LD_DWORD(fs->win.d8 + BPB_FATSz32);
    fs->fsize = fasize;

    fs->n_fats = fs->win.d8[BPB_NumFATs];                   /* Number of FAT copies */
    if (fs->n_fats != 1 && fs->n_fats != 2)             /* (Must be 1 or 2) */
        return FR_NO_FILESYSTEM;
    fasize *= fs->n_fats;                               /* Number of sectors for FAT area */

    fs->csize = fs->win.d8[BPB_SecPerClus];             /* Number of sectors per cluster */
    if (!fs->csize || (fs->csize & (fs->csize - 1)))    /* (Must be power of 2) */
        return FR_NO_FILESYSTEM;

    fs->n_rootdir = LD_WORD(fs->win.d8 + BPB_RootEntCnt);   /* Number of root directory
    entries */
    if (fs->n_rootdir % (SS(fs) / SZ_DIRE))             /* (Must be sector aligned) */
        return FR_NO_FILESYSTEM;

    tsect = LD_WORD(fs->win.d8 + BPB_TotSec16);         /* Number of sectors on the volume */
    if (!tsect) tsect = LD_DWORD(fs->win.d8 + BPB_TotSec32);

    nrsv = LD_WORD(fs->win.d8 + BPB_RsvdSecCnt);            /* Number of reserved sectors */
    if (!nrsv) return FR_NO_FILESYSTEM;                 /* (Must not be 0) */

    /* Determine the FAT sub type */
    sysect = nrsv + fasize + fs->n_rootdir / (SS(fs) / SZ_DIRE);    /* RSV + FAT + DIR */
    if (tsect < sysect) return FR_NO_FILESYSTEM;        /* (Invalid volume size) */
    nclst = (tsect - sysect) / fs->csize;               /* Number of clusters */
    if (!nclst) return FR_NO_FILESYSTEM;                /* (Invalid volume size) */
    fmt = FS_FAT12;
    if (nclst >= MIN_FAT16) fmt = FS_FAT16;
    if (nclst >= MIN_FAT32) fmt = FS_FAT32;

    /* Boundaries and Limits */
    fs->n_fatent = nclst + 2;                           /* Number of FAT entries */
    fs->volbase = bsect;                                /* Volume start sector */
```

```
    fs->fatbase = bsect + nrsv;                         /* FAT start sector */
    fs->database = bsect + sysect;                      /* Data start sector */
    if (fmt == FS_FAT32) {
        if (fs->n_rootdir) return FR_NO_FILESYSTEM;     /* (BPB_RootEntCnt must be 0) */
        fs->dirbase = LD_DWORD(fs->win.d8 + BPB_RootClus);  /* Root directory start cluster */
        szbfat = fs->n_fatent * 4;                      /* (Needed FAT size) */
    } else {
        if (!fs->n_rootdir) return FR_NO_FILESYSTEM;    /* (BPB_RootEntCnt must not be 0) */
        fs->dirbase = fs->fatbase + fasize;             /* Root directory start sector */
        szbfat = (fmt == FS_FAT16) ?                    /* (Needed FAT size) */
            fs->n_fatent * 2 : fs->n_fatent * 3 / 2 + (fs->n_fatent & 1);
    }
    if (fs->fsize < (szbfat + (SS(fs) - 1)) / SS(fs))   /* (BPB_FATSz must not be less than
    the size needed) */
        return FR_NO_FILESYSTEM;

#if !_FS_READONLY
    /* Initialize cluster allocation information */
    fs->last_clust = fs->free_clust = 0xFFFFFFFF;

    /* Get fsinfo if available */
    fs->fsi_flag = 0x80;
#if (_FS_NOFSINFO & 3) != 3
    if (fmt == FS_FAT32                 /* Enable FSINFO only if FAT32 and BPB_FSInfo is 1 */
        && LD_WORD(fs->win.d8 + BPB_FSInfo) == 1
        && move_window(fs, bsect + 1) == FR_OK)
    {
        fs->fsi_flag = 0;
        if (LD_WORD(fs->win.d8 + BS_55AA) == 0xAA55 /* Load FSINFO data if available */
            && LD_DWORD(fs->win.d8 + FSI_LeadSig) == 0x41615252
            && LD_DWORD(fs->win.d8 + FSI_StrucSig) == 0x61417272)
        {
#if (_FS_NOFSINFO & 1) == 0
            fs->free_clust = LD_DWORD(fs->win.d8 + FSI_Free_Count);
#endif
#if (_FS_NOFSINFO & 2) == 0
            fs->last_clust = LD_DWORD(fs->win.d8 + FSI_Nxt_Free);
#endif
        }
    }
#endif
#endif
    fs->fs_type = fmt;  /* FAT sub-type */
    fs->id = ++Fsid;    /* File system mount ID */
#if _FS_RPATH
    fs->cdir = 0;       /* Set current directory to root */
#endif
#if _FS_LOCK            /* Clear file lock semaphores */
    clear_lock(fs);
#endif

    return FR_OK;
}




/*-----------------------------------------------------------------------*/
/* Check if the file/directory object is valid or not                    */
/*-----------------------------------------------------------------------*/

static
FRESULT validate (  /* FR_OK(0): The object is valid, !=0: Invalid */
    void* obj       /* Pointer to the object FIL/DIR to check validity */
)
{
    FIL *fil = (FIL*)obj;   /* Assuming offset of .fs and .id in the FIL/DIR structure is
    identical */


    if (!fil || !fil->fs || !fil->fs->fs_type || fil->fs->id != fil->id ||
    (disk_status(fil->fs->drv) & STA_NOINIT))
        return FR_INVALID_OBJECT;
```

```
    ENTER_FF(fil->fs);      /* Lock file system */

    return FR_OK;
}



/*-----------------------------------------------------------------------

   Public Functions

-----------------------------------------------------------------------*/



/*-----------------------------------------------------------------------*/
/* Mount/Unmount a Logical Drive                                         */
/*-----------------------------------------------------------------------*/

FRESULT f_mount (
    FATFS* fs,          /* Pointer to the file system object (NULL:unmount)*/
    const TCHAR* path,  /* Logical drive number to be mounted/unmounted */
    BYTE opt            /* 0:Do not mount (delayed mount), 1:Mount immediately */
)
{
    FATFS *cfs;
    int vol;
    FRESULT res;
    const TCHAR *rp = path;


    vol = get_ldnumber(&rp);
    if (vol < 0) return FR_INVALID_DRIVE;
    cfs = FatFs[vol];                   /* Pointer to fs object */

    if (cfs) {
#if _FS_LOCK
        clear_lock(cfs);
#endif
#if _FS_REENTRANT                       /* Discard sync object of the current volume */
        if (!ff_del_syncobj(cfs->sobj)) return FR_INT_ERR;
#endif
        cfs->fs_type = 0;               /* Clear old fs object */
    }

    if (fs) {
        fs->fs_type = 0;                /* Clear new fs object */
#if _FS_REENTRANT                       /* Create sync object for the new volume */
        if (!ff_cre_syncobj((BYTE)vol, &fs->sobj)) return FR_INT_ERR;
#endif
    }
    FatFs[vol] = fs;                    /* Register new fs object */

    if (!fs || opt != 1) return FR_OK;  /* Do not mount now, it will be mounted later */

    res = find_volume(&fs, &path, 0);   /* Force mounted the volume */
    LEAVE_FF(fs, res);
}



/*-----------------------------------------------------------------------*/
/* Open or Create a File                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_open (
    FIL* fp,            /* Pointer to the blank file object */
    const TCHAR* path,  /* Pointer to the file name */
    BYTE mode           /* Access mode and file open mode flags */
)
```

```
{
    FRESULT res;
    DIR dj;
    BYTE *dir;
    DEFINE_NAMEBUF;
#if !_FS_READONLY
    DWORD dw, cl;
#endif


    if (!fp) return FR_INVALID_OBJECT;
    fp->fs = 0;         /* Clear file object */

    /* Get logical drive number */
#if !_FS_READONLY
    mode &= FA_READ | FA_WRITE | FA_CREATE_ALWAYS | FA_OPEN_ALWAYS | FA_CREATE_NEW;
    res = find_volume(&dj.fs, &path, (BYTE)(mode & ~FA_READ));
#else
    mode &= FA_READ;
    res = find_volume(&dj.fs, &path, 0);
#endif
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);   /* Follow the file path */
        dir = dj.dir;
#if !_FS_READONLY   /* R/W configuration */
        if (res == FR_OK) {
            if (!dir)   /* Default directory itself */
                res = FR_INVALID_NAME;
#if _FS_LOCK
            else
                res = chk_lock(&dj, (mode & ~FA_READ) ? 1 : 0);
#endif
        }
        /* Create or Open a file */
        if (mode & (FA_CREATE_ALWAYS | FA_OPEN_ALWAYS | FA_CREATE_NEW)) {
            if (res != FR_OK) {                 /* No file, create new */
                if (res == FR_NO_FILE)          /* There is no file to open, create a new
                entry */
#if _FS_LOCK
                    res = enq_lock() ? dir_register(&dj) : FR_TOO_MANY_OPEN_FILES;
#else
                    res = dir_register(&dj);
#endif
                mode |= FA_CREATE_ALWAYS;       /* File is created */
                dir = dj.dir;                   /* New entry */
            }
            else {                              /* Any object is already existing */
                if (dir[DIR_Attr] & (AM_RDO | AM_DIR)) {    /* Cannot overwrite it (R/O or
                DIR) */
                    res = FR_DENIED;
                } else {
                    if (mode & FA_CREATE_NEW)   /* Cannot create as new file */
                        res = FR_EXIST;
                }
            }
            if (res == FR_OK && (mode & FA_CREATE_ALWAYS)) {    /* Truncate it if overwrite
            mode */
                dw = GET_FATTIME();             /* Created time */
                ST_DWORD(dir + DIR_CrtTime, dw);
                dir[DIR_Attr] = 0;              /* Reset attribute */
                ST_DWORD(dir + DIR_FileSize, 0);/* size = 0 */
                cl = ld_clust(dj.fs, dir);      /* Get start cluster */
                st_clust(dir, 0);               /* cluster = 0 */
                dj.fs->wflag = 1;
                if (cl) {                       /* Remove the cluster chain if exist */
                    dw = dj.fs->winsect;
                    res = remove_chain(dj.fs, cl);
                    if (res == FR_OK) {
                        dj.fs->last_clust = cl - 1; /* Reuse the cluster hole */
                        res = move_window(dj.fs, dw);
                    }
                }
```

```
            }
        }
        else {  /* Open an existing file */
            if (res == FR_OK) {                 /* Follow succeeded */
                if (dir[DIR_Attr] & AM_DIR) {   /* It is a directory */
                    res = FR_NO_FILE;
                } else {
                    if ((mode & FA_WRITE) && (dir[DIR_Attr] & AM_RDO)) /* R/O violation */
                        res = FR_DENIED;
                }
            }
        }
        if (res == FR_OK) {
            if (mode & FA_CREATE_ALWAYS)        /* Set file change flag if created or
            overwritten */
                mode |= FA__WRITTEN;
            fp->dir_sect = dj.fs->winsect;      /* Pointer to the directory entry */
            fp->dir_ptr = dir;
#if _FS_LOCK
            fp->lockid = inc_lock(&dj, (mode & ~FA_READ) ? 1 : 0);
            if (!fp->lockid) res = FR_INT_ERR;
#endif
        }

#else               /* R/O configuration */
        if (res == FR_OK) {                 /* Follow succeeded */
            dir = dj.dir;
            if (!dir) {                     /* Current directory itself */
                res = FR_INVALID_NAME;
            } else {
                if (dir[DIR_Attr] & AM_DIR) /* It is a directory */
                    res = FR_NO_FILE;
            }
        }
#endif
        FREE_BUF();

        if (res == FR_OK) {
            fp->flag = mode;                    /* File access mode */
            fp->err = 0;                        /* Clear error flag */
            fp->sclust = ld_clust(dj.fs, dir);  /* File start cluster */
            fp->fsize = LD_DWORD(dir + DIR_FileSize);   /* File size */
            fp->fptr = 0;                       /* File pointer */
            fp->dsect = 0;
#if _USE_FASTSEEK
            fp->cltbl = 0;                      /* Normal seek mode */
#endif
            fp->fs = dj.fs;                     /* Validate file object */
            fp->id = fp->fs->id;
        }
    }

    LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Read File                                                             */
/*-----------------------------------------------------------------------*/

FRESULT f_read (
    FIL* fp,        /* Pointer to the file object */
    void* buff,     /* Pointer to data buffer */
    UINT btr,       /* Number of bytes to read */
    UINT* br        /* Pointer to number of bytes read */
)
{
    FRESULT res;
    DWORD clst, sect, remain;
    UINT rcnt, cc;
    BYTE csect, *rbuff = (BYTE*)buff;
```

```
    *br = 0;    /* Clear read byte counter */

    res = validate(fp);                         /* Check validity */
    if (res != FR_OK) LEAVE_FF(fp->fs, res);
    if (fp->err)                                /* Check error */
        LEAVE_FF(fp->fs, (FRESULT)fp->err);
    if (!(fp->flag & FA_READ))                  /* Check access mode */
        LEAVE_FF(fp->fs, FR_DENIED);
    remain = fp->fsize - fp->fptr;
    if (btr > remain) btr = (UINT)remain;       /* Truncate btr by remaining bytes */

    for ( ;  btr;                               /* Repeat until all data read */
        rbuff += rcnt, fp->fptr += rcnt, *br += rcnt, btr -= rcnt) {
        if ((fp->fptr % SS(fp->fs)) == 0) {     /* On the sector boundary? */
            csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));    /* Sector offset
            in the cluster */
            if (!csect) {                       /* On the cluster boundary? */
                if (fp->fptr == 0) {            /* On the top of the file? */
                    clst = fp->sclust;          /* Follow from the origin */
                } else {                        /* Middle or end of the file */
#if _USE_FASTSEEK
                    if (fp->cltbl)
                        clst = clmt_clust(fp, fp->fptr);    /* Get cluster# from the CLMT */
                    else
#endif
                        clst = get_fat(fp->fs, fp->clust);  /* Follow cluster chain on the
                        FAT */
                }
                if (clst < 2) ABORT(fp->fs, FR_INT_ERR);
                if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                fp->clust = clst;               /* Update current cluster */
            }
            sect = clust2sect(fp->fs, fp->clust);   /* Get current sector */
            if (!sect) ABORT(fp->fs, FR_INT_ERR);
            sect += csect;
            cc = btr / SS(fp->fs);              /* When remaining bytes >= sector size, */
            if (cc) {                           /* Read maximum contiguous sectors directly */
                if (csect + cc > fp->fs->csize) /* Clip at cluster boundary */
                    cc = fp->fs->csize - csect;
                if (disk_read(fp->fs->drv, rbuff, sect, cc) != RES_OK)
                    ABORT(fp->fs, FR_DISK_ERR);
#if !_FS_READONLY && _FS_MINIMIZE <= 2          /* Replace one of the read sectors with
cached data if it contains a dirty sector */
#if _FS_TINY
                if (fp->fs->wflag && fp->fs->winsect - sect < cc)
                    mem_cpy(rbuff + ((fp->fs->winsect - sect) * SS(fp->fs)), fp->fs->win.d8,
                    SS(fp->fs));
#else
                if ((fp->flag & FA__DIRTY) && fp->dsect - sect < cc)
                    mem_cpy(rbuff + ((fp->dsect - sect) * SS(fp->fs)), fp->buf.d8,
                    SS(fp->fs));
#endif
#endif
                rcnt = SS(fp->fs) * cc;         /* Number of bytes transferred */
                continue;
            }
#if !_FS_TINY
            if (fp->dsect != sect) {            /* Load data sector if not in cache */
#if !_FS_READONLY
                if (fp->flag & FA__DIRTY) {     /* Write-back dirty sector cache */
                    if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                        ABORT(fp->fs, FR_DISK_ERR);
                    fp->flag &= ~FA__DIRTY;
                }
#endif
                if (disk_read(fp->fs->drv, fp->buf.d8, sect, 1) != RES_OK)  /* Fill sector
                cache */
                    ABORT(fp->fs, FR_DISK_ERR);
            }
#endif
            fp->dsect = sect;
```

```
        }
        rcnt = SS(fp->fs) - ((UINT)fp->fptr % SS(fp->fs));  /* Get partial sector data from
        sector buffer */
        if (rcnt > btr) rcnt = btr;
#if _FS_TINY
        if (move_window(fp->fs, fp->dsect) != FR_OK)        /* Move sector window */
            ABORT(fp->fs, FR_DISK_ERR);
        mem_cpy(rbuff, &fp->fs->win.d8[fp->fptr % SS(fp->fs)], rcnt);   /* Pick partial
        sector */
#else
        mem_cpy(rbuff, &fp->buf.d8[fp->fptr % SS(fp->fs)], rcnt);   /* Pick partial sector */
#endif
    }

    LEAVE_FF(fp->fs, FR_OK);
}




#if !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Write File                                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_write (
    FIL* fp,            /* Pointer to the file object */
    const void *buff,   /* Pointer to the data to be written */
    UINT btw,           /* Number of bytes to write */
    UINT* bw            /* Pointer to number of bytes written */
)
{
    FRESULT res;
    DWORD clst, sect;
    UINT wcnt, cc;
    const BYTE *wbuff = (const BYTE*)buff;
    BYTE csect;


    *bw = 0;    /* Clear write byte counter */

    res = validate(fp);                     /* Check validity */
    if (res != FR_OK) LEAVE_FF(fp->fs, res);
    if (fp->err)                            /* Check error */
        LEAVE_FF(fp->fs, (FRESULT)fp->err);
    if (!(fp->flag & FA_WRITE))             /* Check access mode */
        LEAVE_FF(fp->fs, FR_DENIED);
    if (fp->fptr + btw < fp->fptr) btw = 0; /* File size cannot reach 4GB */

    for ( ;  btw;                           /* Repeat until all data written */
        wbuff += wcnt, fp->fptr += wcnt, *bw += wcnt, btw -= wcnt) {
        if ((fp->fptr % SS(fp->fs)) == 0) { /* On the sector boundary? */
            csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));    /* Sector offset
            in the cluster */
            if (!csect) {                   /* On the cluster boundary? */
                if (fp->fptr == 0) {        /* On the top of the file? */
                    clst = fp->sclust;      /* Follow from the origin */
                    if (clst == 0)          /* When no cluster is allocated, */
                        clst = create_chain(fp->fs, 0); /* Create a new cluster chain */
                } else {                    /* Middle or end of the file */
#if _USE_FASTSEEK
                    if (fp->cltbl)
                        clst = clmt_clust(fp, fp->fptr);    /* Get cluster# from the CLMT */
                    else
#endif
                        clst = create_chain(fp->fs, fp->clust); /* Follow or stretch cluster
                        chain on the FAT */
                }
                if (clst == 0) break;       /* Could not allocate a new cluster (disk full) */
                if (clst == 1) ABORT(fp->fs, FR_INT_ERR);
                if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                fp->clust = clst;           /* Update current cluster */
                if (fp->sclust == 0) fp->sclust = clst; /* Set start cluster if the first
```

```
                write */
            }
#if _FS_TINY
            if (fp->fs->winsect == fp->dsect && sync_window(fp->fs))    /* Write-back sector
            cache */
                ABORT(fp->fs, FR_DISK_ERR);
#else
            if (fp->flag & FA__DIRTY) {     /* Write-back sector cache */
                if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                    ABORT(fp->fs, FR_DISK_ERR);
                fp->flag &= ~FA__DIRTY;
            }
#endif
            sect = clust2sect(fp->fs, fp->clust);   /* Get current sector */
            if (!sect) ABORT(fp->fs, FR_INT_ERR);
            sect += csect;
            cc = btw / SS(fp->fs);          /* When remaining bytes >= sector size, */
            if (cc) {                       /* Write maximum contiguous sectors directly */
                if (csect + cc > fp->fs->csize) /* Clip at cluster boundary */
                    cc = fp->fs->csize - csect;
                if (disk_write(fp->fs->drv, wbuff, sect, cc) != RES_OK)
                    ABORT(fp->fs, FR_DISK_ERR);
#if _FS_MINIMIZE <= 2
#if _FS_TINY
                if (fp->fs->winsect - sect < cc) {  /* Refill sector cache if it gets
                invalidated by the direct write */
                    mem_cpy(fp->fs->win.d8, wbuff + ((fp->fs->winsect - sect) * SS(fp->fs)),
                    SS(fp->fs));
                    fp->fs->wflag = 0;
                }
#else
                if (fp->dsect - sect < cc) { /* Refill sector cache if it gets invalidated
                by the direct write */
                    mem_cpy(fp->buf.d8, wbuff + ((fp->dsect - sect) * SS(fp->fs)),
                    SS(fp->fs));
                    fp->flag &= ~FA__DIRTY;
                }
#endif
#endif
                wcnt = SS(fp->fs) * cc;     /* Number of bytes transferred */
                continue;
            }
#if _FS_TINY
            if (fp->fptr >= fp->fsize) {    /* Avoid silly cache filling at growing edge */
                if (sync_window(fp->fs)) ABORT(fp->fs, FR_DISK_ERR);
                fp->fs->winsect = sect;
            }
#else
            if (fp->dsect != sect) {        /* Fill sector cache with file data */
                if (fp->fptr < fp->fsize &&
                    disk_read(fp->fs->drv, fp->buf.d8, sect, 1) != RES_OK)
                        ABORT(fp->fs, FR_DISK_ERR);
            }
#endif
            fp->dsect = sect;
        }
        wcnt = SS(fp->fs) - ((UINT)fp->fptr % SS(fp->fs));/* Put partial sector into file
        I/O buffer */
        if (wcnt > btw) wcnt = btw;
#if _FS_TINY
        if (move_window(fp->fs, fp->dsect) != FR_OK)    /* Move sector window */
            ABORT(fp->fs, FR_DISK_ERR);
        mem_cpy(&fp->fs->win.d8[fp->fptr % SS(fp->fs)], wbuff, wcnt);   /* Fit partial
        sector */
        fp->fs->wflag = 1;
#else
        mem_cpy(&fp->buf.d8[fp->fptr % SS(fp->fs)], wbuff, wcnt);   /* Fit partial sector */
        fp->flag |= FA__DIRTY;
#endif
    }

    if (fp->fptr > fp->fsize) fp->fsize = fp->fptr; /* Update file size if needed */
    fp->flag |= FA__WRITTEN;                        /* Set file change flag */
```

```
    LEAVE_FF(fp->fs, FR_OK);
}




/*-----------------------------------------------------------------------*/
/* Synchronize the File                                                  */
/*-----------------------------------------------------------------------*/

FRESULT f_sync (
    FIL* fp     /* Pointer to the file object */
)
{
    FRESULT res;
    DWORD tm;
    BYTE *dir;


    res = validate(fp);                 /* Check validity of the object */
    if (res == FR_OK) {
        if (fp->flag & FA__WRITTEN) {   /* Has the file been written? */
            /* Write-back dirty buffer */
#if !_FS_TINY
            if (fp->flag & FA__DIRTY) {
                if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                    LEAVE_FF(fp->fs, FR_DISK_ERR);
                fp->flag &= ~FA__DIRTY;
            }
#endif
            /* Update the directory entry */
            res = move_window(fp->fs, fp->dir_sect);
            if (res == FR_OK) {
                dir = fp->dir_ptr;
                dir[DIR_Attr] |= AM_ARC;                    /* Set archive bit */
                ST_DWORD(dir + DIR_FileSize, fp->fsize);    /* Update file size */
                st_clust(dir, fp->sclust);                  /* Update start cluster */
                tm = GET_FATTIME();                         /* Update updated time */
                ST_DWORD(dir + DIR_WrtTime, tm);
                ST_WORD(dir + DIR_LstAccDate, 0);
                fp->flag &= ~FA__WRITTEN;
                fp->fs->wflag = 1;
                res = sync_fs(fp->fs);
            }
        }
    }

    LEAVE_FF(fp->fs, res);
}

#endif /* !_FS_READONLY */




/*-----------------------------------------------------------------------*/
/* Close File                                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_close (
    FIL *fp     /* Pointer to the file object to be closed */
)
{
    FRESULT res;


#if !_FS_READONLY
    res = f_sync(fp);                   /* Flush cached data */
    if (res == FR_OK)
#endif
    {
        res = validate(fp);             /* Lock volume */
```

```
        if (res == FR_OK) {
#if _FS_REENTRANT
            FATFS *fs = fp->fs;
#endif
#if _FS_LOCK
            res = dec_lock(fp->lockid); /* Decrement file open counter */
            if (res == FR_OK)
#endif
                fp->fs = 0;             /* Invalidate file object */
#if _FS_REENTRANT
            unlock_fs(fs, FR_OK);       /* Unlock volume */
#endif
        }
    }
    return res;
}




/*-----------------------------------------------------------------------*/
/* Change Current Directory or Current Drive, Get Current Directory      */
/*-----------------------------------------------------------------------*/

#if _FS_RPATH >= 1
#if _VOLUMES >= 2
FRESULT f_chdrive (
    const TCHAR* path       /* Drive number */
)
{
    int vol;


    vol = get_ldnumber(&path);
    if (vol < 0) return FR_INVALID_DRIVE;

    CurrVol = (BYTE)vol;

    return FR_OK;
}
#endif


FRESULT f_chdir (
    const TCHAR* path   /* Pointer to the directory path */
)
{
    FRESULT res;
    DIR dj;
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 0);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);       /* Follow the path */
        FREE_BUF();
        if (res == FR_OK) {                 /* Follow completed */
            if (!dj.dir) {
                dj.fs->cdir = dj.sclust;    /* Start directory itself */
            } else {
                if (dj.dir[DIR_Attr] & AM_DIR)  /* Reached to the directory */
                    dj.fs->cdir = ld_clust(dj.fs, dj.dir);
                else
                    res = FR_NO_PATH;       /* Reached but a file */
            }
        }
        if (res == FR_NO_FILE) res = FR_NO_PATH;
    }

    LEAVE_FF(dj.fs, res);
}
```

```
#if _FS_RPATH >= 2
FRESULT f_getcwd (
    TCHAR* buff,    /* Pointer to the directory path */
    UINT len        /* Size of path */
)
{
    FRESULT res;
    DIR dj;
    UINT i, n;
    DWORD ccl;
    TCHAR *tp;
    FILINFO fno;
    DEFINE_NAMEBUF;


    *buff = 0;
    /* Get logical drive number */
    res = find_volume(&dj.fs, (const TCHAR**)&buff, 0); /* Get current volume */
    if (res == FR_OK) {
        INIT_BUF(dj);
        i = len;            /* Bottom of buffer (directory stack base) */
        dj.sclust = dj.fs->cdir;            /* Start to follow upper directory from current
        directory */
        while ((ccl = dj.sclust) != 0) {    /* Repeat while current directory is a
        sub-directory */
            res = dir_sdi(&dj, 1);          /* Get parent directory */
            if (res != FR_OK) break;
            res = dir_read(&dj, 0);
            if (res != FR_OK) break;
            dj.sclust = ld_clust(dj.fs, dj.dir);    /* Goto parent directory */
            res = dir_sdi(&dj, 0);
            if (res != FR_OK) break;
            do {                            /* Find the entry links to the child directory */
                res = dir_read(&dj, 0);
                if (res != FR_OK) break;
                if (ccl == ld_clust(dj.fs, dj.dir)) break;  /* Found the entry */
                res = dir_next(&dj, 0);
            } while (res == FR_OK);
            if (res == FR_NO_FILE) res = FR_INT_ERR;/* It cannot be 'not found'. */
            if (res != FR_OK) break;
#if _USE_LFN
            fno.lfname = buff;
            fno.lfsize = i;
#endif
            get_fileinfo(&dj, &fno);        /* Get the directory name and push it to the
            buffer */
            tp = fno.fname;
#if _USE_LFN
            if (*buff) tp = buff;
#endif
            for (n = 0; tp[n]; n++) ;
            if (i < n + 3) {
                res = FR_NOT_ENOUGH_CORE; break;
            }
            while (n) buff[--i] = tp[--n];
            buff[--i] = '/';
        }
        tp = buff;
        if (res == FR_OK) {
#if _VOLUMES >= 2
            *tp++ = '0' + CurrVol;          /* Put drive number */
            *tp++ = ':';
#endif
            if (i == len) {                 /* Root-directory */
                *tp++ = '/';
            } else {                        /* Sub-directroy */
                do      /* Add stacked path str */
                    *tp++ = buff[i++];
                while (i < len);
            }
        }
```

```
        *tp = 0;
        FREE_BUF();
    }

    LEAVE_FF(dj.fs, res);
}
#endif /* _FS_RPATH >= 2 */
#endif /* _FS_RPATH >= 1 */



#if _FS_MINIMIZE <= 2
/*-----------------------------------------------------------------------*/
/* Seek File R/W Pointer                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_lseek (
    FIL* fp,        /* Pointer to the file object */
    DWORD ofs       /* File pointer from top of file */
)
{
    FRESULT res;
    DWORD clst, bcs, nsect, ifptr;
#if _USE_FASTSEEK
    DWORD cl, pcl, ncl, tcl, dsc, tlen, ulen, *tbl;
#endif


    res = validate(fp);                 /* Check validity of the object */
    if (res != FR_OK) LEAVE_FF(fp->fs, res);
    if (fp->err)                        /* Check error */
        LEAVE_FF(fp->fs, (FRESULT)fp->err);

#if _USE_FASTSEEK
    if (fp->cltbl) {    /* Fast seek */
        if (ofs == CREATE_LINKMAP) {    /* Create CLMT */
            tbl = fp->cltbl;
            tlen = *tbl++; ulen = 2;    /* Given table size and required table size */
            cl = fp->sclust;            /* Top of the chain */
            if (cl) {
                do {
                    /* Get a fragment */
                    tcl = cl; ncl = 0; ulen += 2;   /* Top, length and used items */
                    do {
                        pcl = cl; ncl++;
                        cl = get_fat(fp->fs, cl);
                        if (cl <= 1) ABORT(fp->fs, FR_INT_ERR);
                        if (cl == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                    } while (cl == pcl + 1);
                    if (ulen <= tlen) {     /* Store the length and top of the fragment */
                        *tbl++ = ncl; *tbl++ = tcl;
                    }
                } while (cl < fp->fs->n_fatent);    /* Repeat until end of chain */
            }
            *fp->cltbl = ulen;  /* Number of items used */
            if (ulen <= tlen)
                *tbl = 0;       /* Terminate table */
            else
                res = FR_NOT_ENOUGH_CORE;   /* Given table size is smaller than required */

        } else {                            /* Fast seek */
            if (ofs > fp->fsize)            /* Clip offset at the file size */
                ofs = fp->fsize;
            fp->fptr = ofs;                 /* Set file pointer */
            if (ofs) {
                fp->clust = clmt_clust(fp, ofs - 1);
                dsc = clust2sect(fp->fs, fp->clust);
                if (!dsc) ABORT(fp->fs, FR_INT_ERR);
                dsc += (ofs - 1) / SS(fp->fs) & (fp->fs->csize - 1);
                if (fp->fptr % SS(fp->fs) && dsc != fp->dsect) {    /* Refill sector cache
                if needed */
#if !_FS_TINY
#if !_FS_READONLY
```

```
                    if (fp->flag & FA__DIRTY) {      /* Write-back dirty sector cache */
                        if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                            ABORT(fp->fs, FR_DISK_ERR);
                        fp->flag &= ~FA__DIRTY;
                    }
#endif
                    if (disk_read(fp->fs->drv, fp->buf.d8, dsc, 1) != RES_OK)    /* Load
                    current sector */
                        ABORT(fp->fs, FR_DISK_ERR);
#endif
                    fp->dsect = dsc;
                }
            }
        }
    } else
#endif

    /* Normal Seek */
    {
        if (ofs > fp->fsize                 /* In read-only mode, clip offset with the file
        size */
#if !_FS_READONLY
            && !(fp->flag & FA_WRITE)
#endif
            ) ofs = fp->fsize;

        ifptr = fp->fptr;
        fp->fptr = nsect = 0;
        if (ofs) {
            bcs = (DWORD)fp->fs->csize * SS(fp->fs);    /* Cluster size (byte) */
            if (ifptr > 0 &&
                (ofs - 1) / bcs >= (ifptr - 1) / bcs) { /* When seek to same or following
                cluster, */
                fp->fptr = (ifptr - 1) & ~(bcs - 1);    /* start from the current cluster */
                ofs -= fp->fptr;
                clst = fp->clust;
            } else {                                    /* When seek to back cluster, */
                clst = fp->sclust;                      /* start from the first cluster */
#if !_FS_READONLY
                if (clst == 0) {                        /* If no cluster chain, create a new
                chain */
                    clst = create_chain(fp->fs, 0);
                    if (clst == 1) ABORT(fp->fs, FR_INT_ERR);
                    if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                    fp->sclust = clst;
                }
#endif
                fp->clust = clst;
            }
            if (clst != 0) {
                while (ofs > bcs) {                     /* Cluster following loop */
#if !_FS_READONLY
                    if (fp->flag & FA_WRITE) {          /* Check if in write mode or not */
                        clst = create_chain(fp->fs, clst);  /* Force stretch if in write
                        mode */
                        if (clst == 0) {                /* When disk gets full, clip file
                        size */
                            ofs = bcs; break;
                        }
                    } else
#endif
                        clst = get_fat(fp->fs, clst);   /* Follow cluster chain if not in
                        write mode */
                    if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                    if (clst <= 1 || clst >= fp->fs->n_fatent) ABORT(fp->fs, FR_INT_ERR);
                    fp->clust = clst;
                    fp->fptr += bcs;
                    ofs -= bcs;
                }
                fp->fptr += ofs;
                if (ofs % SS(fp->fs)) {
                    nsect = clust2sect(fp->fs, clst);   /* Current sector */
                    if (!nsect) ABORT(fp->fs, FR_INT_ERR);
```

```
                    nsect += ofs / SS(fp->fs);
                }
            }
        }
        if (fp->fptr % SS(fp->fs) && nsect != fp->dsect) {  /* Fill sector cache if needed */
#if !_FS_TINY
#if !_FS_READONLY
            if (fp->flag & FA__DIRTY) {         /* Write-back dirty sector cache */
                if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                    ABORT(fp->fs, FR_DISK_ERR);
                fp->flag &= ~FA__DIRTY;
            }
#endif
            if (disk_read(fp->fs->drv, fp->buf.d8, nsect, 1) != RES_OK) /* Fill sector cache
            */
                ABORT(fp->fs, FR_DISK_ERR);
#endif
            fp->dsect = nsect;
        }
#if !_FS_READONLY
        if (fp->fptr > fp->fsize) {         /* Set file change flag if the file size is
        extended */
            fp->fsize = fp->fptr;
            fp->flag |= FA__WRITTEN;
        }
#endif
    }

    LEAVE_FF(fp->fs, res);
}



#if _FS_MINIMIZE <= 1
/*-----------------------------------------------------------------------*/
/* Create a Directory Object                                             */
/*-----------------------------------------------------------------------*/

FRESULT f_opendir (
    DIR* dp,            /* Pointer to directory object to create */
    const TCHAR* path   /* Pointer to the directory path */
)
{
    FRESULT res;
    FATFS* fs;
    DEFINE_NAMEBUF;


    if (!dp) return FR_INVALID_OBJECT;

    /* Get logical drive number */
    res = find_volume(&fs, &path, 0);
    if (res == FR_OK) {
        dp->fs = fs;
        INIT_BUF(*dp);
        res = follow_path(dp, path);            /* Follow the path to the directory */
        FREE_BUF();
        if (res == FR_OK) {                     /* Follow completed */
            if (dp->dir) {                      /* It is not the origin directory itself */
                if (dp->dir[DIR_Attr] & AM_DIR) /* The object is a sub directory */
                    dp->sclust = ld_clust(fs, dp->dir);
                else                            /* The object is a file */
                    res = FR_NO_PATH;
            }
            if (res == FR_OK) {
                dp->id = fs->id;
                res = dir_sdi(dp, 0);           /* Rewind directory */
#if _FS_LOCK
                if (res == FR_OK) {
                    if (dp->sclust) {
                        dp->lockid = inc_lock(dp, 0);   /* Lock the sub directory */
                        if (!dp->lockid)
                            res = FR_TOO_MANY_OPEN_FILES;
```

```
                    } else {
                        dp->lockid = 0; /* Root directory need not to be locked */
                    }
                }
#endif
            }
        }
        if (res == FR_NO_FILE) res = FR_NO_PATH;
    }
    if (res != FR_OK) dp->fs = 0;       /* Invalidate the directory object if function faild
    */

    LEAVE_FF(fs, res);
}




/*-----------------------------------------------------------------------*/
/* Close Directory                                                       */
/*-----------------------------------------------------------------------*/

FRESULT f_closedir (
    DIR *dp     /* Pointer to the directory object to be closed */
)
{
    FRESULT res;


    res = validate(dp);
    if (res == FR_OK) {
#if _FS_REENTRANT
        FATFS *fs = dp->fs;
#endif
#if _FS_LOCK
        if (dp->lockid)             /* Decrement sub-directory open counter */
            res = dec_lock(dp->lockid);
        if (res == FR_OK)
#endif
            dp->fs = 0;             /* Invalidate directory object */
#if _FS_REENTRANT
        unlock_fs(fs, FR_OK);       /* Unlock volume */
#endif
    }
    return res;
}




/*-----------------------------------------------------------------------*/
/* Read Directory Entries in Sequence                                    */
/*-----------------------------------------------------------------------*/

FRESULT f_readdir (
    DIR* dp,            /* Pointer to the open directory object */
    FILINFO* fno        /* Pointer to file information to return */
)
{
    FRESULT res;
    DEFINE_NAMEBUF;


    res = validate(dp);                     /* Check validity of the object */
    if (res == FR_OK) {
        if (!fno) {
            res = dir_sdi(dp, 0);           /* Rewind the directory object */
        } else {
            INIT_BUF(*dp);
            res = dir_read(dp, 0);          /* Read an item */
            if (res == FR_NO_FILE) {        /* Reached end of directory */
                dp->sect = 0;
                res = FR_OK;
```

```
            }
            if (res == FR_OK) {             /* A valid entry is found */
                get_fileinfo(dp, fno);      /* Get the object information */
                res = dir_next(dp, 0);      /* Increment index for next */
                if (res == FR_NO_FILE) {
                    dp->sect = 0;
                    res = FR_OK;
                }
            }
            FREE_BUF();
        }
    }

    LEAVE_FF(dp->fs, res);
}



#if _USE_FIND
/*-----------------------------------------------------------------------*/
/* Find next file                                                        */
/*-----------------------------------------------------------------------*/

FRESULT f_findnext (
    DIR* dp,        /* Pointer to the open directory object */
    FILINFO* fno    /* Pointer to the file information structure */
)
{
    FRESULT res;


    for (;;) {
        res = f_readdir(dp, fno);       /* Get a directory item */
        if (res != FR_OK || !fno || !fno->fname[0]) break;  /* Terminate if any error or end
        of directory */
#if _USE_LFN
        if (fno->lfname && pattern_matching(dp->pat, fno->lfname, 0, 0)) break; /* Test for
        LFN if exist */
#endif
        if (pattern_matching(dp->pat, fno->fname, 0, 0)) break; /* Test for SFN */
    }
    return res;

}



/*-----------------------------------------------------------------------*/
/* Find first file                                                       */
/*-----------------------------------------------------------------------*/

FRESULT f_findfirst (
    DIR* dp,                /* Pointer to the blank directory object */
    FILINFO* fno,           /* Pointer to the file information structure */
    const TCHAR* path,      /* Pointer to the directory to open */
    const TCHAR* pattern    /* Pointer to the matching pattern */
)
{
    FRESULT res;


    dp->pat = pattern;      /* Save pointer to pattern string */
    res = f_opendir(dp, path);      /* Open the target directory */
    if (res == FR_OK)
        res = f_findnext(dp, fno);  /* Find the first item */
    return res;
}

#endif  /* _USE_FIND */



#if _FS_MINIMIZE == 0
```

```
/*-----------------------------------------------------------------------*/
/* Get File Status                                                       */
/*-----------------------------------------------------------------------*/

FRESULT f_stat (
    const TCHAR* path,  /* Pointer to the file path */
    FILINFO* fno        /* Pointer to file information to return */
)
{
    FRESULT res;
    DIR dj;
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 0);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);   /* Follow the file path */
        if (res == FR_OK) {             /* Follow completed */
            if (dj.dir) {       /* Found an object */
                if (fno) get_fileinfo(&dj, fno);
            } else {            /* It is root directory */
                res = FR_INVALID_NAME;
            }
        }
        FREE_BUF();
    }

    LEAVE_FF(dj.fs, res);
}



#if !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Get Number of Free Clusters                                           */
/*-----------------------------------------------------------------------*/

FRESULT f_getfree (
    const TCHAR* path,  /* Path name of the logical drive number */
    DWORD* nclst,       /* Pointer to a variable to return number of free clusters */
    FATFS** fatfs       /* Pointer to return pointer to corresponding file system object */
)
{
    FRESULT res;
    FATFS *fs;
    DWORD n, clst, sect, stat;
    UINT i;
    BYTE fat, *p;


    /* Get logical drive number */
    res = find_volume(fatfs, &path, 0);
    fs = *fatfs;
    if (res == FR_OK) {
        /* If free_clust is valid, return it without full cluster scan */
        if (fs->free_clust <= fs->n_fatent - 2) {
            *nclst = fs->free_clust;
        } else {
            /* Get number of free clusters */
            fat = fs->fs_type;
            n = 0;
            if (fat == FS_FAT12) {
                clst = 2;
                do {
                    stat = get_fat(fs, clst);
                    if (stat == 0xFFFFFFFF) { res = FR_DISK_ERR; break; }
                    if (stat == 1) { res = FR_INT_ERR; break; }
                    if (stat == 0) n++;
                } while (++clst < fs->n_fatent);
            } else {
                clst = fs->n_fatent;
```

```
                sect = fs->fatbase;
                i = 0; p = 0;
                do {
                    if (!i) {
                        res = move_window(fs, sect++);
                        if (res != FR_OK) break;
                        p = fs->win.d8;
                        i = SS(fs);
                    }
                    if (fat == FS_FAT16) {
                        if (LD_WORD(p) == 0) n++;
                        p += 2; i -= 2;
                    } else {
                        if ((LD_DWORD(p) & 0x0FFFFFFF) == 0) n++;
                        p += 4; i -= 4;
                    }
                } while (--clst);
            }
            fs->free_clust = n;
            fs->fsi_flag |= 1;
            *nclst = n;
        }
    }
    LEAVE_FF(fs, res);
}



/*-----------------------------------------------------------------------*/
/* Truncate File                                                         */
/*-----------------------------------------------------------------------*/

FRESULT f_truncate (
    FIL* fp     /* Pointer to the file object */
)
{
    FRESULT res;
    DWORD ncl;


    res = validate(fp);                     /* Check validity of the object */
    if (res == FR_OK) {
        if (fp->err) {                      /* Check error */
            res = (FRESULT)fp->err;
        } else {
            if (!(fp->flag & FA_WRITE))     /* Check access mode */
                res = FR_DENIED;
        }
    }
    if (res == FR_OK) {
        if (fp->fsize > fp->fptr) {
            fp->fsize = fp->fptr;   /* Set file size to current R/W point */
            fp->flag |= FA__WRITTEN;
            if (fp->fptr == 0) {    /* When set file size to zero, remove entire cluster
            chain */
                res = remove_chain(fp->fs, fp->sclust);
                fp->sclust = 0;
            } else {                /* When truncate a part of the file, remove remaining
            clusters */
                ncl = get_fat(fp->fs, fp->clust);
                res = FR_OK;
                if (ncl == 0xFFFFFFFF) res = FR_DISK_ERR;
                if (ncl == 1) res = FR_INT_ERR;
                if (res == FR_OK && ncl < fp->fs->n_fatent) {
                    res = put_fat(fp->fs, fp->clust, 0x0FFFFFFF);
                    if (res == FR_OK) res = remove_chain(fp->fs, ncl);
                }
            }
#if !_FS_TINY
            if (res == FR_OK && (fp->flag & FA__DIRTY)) {
                if (disk_write(fp->fs->drv, fp->buf.d8, fp->dsect, 1) != RES_OK)
                    res = FR_DISK_ERR;
```

```
                else
                    fp->flag &= ~FA__DIRTY;
            }
#endif
        }
        if (res != FR_OK) fp->err = (FRESULT)res;
    }

    LEAVE_FF(fp->fs, res);
}




/*-----------------------------------------------------------------------*/
/* Delete a File or Directory                                            */
/*-----------------------------------------------------------------------*/

FRESULT f_unlink (
    const TCHAR* path       /* Pointer to the file or directory path */
)
{
    FRESULT res;
    DIR dj, sdj;
    BYTE *dir;
    DWORD dclst = 0;
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 1);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);       /* Follow the file path */
        if (_FS_RPATH && res == FR_OK && (dj.fn[NSFLAG] & NS_DOT))
            res = FR_INVALID_NAME;          /* Cannot remove dot entry */
#if _FS_LOCK
        if (res == FR_OK) res = chk_lock(&dj, 2);   /* Cannot remove open object */
#endif
        if (res == FR_OK) {                 /* The object is accessible */
            dir = dj.dir;
            if (!dir) {
                res = FR_INVALID_NAME;      /* Cannot remove the origin directory */
            } else {
                if (dir[DIR_Attr] & AM_RDO)
                    res = FR_DENIED;        /* Cannot remove R/O object */
            }
            if (res == FR_OK) {
                dclst = ld_clust(dj.fs, dir);
                if (dclst && (dir[DIR_Attr] & AM_DIR)) {    /* Is it a sub-directory ? */
#if _FS_RPATH
                    if (dclst == dj.fs->cdir) {             /* Is it the current directory? */
                        res = FR_DENIED;
                    } else
#endif
                    {
                        mem_cpy(&sdj, &dj, sizeof (DIR));   /* Open the sub-directory */
                        sdj.sclust = dclst;
                        res = dir_sdi(&sdj, 2);
                        if (res == FR_OK) {
                            res = dir_read(&sdj, 0);            /* Read an item (excluding
                            dot entries) */
                            if (res == FR_OK) res = FR_DENIED;  /* Not empty? (cannot
                            remove) */
                            if (res == FR_NO_FILE) res = FR_OK; /* Empty? (can remove) */
                        }
                    }
                }
            }
            if (res == FR_OK) {
                res = dir_remove(&dj);      /* Remove the directory entry */
                if (res == FR_OK && dclst)  /* Remove the cluster chain if exist */
                    res = remove_chain(dj.fs, dclst);
```

```
                if (res == FR_OK) res = sync_fs(dj.fs);
            }
        }
        FREE_BUF();
    }

    LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Create a Directory                                                    */
/*-----------------------------------------------------------------------*/

FRESULT f_mkdir (
    const TCHAR* path       /* Pointer to the directory path */
)
{
    FRESULT res;
    DIR dj;
    BYTE *dir, n;
    DWORD dsc, dcl, pcl, tm = GET_FATTIME();
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 1);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);           /* Follow the file path */
        if (res == FR_OK) res = FR_EXIST;       /* Any object with same name is already
        existing */
        if (_FS_RPATH && res == FR_NO_FILE && (dj.fn[NSFLAG] & NS_DOT))
            res = FR_INVALID_NAME;
        if (res == FR_NO_FILE) {                /* Can create a new directory */
            dcl = create_chain(dj.fs, 0);       /* Allocate a cluster for the new directory
            table */
            res = FR_OK;
            if (dcl == 0) res = FR_DENIED;      /* No space to allocate a new cluster */
            if (dcl == 1) res = FR_INT_ERR;
            if (dcl == 0xFFFFFFFF) res = FR_DISK_ERR;
            if (res == FR_OK)                   /* Flush FAT */
                res = sync_window(dj.fs);
            if (res == FR_OK) {                 /* Initialize the new directory table */
                dsc = clust2sect(dj.fs, dcl);
                dir = dj.fs->win.d8;
                mem_set(dir, 0, SS(dj.fs));
                mem_set(dir + DIR_Name, ' ', 11);   /* Create "." entry */
                dir[DIR_Name] = '.';
                dir[DIR_Attr] = AM_DIR;
                ST_DWORD(dir + DIR_WrtTime, tm);
                st_clust(dir, dcl);
                mem_cpy(dir + SZ_DIRE, dir, SZ_DIRE);   /* Create ".." entry */
                dir[SZ_DIRE + 1] = '.'; pcl = dj.sclust;
                if (dj.fs->fs_type == FS_FAT32 && pcl == dj.fs->dirbase)
                    pcl = 0;
                st_clust(dir + SZ_DIRE, pcl);
                for (n = dj.fs->csize; n; n--) {    /* Write dot entries and clear following
                sectors */
                    dj.fs->winsect = dsc++;
                    dj.fs->wflag = 1;
                    res = sync_window(dj.fs);
                    if (res != FR_OK) break;
                    mem_set(dir, 0, SS(dj.fs));
                }
            }
            if (res == FR_OK) res = dir_register(&dj);  /* Register the object to the
            directoy */
            if (res != FR_OK) {
                remove_chain(dj.fs, dcl);           /* Could not register, remove cluster
                chain */
```

```
            } else {
                dir = dj.dir;
                dir[DIR_Attr] = AM_DIR;             /* Attribute */
                ST_DWORD(dir + DIR_WrtTime, tm);    /* Created time */
                st_clust(dir, dcl);                 /* Table start cluster */
                dj.fs->wflag = 1;
                res = sync_fs(dj.fs);
            }
        }
        FREE_BUF();
    }

    LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Change Attribute                                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_chmod (
    const TCHAR* path,  /* Pointer to the file path */
    BYTE attr,          /* Attribute bits */
    BYTE mask           /* Attribute mask to change */
)
{
    FRESULT res;
    DIR dj;
    BYTE *dir;
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 1);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);       /* Follow the file path */
        FREE_BUF();
        if (_FS_RPATH && res == FR_OK && (dj.fn[NSFLAG] & NS_DOT))
            res = FR_INVALID_NAME;
        if (res == FR_OK) {
            dir = dj.dir;
            if (!dir) {                     /* Is it a root directory? */
                res = FR_INVALID_NAME;
            } else {                        /* File or sub directory */
                mask &= AM_RDO|AM_HID|AM_SYS|AM_ARC;    /* Valid attribute mask */
                dir[DIR_Attr] = (attr & mask) | (dir[DIR_Attr] & (BYTE)~mask);  /* Apply
                attribute change */
                dj.fs->wflag = 1;
                res = sync_fs(dj.fs);
            }
        }
    }

    LEAVE_FF(dj.fs, res);
}




/*-----------------------------------------------------------------------*/
/* Rename File/Directory                                                 */
/*-----------------------------------------------------------------------*/

FRESULT f_rename (
    const TCHAR* path_old,  /* Pointer to the object to be renamed */
    const TCHAR* path_new   /* Pointer to the new name */
)
{
    FRESULT res;
    DIR djo, djn;
```

```
    BYTE buf[21], *dir;
    DWORD dw;
    DEFINE_NAMEBUF;


    /* Get logical drive number of the source object */
    res = find_volume(&djo.fs, &path_old, 1);
    if (res == FR_OK) {
        djn.fs = djo.fs;
        INIT_BUF(djo);
        res = follow_path(&djo, path_old);      /* Check old object */
        if (_FS_RPATH && res == FR_OK && (djo.fn[NSFLAG] & NS_DOT))
            res = FR_INVALID_NAME;
#if _FS_LOCK
        if (res == FR_OK) res = chk_lock(&djo, 2);
#endif
        if (res == FR_OK) {                     /* Old object is found */
            if (!djo.dir) {                     /* Is root dir? */
                res = FR_NO_FILE;
            } else {
                mem_cpy(buf, djo.dir + DIR_Attr, 21);   /* Save information about object
                except name */
                mem_cpy(&djn, &djo, sizeof (DIR));      /* Duplicate the directory object */
                if (get_ldnumber(&path_new) >= 0)       /* Snip drive number off and ignore
                it */
                    res = follow_path(&djn, path_new);  /* and make sure if new object name
                    is not conflicting */
                else
                    res = FR_INVALID_DRIVE;
                if (res == FR_OK) res = FR_EXIST;       /* The new object name is already
                existing */
                if (res == FR_NO_FILE) {                /* It is a valid path and no name
                collision */
                    res = dir_register(&djn);           /* Register the new entry */
                    if (res == FR_OK) {
/* Start of critical section where any interruption can cause a cross-link */
                        dir = djn.dir;                  /* Copy information about object
                        except name */
                        mem_cpy(dir + 13, buf + 2, 19);
                        dir[DIR_Attr] = buf[0] | AM_ARC;
                        djo.fs->wflag = 1;
                        if ((dir[DIR_Attr] & AM_DIR) && djo.sclust != djn.sclust) { /*
                        Update .. entry in the sub-directory if needed */
                            dw = clust2sect(djo.fs, ld_clust(djo.fs, dir));
                            if (!dw) {
                                res = FR_INT_ERR;
                            } else {
                                res = move_window(djo.fs, dw);
                                dir = djo.fs->win.d8 + SZ_DIRE * 1; /* Ptr to .. entry */
                                if (res == FR_OK && dir[1] == '.') {
                                    st_clust(dir, djn.sclust);
                                    djo.fs->wflag = 1;
                                }
                            }
                        }
                        if (res == FR_OK) {
                            res = dir_remove(&djo);     /* Remove old entry */
                            if (res == FR_OK)
                                res = sync_fs(djo.fs);
                        }
/* End of critical section */
                    }
                }
            }
        }
        FREE_BUF();
    }

    LEAVE_FF(djo.fs, res);
}
```

```
/*-----------------------------------------------------------------------*/
/* Change Timestamp                                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_utime (
    const TCHAR* path,  /* Pointer to the file/directory name */
    const FILINFO* fno  /* Pointer to the time stamp to be set */
)
{
    FRESULT res;
    DIR dj;
    BYTE *dir;
    DEFINE_NAMEBUF;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 1);
    if (res == FR_OK) {
        INIT_BUF(dj);
        res = follow_path(&dj, path);   /* Follow the file path */
        FREE_BUF();
        if (_FS_RPATH && res == FR_OK && (dj.fn[NSFLAG] & NS_DOT))
            res = FR_INVALID_NAME;
        if (res == FR_OK) {
            dir = dj.dir;
            if (!dir) {                 /* Root directory */
                res = FR_INVALID_NAME;
            } else {                    /* File or sub-directory */
                ST_WORD(dir + DIR_WrtTime, fno->ftime);
                ST_WORD(dir + DIR_WrtDate, fno->fdate);
                dj.fs->wflag = 1;
                res = sync_fs(dj.fs);
            }
        }
    }

    LEAVE_FF(dj.fs, res);
}

#endif /* !_FS_READONLY */
#endif /* _FS_MINIMIZE == 0 */
#endif /* _FS_MINIMIZE <= 1 */
#endif /* _FS_MINIMIZE <= 2 */




#if _USE_LABEL
/*-----------------------------------------------------------------------*/
/* Get volume label                                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_getlabel (
    const TCHAR* path,  /* Path name of the logical drive number */
    TCHAR* label,       /* Pointer to a buffer to return the volume label */
    DWORD* vsn          /* Pointer to a variable to return the volume serial number */
)
{
    FRESULT res;
    DIR dj;
    UINT i, j;
#if _USE_LFN && _LFN_UNICODE
    WCHAR w;
#endif


    /* Get logical drive number */
    res = find_volume(&dj.fs, &path, 0);

    /* Get volume label */
    if (res == FR_OK && label) {
        dj.sclust = 0;                  /* Open root directory */
```

```
        res = dir_sdi(&dj, 0);
        if (res == FR_OK) {
            res = dir_read(&dj, 1);     /* Get an entry with AM_VOL */
            if (res == FR_OK) {         /* A volume label is exist */
#if _USE_LFN && _LFN_UNICODE
                i = j = 0;
                do {
                    w = (i < 11) ? dj.dir[i++] : ' ';
                    if (IsDBCS1(w) && i < 11 && IsDBCS2(dj.dir[i]))
                        w = w << 8 | dj.dir[i++];
                    label[j++] = ff_convert(w, 1);  /* OEM -> Unicode */
                } while (j < 11);
#else
                mem_cpy(label, dj.dir, 11);
#endif
                j = 11;
                do {
                    label[j] = 0;
                    if (!j) break;
                } while (label[--j] == ' ');
            }
            if (res == FR_NO_FILE) {    /* No label, return nul string */
                label[0] = 0;
                res = FR_OK;
            }
        }
    }

    /* Get volume serial number */
    if (res == FR_OK && vsn) {
        res = move_window(dj.fs, dj.fs->volbase);
        if (res == FR_OK) {
            i = dj.fs->fs_type == FS_FAT32 ? BS_VolID32 : BS_VolID;
            *vsn = LD_DWORD(&dj.fs->win.d8[i]);
        }
    }

    LEAVE_FF(dj.fs, res);
}



#if !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Set volume label                                                      */
/*-----------------------------------------------------------------------*/

FRESULT f_setlabel (
    const TCHAR* label  /* Pointer to the volume label to set */
)
{
    FRESULT res;
    DIR dj;
    BYTE vn[11];
    UINT i, j, sl;
    WCHAR w;
    DWORD tm;


    /* Get logical drive number */
    res = find_volume(&dj.fs, &label, 1);
    if (res) LEAVE_FF(dj.fs, res);

    /* Create a volume label in directory form */
    vn[0] = 0;
    for (sl = 0; label[sl]; sl++) ;             /* Get name length */
    for ( ; sl && label[sl - 1] == ' '; sl--) ; /* Remove trailing spaces */
    if (sl) {   /* Create volume label in directory form */
        i = j = 0;
        do {
#if _USE_LFN && _LFN_UNICODE
            w = ff_convert(ff_wtoupper(label[i++]), 0);
#else
```

```
            w = (BYTE)label[i++];
            if (IsDBCS1(w))
                w = (j < 10 && i < sl && IsDBCS2(label[i])) ? w << 8 | (BYTE)label[i++] : 0;
#if _USE_LFN
            w = ff_convert(ff_wtoupper(ff_convert(w, 1)), 0);
#else
            if (IsLower(w)) w -= 0x20;          /* To upper ASCII characters */
#ifdef _EXCVT
            if (w >= 0x80) w = ExCvt[w - 0x80]; /* To upper extended characters (SBCS cfg) */
#else
            if (!_DF1S && w >= 0x80) w = 0;     /* Reject extended characters (ASCII cfg) */
#endif
#endif
#endif
            if (!w || chk_chr("\"*+,.:;<=>\?[]|\x7F", w) || j >= (UINT)((w >= 0x100) ? 10 :
            11)) /* Reject invalid characters for volume label */
                LEAVE_FF(dj.fs, FR_INVALID_NAME);
            if (w >= 0x100) vn[j++] = (BYTE)(w >> 8);
            vn[j++] = (BYTE)w;
        } while (i < sl);
        while (j < 11) vn[j++] = ' ';   /* Fill remaining name field */
        if (vn[0] == DDEM) LEAVE_FF(dj.fs, FR_INVALID_NAME);    /* Reject illegal name
        (heading DDEM) */
    }

    /* Set volume label */
    dj.sclust = 0;                  /* Open root directory */
    res = dir_sdi(&dj, 0);
    if (res == FR_OK) {
        res = dir_read(&dj, 1);     /* Get an entry with AM_VOL */
        if (res == FR_OK) {         /* A volume label is found */
            if (vn[0]) {
                mem_cpy(dj.dir, vn, 11);    /* Change the volume label name */
                tm = GET_FATTIME();
                ST_DWORD(dj.dir + DIR_WrtTime, tm);
            } else {
                dj.dir[0] = DDEM;           /* Remove the volume label */
            }
            dj.fs->wflag = 1;
            res = sync_fs(dj.fs);
        } else {                    /* No volume label is found or error */
            if (res == FR_NO_FILE) {
                res = FR_OK;
                if (vn[0]) {                /* Create volume label as new */
                    res = dir_alloc(&dj, 1);    /* Allocate an entry for volume label */
                    if (res == FR_OK) {
                        mem_set(dj.dir, 0, SZ_DIRE);    /* Set volume label */
                        mem_cpy(dj.dir, vn, 11);
                        dj.dir[DIR_Attr] = AM_VOL;
                        tm = GET_FATTIME();
                        ST_DWORD(dj.dir + DIR_WrtTime, tm);
                        dj.fs->wflag = 1;
                        res = sync_fs(dj.fs);
                    }
                }
            }
        }
    }

    LEAVE_FF(dj.fs, res);
}

#endif /* !_FS_READONLY */
#endif /* _USE_LABEL */



/*-----------------------------------------------------------------------*/
/* Forward data to the stream directly (available on only tiny cfg)      */
/*-----------------------------------------------------------------------*/
#if _USE_FORWARD && _FS_TINY

FRESULT f_forward (
```

```
    FIL* fp,                        /* Pointer to the file object */
    UINT (*func)(const BYTE*,UINT), /* Pointer to the streaming function */
    UINT btf,                       /* Number of bytes to forward */
    UINT* bf                        /* Pointer to number of bytes forwarded */
)
{
    FRESULT res;
    DWORD remain, clst, sect;
    UINT rcnt;
    BYTE csect;


    *bf = 0;    /* Clear transfer byte counter */

    res = validate(fp);                             /* Check validity of the object */
    if (res != FR_OK) LEAVE_FF(fp->fs, res);
    if (fp->err)                                    /* Check error */
        LEAVE_FF(fp->fs, (FRESULT)fp->err);
    if (!(fp->flag & FA_READ))                      /* Check access mode */
        LEAVE_FF(fp->fs, FR_DENIED);

    remain = fp->fsize - fp->fptr;
    if (btf > remain) btf = (UINT)remain;           /* Truncate btf by remaining bytes */

    for ( ;  btf && (*func)(0, 0);                  /* Repeat until all data transferred or
    stream becomes busy */
        fp->fptr += rcnt, *bf += rcnt, btf -= rcnt) {
        csect = (BYTE)(fp->fptr / SS(fp->fs) & (fp->fs->csize - 1));    /* Sector offset in
        the cluster */
        if ((fp->fptr % SS(fp->fs)) == 0) {         /* On the sector boundary? */
            if (!csect) {                           /* On the cluster boundary? */
                clst = (fp->fptr == 0) ?            /* On the top of the file? */
                    fp->sclust : get_fat(fp->fs, fp->clust);
                if (clst <= 1) ABORT(fp->fs, FR_INT_ERR);
                if (clst == 0xFFFFFFFF) ABORT(fp->fs, FR_DISK_ERR);
                fp->clust = clst;                   /* Update current cluster */
            }
        }
        sect = clust2sect(fp->fs, fp->clust);       /* Get current data sector */
        if (!sect) ABORT(fp->fs, FR_INT_ERR);
        sect += csect;
        if (move_window(fp->fs, sect) != FR_OK)     /* Move sector window */
            ABORT(fp->fs, FR_DISK_ERR);
        fp->dsect = sect;
        rcnt = SS(fp->fs) - (WORD)(fp->fptr % SS(fp->fs));  /* Forward data from sector
        window */
        if (rcnt > btf) rcnt = btf;
        rcnt = (*func)(&fp->fs->win.d8[(WORD)fp->fptr % SS(fp->fs)], rcnt);
        if (!rcnt) ABORT(fp->fs, FR_INT_ERR);
    }

    LEAVE_FF(fp->fs, FR_OK);
}
#endif /* _USE_FORWARD */



#if _USE_MKFS && !_FS_READONLY
/*-----------------------------------------------------------------------*/
/* Create file system on the logical drive                               */
/*-----------------------------------------------------------------------*/
#define N_ROOTDIR   512     /* Number of root directory entries for FAT12/16 */
#define N_FATS      1       /* Number of FATs (1 or 2) */


FRESULT f_mkfs (
    const TCHAR* path,  /* Logical drive number */
    BYTE sfd,           /* Partitioning rule 0:FDISK, 1:SFD */
    UINT au             /* Size of allocation unit in unit of byte or sector */
)
{
    static const WORD vst[] = { 1024,   512,  256,  128,   64,    32,   16,    8,    4,
    2,   0};
```

```
    static const WORD cst[] = {32768, 16384, 8192, 4096, 2048, 16384, 8192, 4096, 2048,
    1024, 512};
    int vol;
    BYTE fmt, md, sys, *tbl, pdrv, part;
    DWORD n_clst, vs, n, wsect;
    UINT i;
    DWORD b_vol, b_fat, b_dir, b_data;  /* LBA */
    DWORD n_vol, n_rsv, n_fat, n_dir;   /* Size */
    FATFS *fs;
    DSTATUS stat;
#if _USE_TRIM
    DWORD eb[2];
#endif


    /* Check mounted drive and clear work area */
    if (sfd > 1) return FR_INVALID_PARAMETER;
    vol = get_ldnumber(&path);
    if (vol < 0) return FR_INVALID_DRIVE;
    fs = FatFs[vol];
    if (!fs) return FR_NOT_ENABLED;
    fs->fs_type = 0;
    pdrv = LD2PD(vol);  /* Physical drive */
    part = LD2PT(vol);  /* Partition (0:auto detect, 1-4:get from partition table)*/

    /* Get disk statics */
    stat = disk_initialize(pdrv);
    if (stat & STA_NOINIT) return FR_NOT_READY;
    if (stat & STA_PROTECT) return FR_WRITE_PROTECTED;
#if _MAX_SS != _MIN_SS      /* Get disk sector size */
    if (disk_ioctl(pdrv, GET_SECTOR_SIZE, &SS(fs)) != RES_OK || SS(fs) > _MAX_SS || SS(fs) <
    _MIN_SS)
        return FR_DISK_ERR;
#endif
    if (_MULTI_PARTITION && part) {
        /* Get partition information from partition table in the MBR */
        if (disk_read(pdrv, fs->win.d8, 0, 1) != RES_OK) return FR_DISK_ERR;
        if (LD_WORD(fs->win.d8 + BS_55AA) != 0xAA55) return FR_MKFS_ABORTED;
        tbl = &fs->win.d8[MBR_Table + (part - 1) * SZ_PTE];
        if (!tbl[4]) return FR_MKFS_ABORTED;    /* No partition? */
        b_vol = LD_DWORD(tbl + 8);  /* Volume start sector */
        n_vol = LD_DWORD(tbl + 12); /* Volume size */
    } else {
        /* Create a partition in this function */
        if (disk_ioctl(pdrv, GET_SECTOR_COUNT, &n_vol) != RES_OK || n_vol < 128)
            return FR_DISK_ERR;
        b_vol = (sfd) ? 0 : 63;     /* Volume start sector */
        n_vol -= b_vol;             /* Volume size */
    }

    if (au & (au - 1)) au = 0;
    if (!au) {                      /* AU auto selection */
        vs = n_vol / (2000 / (SS(fs) / 512));
        for (i = 0; vs < vst[i]; i++) ;
        au = cst[i];
    }
    if (au >= _MIN_SS) au /= SS(fs);    /* Number of sectors per cluster */
    if (!au) au = 1;
    if (au > 128) au = 128;

    /* Pre-compute number of clusters and FAT sub-type */
    n_clst = n_vol / au;
    fmt = FS_FAT12;
    if (n_clst >= MIN_FAT16) fmt = FS_FAT16;
    if (n_clst >= MIN_FAT32) fmt = FS_FAT32;

    /* Determine offset and size of FAT structure */
    if (fmt == FS_FAT32) {
        n_fat = ((n_clst * 4) + 8 + SS(fs) - 1) / SS(fs);
        n_rsv = 32;
        n_dir = 0;
    } else {
        n_fat = (fmt == FS_FAT12) ? (n_clst * 3 + 1) / 2 + 3 : (n_clst * 2) + 4;
```

```
    n_fat = (n_fat + SS(fs) - 1) / SS(fs);
    n_rsv = 1;
    n_dir = (DWORD)N_ROOTDIR * SZ_DIRE / SS(fs);
}
b_fat = b_vol + n_rsv;              /* FAT area start sector */
b_dir = b_fat + n_fat * N_FATS;     /* Directory area start sector */
b_data = b_dir + n_dir;             /* Data area start sector */
if (n_vol < b_data + au - b_vol) return FR_MKFS_ABORTED;    /* Too small volume */

/* Align data start sector to erase block boundary (for flash memory media) */
if (disk_ioctl(pdrv, GET_BLOCK_SIZE, &n) != RES_OK || !n || n > 32768) n = 1;
n = (b_data + n - 1) & ~(n - 1);    /* Next nearest erase block from current data start */
n = (n - b_data) / N_FATS;
if (fmt == FS_FAT32) {      /* FAT32: Move FAT offset */
    n_rsv += n;
    b_fat += n;
} else {                    /* FAT12/16: Expand FAT size */
    n_fat += n;
}

/* Determine number of clusters and final check of validity of the FAT sub-type */
n_clst = (n_vol - n_rsv - n_fat * N_FATS - n_dir) / au;
if (   (fmt == FS_FAT16 && n_clst < MIN_FAT16)
    || (fmt == FS_FAT32 && n_clst < MIN_FAT32))
    return FR_MKFS_ABORTED;

/* Determine system ID in the partition table */
if (fmt == FS_FAT32) {
    sys = 0x0C;     /* FAT32X */
} else {
    if (fmt == FS_FAT12 && n_vol < 0x10000) {
        sys = 0x01; /* FAT12(<65536) */
    } else {
        sys = (n_vol < 0x10000) ? 0x04 : 0x06;  /* FAT16(<65536) : FAT12/16(>=65536) */
    }
}

if (_MULTI_PARTITION && part) {
    /* Update system ID in the partition table */
    tbl = &fs->win.d8[MBR_Table + (part - 1) * SZ_PTE];
    tbl[4] = sys;
    if (disk_write(pdrv, fs->win.d8, 0, 1) != RES_OK)   /* Write it to teh MBR */
        return FR_DISK_ERR;
    md = 0xF8;
} else {
    if (sfd) {  /* No partition table (SFD) */
        md = 0xF0;
    } else {    /* Create partition table (FDISK) */
        mem_set(fs->win.d8, 0, SS(fs));
        tbl = fs->win.d8 + MBR_Table;   /* Create partition table for single partition
        in the drive */
        tbl[1] = 1;                     /* Partition start head */
        tbl[2] = 1;                     /* Partition start sector */
        tbl[3] = 0;                     /* Partition start cylinder */
        tbl[4] = sys;                   /* System type */
        tbl[5] = 254;                   /* Partition end head */
        n = (b_vol + n_vol) / 63 / 255;
        tbl[6] = (BYTE)(n >> 2 | 63);   /* Partition end sector */
        tbl[7] = (BYTE)n;               /* End cylinder */
        ST_DWORD(tbl + 8, 63);          /* Partition start in LBA */
        ST_DWORD(tbl + 12, n_vol);      /* Partition size in LBA */
        ST_WORD(fs->win.d8 + BS_55AA, 0xAA55);  /* MBR signature */
        if (disk_write(pdrv, fs->win.d8, 0, 1) != RES_OK)   /* Write it to the MBR */
            return FR_DISK_ERR;
        md = 0xF8;
    }
}

/* Create BPB in the VBR */
tbl = fs->win.d8;                           /* Clear sector */
mem_set(tbl, 0, SS(fs));
mem_cpy(tbl, "\xEB\xFE\x90" "MSDOS5.0", 11);/* Boot jump code, OEM name */
i = SS(fs);                                 /* Sector size */
```

```
    ST_WORD(tbl + BPB_BytsPerSec, i);
    tbl[BPB_SecPerClus] = (BYTE)au;          /* Sectors per cluster */
    ST_WORD(tbl + BPB_RsvdSecCnt, n_rsv);    /* Reserved sectors */
    tbl[BPB_NumFATs] = N_FATS;               /* Number of FATs */
    i = (fmt == FS_FAT32) ? 0 : N_ROOTDIR;   /* Number of root directory entries */
    ST_WORD(tbl + BPB_RootEntCnt, i);
    if (n_vol < 0x10000) {                   /* Number of total sectors */
        ST_WORD(tbl + BPB_TotSec16, n_vol);
    } else {
        ST_DWORD(tbl + BPB_TotSec32, n_vol);
    }
    tbl[BPB_Media] = md;                     /* Media descriptor */
    ST_WORD(tbl + BPB_SecPerTrk, 63);        /* Number of sectors per track */
    ST_WORD(tbl + BPB_NumHeads, 255);        /* Number of heads */
    ST_DWORD(tbl + BPB_HiddSec, b_vol);      /* Hidden sectors */
    n = GET_FATTIME();                       /* Use current time as VSN */
    if (fmt == FS_FAT32) {
        ST_DWORD(tbl + BS_VolID32, n);       /* VSN */
        ST_DWORD(tbl + BPB_FATSz32, n_fat); /* Number of sectors per FAT */
        ST_DWORD(tbl + BPB_RootClus, 2);     /* Root directory start cluster (2) */
        ST_WORD(tbl + BPB_FSInfo, 1);        /* FSINFO record offset (VBR + 1) */
        ST_WORD(tbl + BPB_BkBootSec, 6);     /* Backup boot record offset (VBR + 6) */
        tbl[BS_DrvNum32] = 0x80;             /* Drive number */
        tbl[BS_BootSig32] = 0x29;            /* Extended boot signature */
        mem_cpy(tbl + BS_VolLab32, "NO NAME    " "FAT32   ", 19);   /* Volume label, FAT
        signature */
    } else {
        ST_DWORD(tbl + BS_VolID, n);         /* VSN */
        ST_WORD(tbl + BPB_FATSz16, n_fat);  /* Number of sectors per FAT */
        tbl[BS_DrvNum] = 0x80;               /* Drive number */
        tbl[BS_BootSig] = 0x29;              /* Extended boot signature */
        mem_cpy(tbl + BS_VolLab, "NO NAME    " "FAT     ", 19); /* Volume label, FAT
        signature */
    }
    ST_WORD(tbl + BS_55AA, 0xAA55);          /* Signature (Offset is fixed here regardless of
    sector size) */
    if (disk_write(pdrv, tbl, b_vol, 1) != RES_OK)  /* Write it to the VBR sector */
        return FR_DISK_ERR;
    if (fmt == FS_FAT32)                     /* Write backup VBR if needed (VBR + 6) */
        disk_write(pdrv, tbl, b_vol + 6, 1);

    /* Initialize FAT area */
    wsect = b_fat;
    for (i = 0; i < N_FATS; i++) {       /* Initialize each FAT copy */
        mem_set(tbl, 0, SS(fs));             /* 1st sector of the FAT  */
        n = md;                              /* Media descriptor byte */
        if (fmt != FS_FAT32) {
            n |= (fmt == FS_FAT12) ? 0x00FFFF00 : 0xFFFFFF00;
            ST_DWORD(tbl + 0, n);            /* Reserve cluster #0-1 (FAT12/16) */
        } else {
            n |= 0xFFFFFF00;
            ST_DWORD(tbl + 0, n);            /* Reserve cluster #0-1 (FAT32) */
            ST_DWORD(tbl + 4, 0xFFFFFFFF);
            ST_DWORD(tbl + 8, 0x0FFFFFFF);   /* Reserve cluster #2 for root directory */
        }
        if (disk_write(pdrv, tbl, wsect++, 1) != RES_OK)
            return FR_DISK_ERR;
        mem_set(tbl, 0, SS(fs));             /* Fill following FAT entries with zero */
        for (n = 1; n < n_fat; n++) {        /* This loop may take a time on FAT32 volume due
        to many single sector writes */
            if (disk_write(pdrv, tbl, wsect++, 1) != RES_OK)
                return FR_DISK_ERR;
        }
    }

    /* Initialize root directory */
    i = (fmt == FS_FAT32) ? au : (UINT)n_dir;
    do {
        if (disk_write(pdrv, tbl, wsect++, 1) != RES_OK)
            return FR_DISK_ERR;
    } while (--i);

#if _USE_TRIM   /* Erase data area if needed */
```

```
    {
        eb[0] = wsect; eb[1] = wsect + (n_clst - ((fmt == FS_FAT32) ? 1 : 0)) * au - 1;
        disk_ioctl(pdrv, CTRL_TRIM, eb);
    }
#endif

    /* Create FSINFO if needed */
    if (fmt == FS_FAT32) {
        ST_DWORD(tbl + FSI_LeadSig, 0x41615252);
        ST_DWORD(tbl + FSI_StrucSig, 0x61417272);
        ST_DWORD(tbl + FSI_Free_Count, n_clst - 1); /* Number of free clusters */
        ST_DWORD(tbl + FSI_Nxt_Free, 2);            /* Last allocated cluster# */
        ST_WORD(tbl + BS_55AA, 0xAA55);
        disk_write(pdrv, tbl, b_vol + 1, 1);    /* Write original (VBR + 1) */
        disk_write(pdrv, tbl, b_vol + 7, 1);    /* Write backup (VBR + 7) */
    }

    return (disk_ioctl(pdrv, CTRL_SYNC, 0) == RES_OK) ? FR_OK : FR_DISK_ERR;
}



#if _MULTI_PARTITION
/*-----------------------------------------------------------------------*/
/* Create partition table on the physical drive                          */
/*-----------------------------------------------------------------------*/

FRESULT f_fdisk (
    BYTE pdrv,          /* Physical drive number */
    const DWORD szt[],  /* Pointer to the size table for each partitions */
    void* work          /* Pointer to the working buffer */
)
{
    UINT i, n, sz_cyl, tot_cyl, b_cyl, e_cyl, p_cyl;
    BYTE s_hd, e_hd, *p, *buf = (BYTE*)work;
    DSTATUS stat;
    DWORD sz_disk, sz_part, s_part;


    stat = disk_initialize(pdrv);
    if (stat & STA_NOINIT) return FR_NOT_READY;
    if (stat & STA_PROTECT) return FR_WRITE_PROTECTED;
    if (disk_ioctl(pdrv, GET_SECTOR_COUNT, &sz_disk)) return FR_DISK_ERR;

    /* Determine CHS in the table regardless of the drive geometry */
    for (n = 16; n < 256 && sz_disk / n / 63 > 1024; n *= 2) ;
    if (n == 256) n--;
    e_hd = n - 1;
    sz_cyl = 63 * n;
    tot_cyl = sz_disk / sz_cyl;

    /* Create partition table */
    mem_set(buf, 0, _MAX_SS);
    p = buf + MBR_Table; b_cyl = 0;
    for (i = 0; i < 4; i++, p += SZ_PTE) {
        p_cyl = (szt[i] <= 100U) ? (DWORD)tot_cyl * szt[i] / 100 : szt[i] / sz_cyl;
        if (!p_cyl) continue;
        s_part = (DWORD)sz_cyl * b_cyl;
        sz_part = (DWORD)sz_cyl * p_cyl;
        if (i == 0) {   /* Exclude first track of cylinder 0 */
            s_hd = 1;
            s_part += 63; sz_part -= 63;
        } else {
            s_hd = 0;
        }
        e_cyl = b_cyl + p_cyl - 1;
        if (e_cyl >= tot_cyl) return FR_INVALID_PARAMETER;

        /* Set partition table */
        p[1] = s_hd;                        /* Start head */
        p[2] = (BYTE)((b_cyl >> 2) + 1);    /* Start sector */
        p[3] = (BYTE)b_cyl;                 /* Start cylinder */
        p[4] = 0x06;                        /* System type (temporary setting) */
```

```
        p[5] = e_hd;                        /* End head */
        p[6] = (BYTE)((e_cyl >> 2) + 63);   /* End sector */
        p[7] = (BYTE)e_cyl;                 /* End cylinder */
        ST_DWORD(p + 8, s_part);            /* Start sector in LBA */
        ST_DWORD(p + 12, sz_part);          /* Partition size */

        /* Next partition */
        b_cyl += p_cyl;
    }
    ST_WORD(p, 0xAA55);

    /* Write it to the MBR */
    return (disk_write(pdrv, buf, 0, 1) != RES_OK || disk_ioctl(pdrv, CTRL_SYNC, 0) !=
    RES_OK) ? FR_DISK_ERR : FR_OK;
}


#endif /* _MULTI_PARTITION */
#endif /* _USE_MKFS && !_FS_READONLY */




#if _USE_STRFUNC
/*-----------------------------------------------------------------------*/
/* Get a string from the file                                            */
/*-----------------------------------------------------------------------*/

TCHAR* f_gets (
    TCHAR* buff,    /* Pointer to the string buffer to read */
    int len,        /* Size of string buffer (characters) */
    FIL* fp         /* Pointer to the file object */
)
{
    int n = 0;
    TCHAR c, *p = buff;
    BYTE s[2];
    UINT rc;


    while (n < len - 1) {   /* Read characters until buffer gets filled */
#if _USE_LFN && _LFN_UNICODE
#if _STRF_ENCODE == 3       /* Read a character in UTF-8 */
        f_read(fp, s, 1, &rc);
        if (rc != 1) break;
        c = s[0];
        if (c >= 0x80) {
            if (c < 0xC0) continue; /* Skip stray trailer */
            if (c < 0xE0) {         /* Two-byte sequence */
                f_read(fp, s, 1, &rc);
                if (rc != 1) break;
                c = (c & 0x1F) << 6 | (s[0] & 0x3F);
                if (c < 0x80) c = '?';
            } else {
                if (c < 0xF0) {     /* Three-byte sequence */
                    f_read(fp, s, 2, &rc);
                    if (rc != 2) break;
                    c = c << 12 | (s[0] & 0x3F) << 6 | (s[1] & 0x3F);
                    if (c < 0x800) c = '?';
                } else {            /* Reject four-byte sequence */
                    c = '?';
                }
            }
        }
#elif _STRF_ENCODE == 2     /* Read a character in UTF-16BE */
        f_read(fp, s, 2, &rc);
        if (rc != 2) break;
        c = s[1] + (s[0] << 8);
#elif _STRF_ENCODE == 1     /* Read a character in UTF-16LE */
        f_read(fp, s, 2, &rc);
        if (rc != 2) break;
        c = s[0] + (s[1] << 8);
#else                       /* Read a character in ANSI/OEM */
```

```
        f_read(fp, s, 1, &rc);
        if (rc != 1) break;
        c = s[0];
        if (IsDBCS1(c)) {
            f_read(fp, s, 1, &rc);
            if (rc != 1) break;
            c = (c << 8) + s[0];
        }
        c = ff_convert(c, 1);   /* OEM -> Unicode */
        if (!c) c = '?';
#endif
#else                       /* Read a character without conversion */
        f_read(fp, s, 1, &rc);
        if (rc != 1) break;
        c = s[0];
#endif
        if (_USE_STRFUNC == 2 && c == '\r') continue;   /* Strip '\r' */
        *p++ = c;
        n++;
        if (c == '\n') break;       /* Break on EOL */
    }
    *p = 0;
    return n ? buff : 0;            /* When no data read (eof or error), return with error. */
}




#if !_FS_READONLY
#include <stdarg.h>
/*-----------------------------------------------------------------------*/
/* Put a character to the file                                           */
/*-----------------------------------------------------------------------*/

typedef struct {
    FIL* fp;
    int idx, nchr;
    BYTE buf[64];
} putbuff;


static
void putc_bfd (
    putbuff* pb,
    TCHAR c
)
{
    UINT bw;
    int i;


    if (_USE_STRFUNC == 2 && c == '\n')  /* LF -> CRLF conversion */
        putc_bfd(pb, '\r');

    i = pb->idx;    /* Buffer write index (-1:error) */
    if (i < 0) return;

#if _USE_LFN && _LFN_UNICODE
#if _STRF_ENCODE == 3           /* Write a character in UTF-8 */
    if (c < 0x80) {             /* 7-bit */
        pb->buf[i++] = (BYTE)c;
    } else {
        if (c < 0x800) {        /* 11-bit */
            pb->buf[i++] = (BYTE)(0xC0 | c >> 6);
        } else {                /* 16-bit */
            pb->buf[i++] = (BYTE)(0xE0 | c >> 12);
            pb->buf[i++] = (BYTE)(0x80 | (c >> 6 & 0x3F));
        }
        pb->buf[i++] = (BYTE)(0x80 | (c & 0x3F));
    }
#elif _STRF_ENCODE == 2         /* Write a character in UTF-16BE */
    pb->buf[i++] = (BYTE)(c >> 8);
    pb->buf[i++] = (BYTE)c;
```

```
#elif _STRF_ENCODE == 1         /* Write a character in UTF-16LE */
    pb->buf[i++] = (BYTE)c;
    pb->buf[i++] = (BYTE)(c >> 8);
#else                           /* Write a character in ANSI/OEM */
    c = ff_convert(c, 0);   /* Unicode -> OEM */
    if (!c) c = '?';
    if (c >= 0x100)
        pb->buf[i++] = (BYTE)(c >> 8);
    pb->buf[i++] = (BYTE)c;
#endif
#else                           /* Write a character without conversion */
    pb->buf[i++] = (BYTE)c;
#endif

    if (i >= (int)(sizeof pb->buf) - 3) {   /* Write buffered characters to the file */
        f_write(pb->fp, pb->buf, (UINT)i, &bw);
        i = (bw == (UINT)i) ? 0 : -1;
    }
    pb->idx = i;
    pb->nchr++;
}



int f_putc (
    TCHAR c,    /* A character to be output */
    FIL* fp     /* Pointer to the file object */
)
{
    putbuff pb;
    UINT nw;


    pb.fp = fp;         /* Initialize output buffer */
    pb.nchr = pb.idx = 0;

    putc_bfd(&pb, c);   /* Put a character */

    if (   pb.idx >= 0  /* Flush buffered characters to the file */
        && f_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR_OK
        && (UINT)pb.idx == nw) return pb.nchr;
    return EOF;
}




/*-----------------------------------------------------------------------*/
/* Put a string to the file                                              */
/*-----------------------------------------------------------------------*/

int f_puts (
    const TCHAR* str,   /* Pointer to the string to be output */
    FIL* fp             /* Pointer to the file object */
)
{
    putbuff pb;
    UINT nw;


    pb.fp = fp;             /* Initialize output buffer */
    pb.nchr = pb.idx = 0;

    while (*str)            /* Put the string */
        putc_bfd(&pb, *str++);

    if (   pb.idx >= 0      /* Flush buffered characters to the file */
        && f_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR_OK
        && (UINT)pb.idx == nw) return pb.nchr;
    return EOF;
}
```

```
/*-----------------------------------------------------------------------*/
/* Put a formatted string to the file                                    */
/*-----------------------------------------------------------------------*/

int f_printf (
    FIL* fp,            /* Pointer to the file object */
    const TCHAR* fmt,   /* Pointer to the format string */
    ...                 /* Optional arguments... */
)
{
    va_list arp;
    BYTE f, r;
    UINT nw, i, j, w;
    DWORD v;
    TCHAR c, d, s[16], *p;
    putbuff pb;


    pb.fp = fp;             /* Initialize output buffer */
    pb.nchr = pb.idx = 0;

    va_start(arp, fmt);

    for (;;) {
        c = *fmt++;
        if (c == 0) break;          /* End of string */
        if (c != '%') {             /* Non escape character */
            putc_bfd(&pb, c);
            continue;
        }
        w = f = 0;
        c = *fmt++;
        if (c == '0') {             /* Flag: '0' padding */
            f = 1; c = *fmt++;
        } else {
            if (c == '-') {         /* Flag: left justified */
                f = 2; c = *fmt++;
            }
        }
        while (IsDigit(c)) {        /* Precision */
            w = w * 10 + c - '0';
            c = *fmt++;
        }
        if (c == 'l' || c == 'L') { /* Prefix: Size is long int */
            f |= 4; c = *fmt++;
        }
        if (!c) break;
        d = c;
        if (IsLower(d)) d -= 0x20;
        switch (d) {                /* Type is... */
        case 'S' :                  /* String */
            p = va_arg(arp, TCHAR*);
            for (j = 0; p[j]; j++) ;
            if (!(f & 2)) {
                while (j++ < w) putc_bfd(&pb, ' ');
            }
            while (*p) putc_bfd(&pb, *p++);
            while (j++ < w) putc_bfd(&pb, ' ');
            continue;
        case 'C' :                  /* Character */
            putc_bfd(&pb, (TCHAR)va_arg(arp, int)); continue;
        case 'B' :                  /* Binary */
            r = 2; break;
        case 'O' :                  /* Octal */
            r = 8; break;
        case 'D' :                  /* Signed decimal */
        case 'U' :                  /* Unsigned decimal */
            r = 10; break;
        case 'X' :                  /* Hexdecimal */
            r = 16; break;
        default:                    /* Unknown type (pass-through) */
```

```
            putc_bfd(&pb, c); continue;
        }

        /* Get an argument and put it in numeral */
        v = (f & 4) ? (DWORD)va_arg(arp, long) : ((d == 'D') ? (DWORD)(long)va_arg(arp, int)
        : (DWORD)va_arg(arp, unsigned int));
        if (d == 'D' && (v & 0x80000000)) {
            v = 0 - v;
            f |= 8;
        }
        i = 0;
        do {
            d = (TCHAR)(v % r); v /= r;
            if (d > 9) d += (c == 'x') ? 0x27 : 0x07;
            s[i++] = d + '0';
        } while (v && i < sizeof s / sizeof s[0]);
        if (f & 8) s[i++] = '-';
        j = i; d = (f & 1) ? '0' : ' ';
        while (!(f & 2) && j++ < w) putc_bfd(&pb, d);
        do putc_bfd(&pb, s[--i]); while (i);
        while (j++ < w) putc_bfd(&pb, d);
    }

    va_end(arp);

    if (   pb.idx >= 0      /* Flush buffered characters to the file */
        && f_write(pb.fp, pb.buf, (UINT)pb.idx, &nw) == FR_OK
        && (UINT)pb.idx == nw) return pb.nchr;
    return EOF;
}

#endif /* !_FS_READONLY */
#endif /* _USE_STRFUNC */
```

```
/*----------------------------------------------------------------------------/
/  FatFs - FAT file system module include R0.11     (C)ChaN, 2015
/-----------------------------------------------------------------------------/
/ FatFs module is a free software that opened under license policy of
/ following conditions.
/
/ Copyright (C) 2015, ChaN, all right reserved.
/
/ 1. Redistributions of source code must retain the above copyright notice,
/    this condition and the following disclaimer.
/
/ This software is provided by the copyright holder and contributors "AS IS"
/ and any warranties related to this software are DISCLAIMED.
/ The copyright owner or contributors be NOT LIABLE for any damages caused
/ by use of this software.
/----------------------------------------------------------------------------*/


#ifndef _FATFS
#define _FATFS  32020   /* Revision ID */

#ifdef __cplusplus
extern "C" {
#endif

#include "integer.h"    /* Basic integer types */
#include "ffconf.h"     /* FatFs configuration options */
#if _FATFS != _FFCONF
#error Wrong configuration file (ffconf.h).
#endif



/* Definitions of volume management */

#if _MULTI_PARTITION        /* Multiple partition configuration */
typedef struct {
    BYTE pd;    /* Physical drive number */
    BYTE pt;    /* Partition: 0:Auto detect, 1-4:Forced partition) */
} PARTITION;
extern PARTITION VolToPart[];   /* Volume - Partition resolution table */
#define LD2PD(vol) (VolToPart[vol].pd)  /* Get physical drive number */
#define LD2PT(vol) (VolToPart[vol].pt)  /* Get partition index */

#else                           /* Single partition configuration */
#define LD2PD(vol) (BYTE)(vol)  /* Each logical drive is bound to the same physical drive
number */
#define LD2PT(vol) 0            /* Find first valid partition or in SFD */

#endif



/* Type of path name strings on FatFs API */

#if _LFN_UNICODE            /* Unicode string */
#if !_USE_LFN
#error _LFN_UNICODE must be 0 at non-LFN cfg.
#endif
#ifndef _INC_TCHAR
typedef WCHAR TCHAR;
#define _T(x) L ## x
#define _TEXT(x) L ## x
#endif

#else                       /* ANSI/OEM string */
#ifndef _INC_TCHAR
typedef char TCHAR;
#define _T(x) x
#define _TEXT(x) x
#endif

#endif
```

```
/* File system object structure (FATFS) */

typedef struct {
  union{
    UINT    d32[_MAX_SS/4]; /* Force 32bits alignement */
    BYTE    d8[_MAX_SS];    /* Disk access window for Directory, FAT (and file data at tiny
    cfg) */
  }win;
    BYTE    fs_type;        /* FAT sub-type (0:Not mounted) */
    BYTE    drv;            /* Physical drive number */
    BYTE    csize;          /* Sectors per cluster (1,2,4...128) */
    BYTE    n_fats;         /* Number of FAT copies (1 or 2) */
    BYTE    wflag;          /* win[] flag (b0:dirty) */
    BYTE    fsi_flag;       /* FSINFO flags (b7:disabled, b0:dirty) */
    WORD    id;             /* File system mount ID */
    WORD    n_rootdir;      /* Number of root directory entries (FAT12/16) */
#if _MAX_SS != _MIN_SS
    WORD    ssize;          /* Bytes per sector (512, 1024, 2048 or 4096) */
#endif
#if _FS_REENTRANT
    _SYNC_t sobj;           /* Identifier of sync object */
#endif
#if !_FS_READONLY
    DWORD   last_clust;     /* Last allocated cluster */
    DWORD   free_clust;     /* Number of free clusters */
#endif
#if _FS_RPATH
    DWORD   cdir;           /* Current directory start cluster (0:root) */
#endif
    DWORD   n_fatent;       /* Number of FAT entries, = number of clusters + 2 */
    DWORD   fsize;          /* Sectors per FAT */
    DWORD   volbase;        /* Volume start sector */
    DWORD   fatbase;        /* FAT start sector */
    DWORD   dirbase;        /* Root directory start sector (FAT32:Cluster#) */
    DWORD   database;       /* Data start sector */
    DWORD   winsect;        /* Current sector appearing in the win[] */

} FATFS;



/* File object structure (FIL) */

typedef struct {
#if !_FS_TINY
  union{
    UINT    d32[_MAX_SS/4]; /* Force 32bits alignement */
    BYTE    d8[_MAX_SS];    /* File data read/write buffer */
  }buf;
#endif
    FATFS*  fs;             /* Pointer to the related file system object (**do not change
    order**) */
    WORD    id;             /* Owner file system mount ID (**do not change order**) */
    BYTE    flag;           /* Status flags */
    BYTE    err;            /* Abort flag (error code) */
    DWORD   fptr;           /* File read/write pointer (Zeroed on file open) */
    DWORD   fsize;          /* File size */
    DWORD   sclust;         /* File start cluster (0:no cluster chain, always 0 when fsize
    is 0) */
    DWORD   clust;          /* Current cluster of fpter (not valid when fprt is 0) */
    DWORD   dsect;          /* Sector number appearing in buf[] (0:invalid) */
#if !_FS_READONLY
    DWORD   dir_sect;       /* Sector number containing the directory entry */
    BYTE*   dir_ptr;        /* Pointer to the directory entry in the win[] */
#endif
#if _USE_FASTSEEK
    DWORD*  cltbl;          /* Pointer to the cluster link map table (Nulled on file open) */
#endif
#if _FS_LOCK
    UINT    lockid;         /* File lock ID origin from 1 (index of file semaphore table
```

```
    Files[]) */
#endif

} FIL;



/* Directory object structure (DIR) */

typedef struct {
#if !_FS_TINY
  union{
            UINT     d32[_MAX_SS/4];  /* Force 32bits alignement */
            BYTE   d8[_MAX_SS];  /* File data read/write buffer */
  }buf;
#endif
    FATFS*  fs;             /* Pointer to the owner file system object (**do not change
    order**) */
    WORD    id;             /* Owner file system mount ID (**do not change order**) */
    WORD    index;          /* Current read/write index number */
    DWORD   sclust;         /* Table start cluster (0:Root dir) */
    DWORD   clust;          /* Current cluster */
    DWORD   sect;           /* Current sector */
    BYTE*   dir;            /* Pointer to the current SFN entry in the win[] */
    BYTE*   fn;             /* Pointer to the SFN (in/out) {file[8],ext[3],status[1]} */
#if _FS_LOCK
    UINT    lockid;         /* File lock ID (index of file semaphore table Files[]) */
#endif
#if _USE_LFN
    WCHAR*  lfn;            /* Pointer to the LFN working buffer */
    WORD    lfn_idx;        /* Last matched LFN index number (0xFFFF:No LFN) */
#endif
#if _USE_FIND
    const TCHAR*    pat;    /* Pointer to the name matching pattern */
#endif
} DIR;



/* File information structure (FILINFO) */

typedef struct {
    DWORD   fsize;          /* File size */
    WORD    fdate;          /* Last modified date */
    WORD    ftime;          /* Last modified time */
    BYTE    fattrib;        /* Attribute */
    TCHAR   fname[13];      /* Short file name (8.3 format) */
#if _USE_LFN
    TCHAR*  lfname;         /* Pointer to the LFN buffer */
    UINT    lfsize;         /* Size of LFN buffer in TCHAR */
#endif
} FILINFO;



/* File function return code (FRESULT) */

typedef enum {
    FR_OK = 0,              /* (0) Succeeded */
    FR_DISK_ERR,            /* (1) A hard error occurred in the low level disk I/O layer */
    FR_INT_ERR,             /* (2) Assertion failed */
    FR_NOT_READY,           /* (3) The physical drive cannot work */
    FR_NO_FILE,             /* (4) Could not find the file */
    FR_NO_PATH,             /* (5) Could not find the path */
    FR_INVALID_NAME,        /* (6) The path name format is invalid */
    FR_DENIED,              /* (7) Access denied due to prohibited access or directory full */
    FR_EXIST,               /* (8) Access denied due to prohibited access */
    FR_INVALID_OBJECT,      /* (9) The file/directory object is invalid */
    FR_WRITE_PROTECTED,     /* (10) The physical drive is write protected */
    FR_INVALID_DRIVE,       /* (11) The logical drive number is invalid */
    FR_NOT_ENABLED,         /* (12) The volume has no work area */
    FR_NO_FILESYSTEM,       /* (13) There is no valid FAT volume */
    FR_MKFS_ABORTED,        /* (14) The f_mkfs() aborted due to any parameter error */
```

```
    FR_TIMEOUT,             /* (15) Could not get a grant to access the volume within
    defined period */
    FR_LOCKED,              /* (16) The operation is rejected according to the file sharing
    policy */
    FR_NOT_ENOUGH_CORE,     /* (17) LFN working buffer could not be allocated */
    FR_TOO_MANY_OPEN_FILES, /* (18) Number of open files > _FS_SHARE */
    FR_INVALID_PARAMETER    /* (19) Given parameter is invalid */
} FRESULT;



/*--------------------------------------------------------------*/
/* FatFs module application interface                           */

FRESULT f_open (FIL* fp, const TCHAR* path, BYTE mode);             /* Open or create a file
*/
FRESULT f_close (FIL* fp);                                          /* Close an open file
object */
FRESULT f_read (FIL* fp, void* buff, UINT btr, UINT* br);           /* Read data from a file
*/
FRESULT f_write (FIL* fp, const void* buff, UINT btw, UINT* bw);    /* Write data to a file */
FRESULT f_forward (FIL* fp, UINT(*func)(const BYTE*,UINT), UINT btf, UINT* bf); /* Forward
data to the stream */
FRESULT f_lseek (FIL* fp, DWORD ofs);                               /* Move file pointer of
a file object */
FRESULT f_truncate (FIL* fp);                                       /* Truncate file */
FRESULT f_sync (FIL* fp);                                           /* Flush cached data of
a writing file */
FRESULT f_opendir (DIR* dp, const TCHAR* path);                     /* Open a directory */
FRESULT f_closedir (DIR* dp);                                       /* Close an open
directory */
FRESULT f_readdir (DIR* dp, FILINFO* fno);                          /* Read a directory item
*/
FRESULT f_findfirst (DIR* dp, FILINFO* fno, const TCHAR* path, const TCHAR* pattern);   /*
Find first file */
FRESULT f_findnext (DIR* dp, FILINFO* fno);                         /* Find next file */
FRESULT f_mkdir (const TCHAR* path);                                /* Create a sub
directory */
FRESULT f_unlink (const TCHAR* path);                               /* Delete an existing
file or directory */
FRESULT f_rename (const TCHAR* path_old, const TCHAR* path_new);    /* Rename/Move a file or
directory */
FRESULT f_stat (const TCHAR* path, FILINFO* fno);                   /* Get file status */
FRESULT f_chmod (const TCHAR* path, BYTE attr, BYTE mask);          /* Change attribute of
the file/dir */
FRESULT f_utime (const TCHAR* path, const FILINFO* fno);            /* Change times-tamp of
the file/dir */
FRESULT f_chdir (const TCHAR* path);                                /* Change current
directory */
FRESULT f_chdrive (const TCHAR* path);                              /* Change current drive */
FRESULT f_getcwd (TCHAR* buff, UINT len);                           /* Get current directory
*/
FRESULT f_getfree (const TCHAR* path, DWORD* nclst, FATFS** fatfs); /* Get number of free
clusters on the drive */
FRESULT f_getlabel (const TCHAR* path, TCHAR* label, DWORD* vsn);   /* Get volume label */
FRESULT f_setlabel (const TCHAR* label);                            /* Set volume label */
FRESULT f_mount (FATFS* fs, const TCHAR* path, BYTE opt);           /* Mount/Unmount a
logical drive */
FRESULT f_mkfs (const TCHAR* path, BYTE sfd, UINT au);              /* Create a file system
on the volume */
FRESULT f_fdisk (BYTE pdrv, const DWORD szt[], void* work);         /* Divide a physical
drive into some partitions */
int f_putc (TCHAR c, FIL* fp);                                      /* Put a character to
the file */
int f_puts (const TCHAR* str, FIL* cp);                             /* Put a string to the
file */
int f_printf (FIL* fp, const TCHAR* str, ...);                      /* Put a formatted
string to the file */
TCHAR* f_gets (TCHAR* buff, int len, FIL* fp);                      /* Get a string from the
file */

#define f_eof(fp) ((int)((fp)->fptr == (fp)->fsize))
#define f_error(fp) ((fp)->err)
```

```
#define f_tell(fp) ((fp)->fptr)
#define f_size(fp) ((fp)->fsize)
#define f_rewind(fp) f_lseek((fp), 0)
#define f_rewinddir(dp) f_readdir((dp), 0)

#ifndef EOF
#define EOF (-1)
#endif




/*--------------------------------------------------------------*/
/* Additional user defined functions                            */

/* RTC function */
#if !_FS_READONLY && !_FS_NORTC
DWORD get_fattime (void);
#endif

/* Unicode support functions */
#if _USE_LFN                            /* Unicode - OEM code conversion */
WCHAR ff_convert (WCHAR chr, UINT dir); /* OEM-Unicode bidirectional conversion */
WCHAR ff_wtoupper (WCHAR chr);          /* Unicode upper-case conversion */
#if _USE_LFN == 3                       /* Memory functions */
void* ff_memalloc (UINT msize);         /* Allocate memory block */
void ff_memfree (void* mblock);         /* Free memory block */
#endif
#endif

/* Sync functions */
#if _FS_REENTRANT
int ff_cre_syncobj (BYTE vol, _SYNC_t* sobj);   /* Create a sync object */
int ff_req_grant (_SYNC_t sobj);                /* Lock sync object */
void ff_rel_grant (_SYNC_t sobj);               /* Unlock sync object */
int ff_del_syncobj (_SYNC_t sobj);              /* Delete a sync object */
#endif




/*--------------------------------------------------------------*/
/* Flags and offset address                                     */


/* File access control and file status flags (FIL.flag) */

#define FA_READ             0x01
#define FA_OPEN_EXISTING    0x00

#if !_FS_READONLY
#define FA_WRITE            0x02
#define FA_CREATE_NEW       0x04
#define FA_CREATE_ALWAYS    0x08
#define FA_OPEN_ALWAYS      0x10
#define FA__WRITTEN         0x20
#define FA__DIRTY           0x40
#endif


/* FAT sub type (FATFS.fs_type) */

#define FS_FAT12    1
#define FS_FAT16    2
#define FS_FAT32    3


/* File attribute bits for directory entry */

#define AM_RDO  0x01    /* Read only */
#define AM_HID  0x02    /* Hidden */
#define AM_SYS  0x04    /* System */
#define AM_VOL  0x08    /* Volume label */
```

```
#define AM_LFN  0x0F    /* LFN entry */
#define AM_DIR  0x10    /* Directory */
#define AM_ARC  0x20    /* Archive */
#define AM_MASK 0x3F    /* Mask of defined bits */


/* Fast seek feature */
#define CREATE_LINKMAP  0xFFFFFFFF



/*---------------------------------*/
/* Multi-byte word access macros  */

#if _WORD_ACCESS == 1   /* Enable word access to the FAT structure */
#define LD_WORD(ptr)         (WORD)(*(WORD*)(BYTE*)(ptr))
#define LD_DWORD(ptr)        (DWORD)(*(DWORD*)(BYTE*)(ptr))
#define ST_WORD(ptr,val)     *(WORD*)(BYTE*)(ptr)=(WORD)(val)
#define ST_DWORD(ptr,val)    *(DWORD*)(BYTE*)(ptr)=(DWORD)(val)
#else                   /* Use byte-by-byte access to the FAT structure */
#define LD_WORD(ptr)         (WORD)(((WORD)*((BYTE*)(ptr)+1)<<8)|(WORD)*(BYTE*)(ptr))
#define LD_DWORD(ptr)
(DWORD)(((DWORD)*((BYTE*)(ptr)+3)<<24)|((DWORD)*((BYTE*)(ptr)+2)<<16)|((WORD)*((BYTE*)(ptr)+1)
<<8)|*(BYTE*)(ptr))
#define ST_WORD(ptr,val)     *(BYTE*)(ptr)=(BYTE)(val);
*((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8)
#define ST_DWORD(ptr,val)    *(BYTE*)(ptr)=(BYTE)(val);
*((BYTE*)(ptr)+1)=(BYTE)((WORD)(val)>>8); *((BYTE*)(ptr)+2)=(BYTE)((DWORD)(val)>>16);
*((BYTE*)(ptr)+3)=(BYTE)((DWORD)(val)>>24)
#endif

#ifdef __cplusplus
}
#endif

#endif /* _FATFS */
```

```
/**
  ******************************************************************************
  * @file    ff_gen_drv.c
  * @author  MCD Application Team
  * @version V1.3.0
  * @date    08-May-2015
  * @brief   FatFs generic low level driver.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ff_gen_drv.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
Disk_drvTypeDef disk = {{0},{0},{0},0};

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/**
  * @brief  Links a compatible diskio driver/lun id and increments the number of active
  *         linked drivers.
  * @note   The number of linked drivers (volumes) is up to 10 due to FatFs limits.
  * @param  drv: pointer to the disk IO Driver structure
  * @param  path: pointer to the logical drive path
  * @param  lun : only used for USB Key Disk to add multi-lun management
            else the paramter must be equal to 0
  * @retval Returns 0 in case of success, otherwise 1.
  */
uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, uint8_t lun)
{
  uint8_t ret = 1;
  uint8_t DiskNum = 0;

  if(disk.nbr <= _VOLUMES)
  {
    disk.is_initialized[disk.nbr] = 0;
    disk.drv[disk.nbr] = drv;
    disk.lun[disk.nbr] = lun;
    DiskNum = disk.nbr++;
    path[0] = DiskNum + '0';
    path[1] = ':';
    path[2] = '/';
    path[3] = 0;
    ret = 0;
  }

  return ret;
}

/**
  * @brief  Links a compatible diskio driver and increments the number of active
  *         linked drivers.
  * @note   The number of linked drivers (volumes) is up to 10 due to FatFs limits
  * @param  drv: pointer to the disk IO Driver structure
  * @param  path: pointer to the logical drive path
  * @retval Returns 0 in case of success, otherwise 1.
  */
uint8_t FATFS_LinkDriver(Diskio_drvTypeDef *drv, char *path)
{
  return FATFS_LinkDriverEx(drv, path, 0);
}

/**
  * @brief  Unlinks a diskio driver and decrements the number of active linked
  *         drivers.
  * @param  path: pointer to the logical drive path
  * @param  lun : not used
  * @retval Returns 0 in case of success, otherwise 1.
  */
uint8_t FATFS_UnLinkDriverEx(char *path, uint8_t lun)
```

```
{
  uint8_t DiskNum = 0;
  uint8_t ret = 1;

  if(disk.nbr >= 1)
  {
    DiskNum = path[0] - '0';
    if(disk.drv[DiskNum] != 0)
    {
      disk.drv[DiskNum] = 0;
      disk.lun[DiskNum] = 0;
      disk.nbr--;
      ret = 0;
    }
  }

  return ret;
}

/**
  * @brief  Unlinks a diskio driver and decrements the number of active linked
  *         drivers.
  * @param  path: pointer to the logical drive path
  * @retval Returns 0 in case of success, otherwise 1.
  */
uint8_t FATFS_UnLinkDriver(char *path)
{
  return FATFS_UnLinkDriverEx(path, 0);
}

/**
  * @brief  Gets number of linked drivers to the FatFs module.
  * @param  None
  * @retval Number of attached drivers.
  */
uint8_t FATFS_GetAttachedDriversNbr(void)
{
  return disk.nbr;
}

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```c
/**
  ******************************************************************************
  * @file    ff_gen_drv.h
  * @author  MCD Application Team
  * @version V1.3.0
  * @date    08-May-2015
  * @brief   Header for ff_gen_drv.c module.
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __FF_GEN_DRV_H
#define __FF_GEN_DRV_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "diskio.h"
#include "ff.h"

/* Exported types ------------------------------------------------------------*/

/**
  * @brief  Disk IO Driver structure definition
  */
typedef struct
{
  DSTATUS (*disk_initialize) (BYTE);                    /*!< Initialize Disk
  Drive                     */
  DSTATUS (*disk_status)     (BYTE);                    /*!< Get Disk
  Status                         */
  DRESULT (*disk_read)       (BYTE, BYTE*, DWORD, UINT);       /*!< Read
  Sector(s)                             */
#if _USE_WRITE == 1
  DRESULT (*disk_write)      (BYTE, const BYTE*, DWORD, UINT); /*!< Write Sector(s) when
  _USE_WRITE = 0       */
#endif /* _USE_WRITE == 1 */
#if _USE_IOCTL == 1
  DRESULT (*disk_ioctl)      (BYTE, BYTE, void*);              /*!< I/O control operation
  when _USE_IOCTL = 1 */
#endif /* _USE_IOCTL == 1 */

}Diskio_drvTypeDef;

/**
  * @brief  Global Disk IO Drivers structure definition
  */
typedef struct
{
  uint8_t                 is_initialized[_VOLUMES];
  Diskio_drvTypeDef       *drv[_VOLUMES];
  uint8_t                 lun[_VOLUMES];
  __IO uint8_t            nbr;

}Disk_drvTypeDef;

/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, uint8_t lun);
uint8_t FATFS_LinkDriver(Diskio_drvTypeDef *drv, char *path);
uint8_t FATFS_UnLinkDriver(char *path);
uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, BYTE lun);
uint8_t FATFS_UnLinkDriverEx(char *path, BYTE lun);
uint8_t FATFS_GetAttachedDriversNbr(void);

#ifdef __cplusplus
}
#endif

#endif /* __FF_GEN_DRV_H */
```

```
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/*---------------------------------------------------------------------------/
/  FatFs - FAT file system module configuration file  R0.10b (C)ChaN, 2014
/---------------------------------------------------------------------------/
/
/ CAUTION! Do not forget to make clean the project after any changes to
/ the configuration options.
/
/----------------------------------------------------------------------------*/
#ifndef _FFCONF
#define _FFCONF 32020 /* Revision ID */

/*---------------------------------------------------------------------------/
/ Additional user header to be used
/----------------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
#include "stm32_adafruit_sd.h"


/*---------------------------------------------------------------------------/
/ Functions and Buffer Configurations
/----------------------------------------------------------------------------*/

#define _FS_TINY            0      /* 0:Normal or 1:Tiny */
/* When _FS_TINY is set to 1, FatFs uses the sector buffer in the file system
/  object instead of the sector buffer in the individual file object for file
/  data transfer. This reduces memory consumption 512 bytes each file object. */


#define _FS_READONLY        0      /* 0:Read/Write or 1:Read only */
/* Setting _FS_READONLY to 1 defines read only configuration. This removes
/  writing functions, f_write, f_sync, f_unlink, f_mkdir, f_chmod, f_rename,
/  f_truncate and useless f_getfree. */


#define _FS_MINIMIZE        0      /* 0 to 3 */
/* The _FS_MINIMIZE option defines minimization level to remove some functions.
/
/   0: Full function.
/   1: f_stat, f_getfree, f_unlink, f_mkdir, f_chmod, f_truncate, f_utime
/      and f_rename are removed.
/   2: f_opendir and f_readdir are removed in addition to 1.
/   3: f_lseek is removed in addition to 2. */


#define _USE_STRFUNC        2      /* 0:Disable or 1-2:Enable */
/* To enable string functions, set _USE_STRFUNC to 1 or 2. */


#define _USE_MKFS           1      /* 0:Disable or 1:Enable */
/* To enable f_mkfs function, set _USE_MKFS to 1 and set _FS_READONLY to 0 */


#define _USE_FASTSEEK       1      /* 0:Disable or 1:Enable */
/* To enable fast seek feature, set _USE_FASTSEEK to 1. */


#define _USE_LABEL          0      /* 0:Disable or 1:Enable */
/* To enable volume label functions, set _USE_LAVEL to 1 */


#define _USE_FORWARD        0      /* 0:Disable or 1:Enable */
/* To enable f_forward function, set _USE_FORWARD to 1 and set _FS_TINY to 1. */


/*---------------------------------------------------------------------------/
/ Local and Namespace Configurations
/----------------------------------------------------------------------------*/

#define _CODE_PAGE        1252
```

```
/* The _CODE_PAGE specifies the OEM code page to be used on the target system.
/  Incorrect setting of the code page can cause a file open failure.
/
/   932  - Japanese Shift-JIS (DBCS, OEM, Windows)
/   936  - Simplified Chinese GBK (DBCS, OEM, Windows)
/   949  - Korean (DBCS, OEM, Windows)
/   950  - Traditional Chinese Big5 (DBCS, OEM, Windows)
/   1250 - Central Europe (Windows)
/   1251 - Cyrillic (Windows)
/   1252 - Latin 1 (Windows)
/   1253 - Greek (Windows)
/   1254 - Turkish (Windows)
/   1255 - Hebrew (Windows)
/   1256 - Arabic (Windows)
/   1257 - Baltic (Windows)
/   1258 - Vietnam (OEM, Windows)
/   437  - U.S. (OEM)
/   720  - Arabic (OEM)
/   737  - Greek (OEM)
/   775  - Baltic (OEM)
/   850  - Multilingual Latin 1 (OEM)
/   858  - Multilingual Latin 1 + Euro (OEM)
/   852  - Latin 2 (OEM)
/   855  - Cyrillic (OEM)
/   866  - Russian (OEM)
/   857  - Turkish (OEM)
/   862  - Hebrew (OEM)
/   874  - Thai (OEM, Windows)
/ 1    - ASCII only (Valid for non LFN cfg.)
*/


#define _USE_LFN     0  /* 0 to 3 */
#define _MAX_LFN     255  /* Maximum LFN length to handle (12 to 255) */
/* The _USE_LFN option switches the LFN feature.
/
/   0: Disable LFN feature. _MAX_LFN has no effect.
/   1: Enable LFN with static working buffer on the BSS. Always NOT reentrant.
/   2: Enable LFN with dynamic working buffer on the STACK.
/   3: Enable LFN with dynamic working buffer on the HEAP.
/
/  To enable LFN feature, Unicode handling functions ff_convert() and ff_wtoupper()
/  function must be added to the project.
/  The LFN working buffer occupies (_MAX_LFN + 1) * 2 bytes. When use stack for the
/  working buffer, take care on stack overflow. When use heap memory for the working
/  buffer, memory management functions, ff_memalloc() and ff_memfree(), must be added
/  to the project. */


#define _LFN_UNICODE    0 /* 0:ANSI/OEM or 1:Unicode */
/* To switch the character encoding on the FatFs API to Unicode, enable LFN feature
/  and set _LFN_UNICODE to 1. */


#define _STRF_ENCODE    3 /* 0:ANSI/OEM, 1:UTF-16LE, 2:UTF-16BE, 3:UTF-8 */
/* When Unicode API is enabled, character encoding on the all FatFs API is switched
/  to Unicode. This option selects the character encoding on the file to be read/written
/  via string functions, f_gets(), f_putc(), f_puts and f_printf().
/  This option has no effect when _LFN_UNICODE is 0. */


#define _FS_RPATH       0 /* 0 to 2 */
/* The _FS_RPATH option configures relative path feature.
/
/   0: Disable relative path feature and remove related functions.
/   1: Enable relative path. f_chdrive() and f_chdir() function are available.
/   2: f_getcwd() function is available in addition to 1.
/
/  Note that output of the f_readdir() fnction is affected by this option. */
```

```
/*---------------------------------------------------------------------------/
/ Drive/Volume Configurations
/---------------------------------------------------------------------------*/

#define _VOLUMES    1
/* Number of volumes (logical drives) to be used. */


#define _MULTI_PARTITION     0 /* 0:Single partition, 1:Enable multiple partition */
/* When set to 0, each volume is bound to the same physical drive number and
/ it can mount only first primaly partition. When it is set to 1, each volume
/ is tied to the partitions listed in VolToPart[]. */


#define _MIN_SS                 512
#define _MAX_SS                 512
/* These options configure the range of sector size to be supported. (512, 1024, 2048 or
/  4096) Always set both 512 for most systems, all memory card and harddisk. But a larger
/  value may be required for on-board flash memory and some type of optical media.
/  When _MAX_SS is larger than _MIN_SS, FatFs is configured to variable sector size and
/  GET_SECTOR_SIZE command must be implemented to the disk_ioctl() function. */


#define _USE_ERASE     0 /* 0:Disable or 1:Enable */
/* To enable sector erase feature, set _USE_ERASE to 1. Also CTRL_ERASE_SECTOR command
/  should be added to the disk_ioctl() function. */


#define _FS_NOFSINFO    0 /* 0 or 1 */
/* If you need to know the correct free space on the FAT32 volume, set this
/  option to 1 and f_getfree() function at first time after volume mount will
/  force a full FAT scan.
/
/  0: Load all informations in the FSINFO if available.
/  1: Do not trust free cluster count in the FSINFO.
*/


/*---------------------------------------------------------------------------/
/ System Configurations
/---------------------------------------------------------------------------*/

#define _WORD_ACCESS    0 /* 0 or 1 */
/* The _WORD_ACCESS option is an only platform dependent option. It defines
/  which access method is used to the word data on the FAT volume.
/
/   0: Byte-by-byte access. Always compatible with all platforms.
/   1: Word access. Do not choose this unless under both the following conditions.
/
/  * Byte order on the memory is little-endian.
/  * Address miss-aligned word access is always allowed for all instructions.
/
/  If it is the case, _WORD_ACCESS can also be set to 1 to improve performance
/  and reduce code size.
*/


/* A header file that defines sync object types on the O/S, such as
/  windows.h, ucos_ii.h and semphr.h, must be included prior to ff.h. */

#define _FS_REENTRANT    0  /* 0:Disable or 1:Enable */
#define _FS_TIMEOUT      1000 /* Timeout period in unit of time ticks */
#define _SYNC_t          0 /* O/S dependent type of sync object. e.g. HANDLE, OS_EVENT*, ID and
etc.. */

/* The _FS_REENTRANT option switches the re-entrancy (thread safe) of the FatFs module.
/
```

```
/   0: Disable re-entrancy. _SYNC_t and _FS_TIMEOUT have no effect.
/   1: Enable re-entrancy. Also user provided synchronization handlers,
/      ff_req_grant(), ff_rel_grant(), ff_del_syncobj() and ff_cre_syncobj()
/      function must be added to the project. */


#define _FS_LOCK    15      /* 0:Disable or >=1:Enable */
/* To enable file lock control feature, set _FS_LOCK to 1 or greater.
   The value defines how many files can be opened simultaneously. */


#endif /* _FFCONFIG */
```

```
/**
  ******************************************************************************
  * @file    sd_diskio.c
  * @author  MCD Application Team
  * @version V1.3.0
  * @date    08-May-2015
  * @brief   SD Disk I/O driver
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include <string.h>
#include "ff_gen_drv.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Block Size in Bytes */
#define BLOCK_SIZE                512

/* Private variables ---------------------------------------------------------*/
/* Disk status */
static volatile DSTATUS Stat = STA_NOINIT;

/* Private function prototypes -----------------------------------------------*/
DSTATUS SD_initialize (BYTE);
DSTATUS SD_status (BYTE);
DRESULT SD_read (BYTE, BYTE*, DWORD, UINT);
#if _USE_WRITE == 1
  DRESULT SD_write (BYTE, const BYTE*, DWORD, UINT);
#endif /* _USE_WRITE == 1 */
#if _USE_IOCTL == 1
  DRESULT SD_ioctl (BYTE, BYTE, void*);
#endif  /* _USE_IOCTL == 1 */

const Diskio_drvTypeDef  SD_Driver =
{
  SD_initialize,
  SD_status,
  SD_read,
#if  _USE_WRITE == 1
  SD_write,
#endif /* _USE_WRITE == 1 */

#if  _USE_IOCTL == 1
  SD_ioctl,
#endif /* _USE_IOCTL == 1 */
};

/* Private functions ---------------------------------------------------------*/

/**
  * @brief  Initializes a Drive
  * @param  lun : not used
  * @retval DSTATUS: Operation status
  */
DSTATUS SD_initialize(BYTE lun)
{
  Stat = STA_NOINIT;

  /* Configure the uSD device */
  if(BSP_SD_Init() == MSD_OK)
  {
    Stat &= ~STA_NOINIT;
  }

  return Stat;
}

/**
  * @brief  Gets Disk Status
  * @param  lun : not used
  * @retval DSTATUS: Operation status
  */
```

```
DSTATUS SD_status(BYTE lun)
{
  Stat = STA_NOINIT;

  if(BSP_SD_GetStatus() == MSD_OK)
  {
    Stat &= ~STA_NOINIT;
  }

  return Stat;
}

/**
  * @brief  Reads Sector(s)
  * @param  lun : not used
  * @param  *buff: Data buffer to store read data
  * @param  sector: Sector address (LBA)
  * @param  count: Number of sectors to read (1..128)
  * @retval DRESULT: Operation result
  */
DRESULT SD_read(BYTE lun, BYTE *buff, DWORD sector, UINT count)
{
  DRESULT res = RES_OK;

  if(BSP_SD_ReadBlocks((uint32_t*)buff,
                       (uint64_t) (sector * BLOCK_SIZE),
                       BLOCK_SIZE,
                       count) != MSD_OK)
  {
    res = RES_ERROR;
  }

  return res;
}

/**
  * @brief  Writes Sector(s)
  * @param  lun : not used
  * @param  *buff: Data to be written
  * @param  sector: Sector address (LBA)
  * @param  count: Number of sectors to write (1..128)
  * @retval DRESULT: Operation result
  */
#if _USE_WRITE == 1
DRESULT SD_write(BYTE lun, const BYTE *buff, DWORD sector, UINT count)
{
  DRESULT res = RES_OK;

  if(BSP_SD_WriteBlocks((uint32_t*)buff,
                        (uint64_t)(sector * BLOCK_SIZE),
                        BLOCK_SIZE, count) != MSD_OK)
  {
    res = RES_ERROR;
  }

  return res;
}
#endif /* _USE_WRITE == 1 */

/**
  * @brief  I/O control operation
  * @param  lun : not used
  * @param  cmd: Control code
  * @param  *buff: Buffer to send/receive control data
  * @retval DRESULT: Operation result
  */
#if _USE_IOCTL == 1
DRESULT SD_ioctl(BYTE lun, BYTE cmd, void *buff)
{
  DRESULT res = RES_ERROR;
  SD_CardInfo CardInfo;

  if (Stat & STA_NOINIT) return RES_NOTRDY;
```

```
  switch (cmd)
  {
  /* Make sure that no pending write process */
  case CTRL_SYNC :
    res = RES_OK;
    break;

  /* Get number of sectors on the disk (DWORD) */
  case GET_SECTOR_COUNT :
    BSP_SD_GetCardInfo(&CardInfo);
    *(DWORD*)buff = CardInfo.CardCapacity / BLOCK_SIZE;
    res = RES_OK;
    break;

  /* Get R/W sector size (WORD) */
  case GET_SECTOR_SIZE :
    *(WORD*)buff = BLOCK_SIZE;
    res = RES_OK;
    break;

  /* Get erase block size in unit of sector (DWORD) */
  case GET_BLOCK_SIZE :
    *(DWORD*)buff = BLOCK_SIZE;
    break;

  default:
    res = RES_PARERR;
  }

  return res;
}
#endif /* _USE_IOCTL == 1 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    sd_diskio.h
  * @author  MCD Application Team
  * @version V1.3.0
  * @date    08-May-2015
  * @brief   Header for sd_diskio.c module
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __SD_DISKIO_H
#define __SD_DISKIO_H

/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
extern Diskio_drvTypeDef  SD_Driver;

#endif /* __SD_DISKIO_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    stm32_adafruit_sd.c
  * @author  MCD Application Team
  * @version V2.0.1
  * @date    04-November-2015
  * @brief   This file provides a set of functions needed to manage the SD card
  *          mounted on the Adafruit 1.8" TFT LCD shield (reference ID 802),
  *          that is used with the STM32 Nucleo board through SPI interface.
  *          It implements a high level communication layer for read and write
  *          from/to this memory. The needed STM32XXxx hardware resources (SPI and
  *          GPIO) are defined in stm32XXxx_nucleo.h file, and the initialization is
  *          performed in SD_IO_Init() function declared in stm32XXxx_nucleo.c
  *          file.
  *          You can easily tailor this driver to any other development board,
  *          by just adapting the defines for hardware resources and
  *          SD_IO_Init() function.
  *
  *          +-------------------------------------------------------+
  *          |                     Pin assignment                    |
  *          +-------------------------+---------------+-------------+
  *          |  STM32XXxx SPI Pins     |     SD        |    Pin      |
  *          +-------------------------+---------------+-------------+
  *          | SD_SPI_CS_PIN           |   ChipSelect  |    1        |
  *          | SD_SPI_MOSI_PIN / MOSI  |   DataIn      |    2        |
  *          |                         |   GND         |    3 (0 V)  |
  *          |                         |   VDD         |    4 (3.3 V)|
  *          | SD_SPI_SCK_PIN / SCLK   |   Clock       |    5        |
  *          |                         |   GND         |    6 (0 V)  |
  *          | SD_SPI_MISO_PIN / MISO  |   DataOut     |    7        |
  *          +-------------------------+---------------+-------------+
  ******************************************************************************
  */

/* File Info : -----------------------------------------------------------------
                                User NOTES
1. How to use this driver:
--------------------------
   - This driver does not need a specific component driver for the micro SD device
     to be included with.

2. Driver description:
---------------------
  + Initialization steps:
     o Initialize the micro SD card using the BSP_SD_Init() function.
     o Checking the SD card presence is not managed because SD detection pin is
       not physically mapped on the Adafruit shield.
     o The function BSP_SD_GetCardInfo() is used to get the micro SD card information
       which is stored in the structure "SD_CardInfo".

  + Micro SD card operations
     o The micro SD card can be accessed with read/write block(s) operations once
       it is ready for access. The access can be performed in polling
       mode by calling the functions BSP_SD_ReadBlocks()/BSP_SD_WriteBlocks()

     o The SD erase block(s) is performed using the function BSP_SD_Erase() with
       specifying the number of blocks to erase.
     o The SD runtime status is returned when calling the function BSP_SD_GetStatus().

------------------------------------------------------------------------------*/

/* Includes ------------------------------------------------------------------*/
#include "stm32_adafruit_sd.h"
#include "stdlib.h"
#include "string.h"
#include "stdio.h"

/** @addtogroup BSP
  * @{
  */

/** @addtogroup STM32_ADAFRUIT
  * @{
```

```
  */

/** @defgroup STM32_ADAFRUIT_SD
  * @{
  */

/* Private typedef -----------------------------------------------------------*/

/** @defgroup STM32_ADAFRUIT_SD_Private_Types_Definitions
  * @{
  */
typedef struct {
  uint8_t r1;
  uint8_t r2;
  uint8_t r3;
  uint8_t r4;
  uint8_t r5;
} SD_CmdAnswer_typedef;

/**
  * @}
  */

/* Private define ------------------------------------------------------------*/

/** @defgroup STM32_ADAFRUIT_SD_Private_Defines
  * @{
  */
#define SD_DUMMY_BYTE            0xFF

#define SD_MAX_FRAME_LENGTH        17    /* Lenght = 16 + 1 */
#define SD_CMD_LENGTH               6

#define SD_MAX_TRY                100    /* Number of try */

#define SD_CSD_STRUCT_V1          0x2    /* CSD struct version V1 */
#define SD_CSD_STRUCT_V2          0x1    /* CSD struct version V2 */


/**
  * @brief  SD ansewer format
  */
typedef enum {
 SD_ANSWER_R1_EXPECTED,
 SD_ANSWER_R1B_EXPECTED,
 SD_ANSWER_R2_EXPECTED,
 SD_ANSWER_R3_EXPECTED,
 SD_ANSWER_R4R5_EXPECTED,
 SD_ANSWER_R7_EXPECTED,
}SD_Answer_type;

/**
  * @brief  Start Data tokens:
  *         Tokens (necessary because at nop/idle (and CS active) only 0xff is
  *         on the data/command line)
  */
#define SD_TOKEN_START_DATA_SINGLE_BLOCK_READ    0xFE  /* Data token start byte, Start
Single Block Read */
#define SD_TOKEN_START_DATA_MULTIPLE_BLOCK_READ  0xFE  /* Data token start byte, Start
Multiple Block Read */
#define SD_TOKEN_START_DATA_SINGLE_BLOCK_WRITE   0xFE  /* Data token start byte, Start
Single Block Write */
#define SD_TOKEN_START_DATA_MULTIPLE_BLOCK_WRITE 0xFD  /* Data token start byte, Start
Multiple Block Write */
#define SD_TOKEN_STOP_DATA_MULTIPLE_BLOCK_WRITE  0xFD  /* Data toke stop byte, Stop Multiple
Block Write */

/**
  * @brief  Commands: CMDxx = CMD-number | 0x40
  */
#define SD_CMD_GO_IDLE_STATE          0   /* CMD0 = 0x40  */
#define SD_CMD_SEND_OP_COND           1   /* CMD1 = 0x41  */
#define SD_CMD_SEND_IF_COND           8   /* CMD8 = 0x48  */
```

```
#define SD_CMD_SEND_CSD                9   /* CMD9 = 0x49  */
#define SD_CMD_SEND_CID                10  /* CMD10 = 0x4A */
#define SD_CMD_STOP_TRANSMISSION       12  /* CMD12 = 0x4C */
#define SD_CMD_SEND_STATUS             13  /* CMD13 = 0x4D */
#define SD_CMD_SET_BLOCKLEN            16  /* CMD16 = 0x50 */
#define SD_CMD_READ_SINGLE_BLOCK       17  /* CMD17 = 0x51 */
#define SD_CMD_READ_MULT_BLOCK         18  /* CMD18 = 0x52 */
#define SD_CMD_SET_BLOCK_COUNT         23  /* CMD23 = 0x57 */
#define SD_CMD_WRITE_SINGLE_BLOCK      24  /* CMD24 = 0x58 */
#define SD_CMD_WRITE_MULT_BLOCK        25  /* CMD25 = 0x59 */
#define SD_CMD_PROG_CSD                27  /* CMD27 = 0x5B */
#define SD_CMD_SET_WRITE_PROT          28  /* CMD28 = 0x5C */
#define SD_CMD_CLR_WRITE_PROT          29  /* CMD29 = 0x5D */
#define SD_CMD_SEND_WRITE_PROT         30  /* CMD30 = 0x5E */
#define SD_CMD_SD_ERASE_GRP_START      32  /* CMD32 = 0x60 */
#define SD_CMD_SD_ERASE_GRP_END        33  /* CMD33 = 0x61 */
#define SD_CMD_UNTAG_SECTOR            34  /* CMD34 = 0x62 */
#define SD_CMD_ERASE_GRP_START         35  /* CMD35 = 0x63 */
#define SD_CMD_ERASE_GRP_END           36  /* CMD36 = 0x64 */
#define SD_CMD_UNTAG_ERASE_GROUP       37  /* CMD37 = 0x65 */
#define SD_CMD_ERASE                   38  /* CMD38 = 0x66 */
#define SD_CMD_SD_APP_OP_COND          41  /* CMD41 = 0x69 */
#define SD_CMD_APP_CMD                 55  /* CMD55 = 0x77 */
#define SD_CMD_READ_OCR                58  /* CMD55 = 0x79 */

/**
  * @brief  SD reponses and error flags
  */
typedef enum
{
/* R1 answer value */
  SD_R1_NO_ERROR              = (0x00),
  SD_R1_IN_IDLE_STATE         = (0x01),
  SD_R1_ERASE_RESET           = (0x02),
  SD_R1_ILLEGAL_COMMAND       = (0x04),
  SD_R1_COM_CRC_ERROR         = (0x08),
  SD_R1_ERASE_SEQUENCE_ERROR= (0x10),
  SD_R1_ADDRESS_ERROR         = (0x20),
  SD_R1_PARAMETER_ERROR       = (0x40),

/* R2 answer value */
  SD_R2_NO_ERROR              = 0x00,
  SD_R2_CARD_LOCKED           = 0x01,
  SD_R2_LOCKUNLOCK_ERROR      = 0x02,
  SD_R2_ERROR                 = 0x04,
  SD_R2_CC_ERROR              = 0x08,
  SD_R2_CARD_ECC_FAILED       = 0x10,
  SD_R2_WP_VIOLATION          = 0x20,
  SD_R2_ERASE_PARAM           = 0x40,
  SD_R2_OUTOFRANGE            = 0x80,

/**
  * @brief  Data response error
  */
  SD_DATA_OK                  = (0x05),
  SD_DATA_CRC_ERROR           = (0x0B),
  SD_DATA_WRITE_ERROR         = (0x0D),
  SD_DATA_OTHER_ERROR         = (0xFF)
} SD_Error;

/**
  * @}
  */

/* Private macro -------------------------------------------------------------*/

/** @defgroup STM32_ADAFRUIT_SD_Private_Macros
  * @{
  */

/**
  * @}
  */
```

```
/* Private variables ---------------------------------------------------------*/

/** @defgroup STM32_ADAFRUIT_SD_Private_Variables
  * @{
  */
__IO uint8_t SdStatus = SD_NOT_PRESENT;

/* flag_SDHC :
      0 :  Standard capacity
      1 : High capacity
*/
uint16_t flag_SDHC = 0;

/**
  * @}
  */

/* Private function prototypes -----------------------------------------------*/
static uint8_t SD_GetCIDRegister(SD_CID* Cid);
static uint8_t SD_GetCSDRegister(SD_CSD* Csd);
static uint8_t SD_GetDataResponse(void);
static uint8_t SD_GoIdleState(void);
static SD_CmdAnswer_typedef SD_SendCmd(uint8_t Cmd, uint32_t Arg, uint8_t Crc, uint8_t
Answer);
static uint8_t SD_WaitData(uint8_t data);
static uint8_t SD_ReadData(void);
/** @defgroup STM32_ADAFRUIT_SD_Private_Function_Prototypes
  * @{
  */
/**
  * @}
  */

/* Private functions ---------------------------------------------------------*/

/** @defgroup STM32_ADAFRUIT_SD_Private_Functions
  * @{
  */

/**
  * @brief  Initializes the SD/SD communication.
  * @param  None
  * @retval The SD Response:
  *         - MSD_ERROR: Sequence failed
  *         - MSD_OK: Sequence succeed
  */
uint8_t BSP_SD_Init(void)
{
  /* Configure IO functionalities for SD pin */
  SD_IO_Init();

  /* SD detection pin is not physically mapped on the Adafruit shield */
  SdStatus = SD_PRESENT;

  /* SD initialized and set to SPI mode properly */
  return SD_GoIdleState();
}

/**
  * @brief  Returns information about specific card.
  * @param  pCardInfo: Pointer to a SD_CardInfo structure that contains all SD
  *         card information.
  * @retval The SD Response:
  *         - MSD_ERROR: Sequence failed
  *         - MSD_OK: Sequence succeed
  */
uint8_t BSP_SD_GetCardInfo(SD_CardInfo *pCardInfo)
{
  uint8_t status;

  status = SD_GetCSDRegister(&(pCardInfo->Csd));
  status|= SD_GetCIDRegister(&(pCardInfo->Cid));
```

```
  if(flag_SDHC == 1 )
  {
    pCardInfo->CardBlockSize = 512;
    pCardInfo->CardCapacity = (pCardInfo->Csd.version.v2.DeviceSize + 1) *
    pCardInfo->CardBlockSize;
  }
  else
  {
    pCardInfo->CardCapacity = (pCardInfo->Csd.version.v1.DeviceSize + 1) ;
    pCardInfo->CardCapacity *= (1 << (pCardInfo->Csd.version.v1.DeviceSizeMul + 2));
    pCardInfo->CardBlockSize = 1 << (pCardInfo->Csd.RdBlockLen);
    pCardInfo->CardCapacity *= pCardInfo->CardBlockSize;
  }

  return status;
}

/**
  * @brief  Reads block(s) from a specified address in the SD card, in polling mode.
  * @param  pData: Pointer to the buffer that will contain the data to transmit
  * @param  ReadAddr: Address from where data is to be read
  * @param  BlockSize: SD card data block size, that should be 512
  * @param  NumOfBlocks: Number of SD blocks to read
  * @retval SD status
  */
uint8_t BSP_SD_ReadBlocks(uint32_t* pData, uint32_t ReadAddr, uint16_t BlockSize, uint32_t
NumberOfBlocks)
{
  uint32_t offset = 0;
  uint8_t retr = BSP_SD_ERROR;
  uint8_t *ptr = NULL;
  SD_CmdAnswer_typedef response;

  /* Send CMD16 (SD_CMD_SET_BLOCKLEN) to set the size of the block and
     Check if the SD acknowledged the set block length command: R1 response (0x00: no
     errors) */
  response = SD_SendCmd(SD_CMD_SET_BLOCKLEN, BlockSize, 0xFF, SD_ANSWER_R1_EXPECTED);
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);
  if ( response.r1 != SD_R1_NO_ERROR)
  {
     goto error;
  }

  ptr = malloc(sizeof(uint8_t)*BlockSize);
  if( ptr == NULL )
  {
     goto error;
  }
  memset(ptr, SD_DUMMY_BYTE, sizeof(uint8_t)*BlockSize);

  /* Data transfer */
  while (NumberOfBlocks--)
  {
    /* Send CMD17 (SD_CMD_READ_SINGLE_BLOCK) to read one block */
    /* Check if the SD acknowledged the read block command: R1 response (0x00: no errors) */
    response = SD_SendCmd(SD_CMD_READ_SINGLE_BLOCK, (ReadAddr + offset)/(flag_SDHC == 1
    ?BlockSize: 1), 0xFF, SD_ANSWER_R1_EXPECTED);
    if ( response.r1 != SD_R1_NO_ERROR)
    {
      goto error;
    }

    /* Now look for the data token to signify the start of the data */
    if (SD_WaitData(SD_TOKEN_START_DATA_SINGLE_BLOCK_READ) == BSP_SD_OK)
    {
      /* Read the SD block data : read NumByteToRead data */
      SD_IO_WriteReadData(ptr, (uint8_t*)pData + offset, BlockSize);

      /* Set next read address*/
      offset += BlockSize;
      /* get CRC bytes (not really needed by us, but required by SD) */
      SD_IO_WriteByte(SD_DUMMY_BYTE);
```

```
      SD_IO_WriteByte(SD_DUMMY_BYTE);
    }
    else
    {
      goto error;
    }

    /* End the command data read cycle */
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
  }

  retr = BSP_SD_OK;

error :
  /* Send dummy byte: 8 Clock pulses of delay */
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);
  if(ptr != NULL) free(ptr);

  /* Return the reponse */
  return retr;
}

/**
  * @brief  Writes block(s) to a specified address in the SD card, in polling mode.
  * @param  pData: Pointer to the buffer that will contain the data to transmit
  * @param  WriteAddr: Address from where data is to be written
  * @param  BlockSize: SD card data block size, that should be 512
  * @param  NumOfBlocks: Number of SD blocks to write
  * @retval SD status
  */
uint8_t BSP_SD_WriteBlocks(uint32_t* pData, uint32_t WriteAddr, uint16_t BlockSize, uint32_t
NumberOfBlocks)
{
  uint32_t offset = 0;
  uint8_t retr = BSP_SD_ERROR;
  uint8_t *ptr = NULL;
  SD_CmdAnswer_typedef response;

  /* Send CMD16 (SD_CMD_SET_BLOCKLEN) to set the size of the block and
     Check if the SD acknowledged the set block length command: R1 response (0x00: no
     errors) */
  response = SD_SendCmd(SD_CMD_SET_BLOCKLEN, BlockSize, 0xFF, SD_ANSWER_R1_EXPECTED);
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);
  if ( response.r1 != SD_R1_NO_ERROR)
  {
    goto error;
  }

  ptr = malloc(sizeof(uint8_t)*BlockSize);
  if (ptr == NULL)
  {
    goto error;
  }

  /* Data transfer */
  while (NumberOfBlocks--)
  {
    /* Send CMD24 (SD_CMD_WRITE_SINGLE_BLOCK) to write blocks  and
       Check if the SD acknowledged the write block command: R1 response (0x00: no errors) */
    response = SD_SendCmd(SD_CMD_WRITE_SINGLE_BLOCK, (WriteAddr + offset)/(flag_SDHC == 1 ?
    BlockSize: 1), 0xFF, SD_ANSWER_R1_EXPECTED);
    if (response.r1 != SD_R1_NO_ERROR)
    {
      goto error;
    }

    /* Send dummy byte for NWR timing : one byte between CMDWRITE and TOKEN */
    SD_IO_WriteByte(SD_DUMMY_BYTE);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
```

```
    /* Send the data token to signify the start of the data */
    SD_IO_WriteByte(SD_TOKEN_START_DATA_SINGLE_BLOCK_WRITE);

    /* Write the block data to SD */
    SD_IO_WriteReadData((uint8_t*)pData + offset, ptr, BlockSize);

    /* Set next write address */
    offset += BlockSize;

    /* Put CRC bytes (not really needed by us, but required by SD) */
    SD_IO_WriteByte(SD_DUMMY_BYTE);
    SD_IO_WriteByte(SD_DUMMY_BYTE);

    /* Read data response */
    if (SD_GetDataResponse() != SD_DATA_OK)
    {
      /* Set response value to failure */
      goto error;
    }

    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
  }
  retr = BSP_SD_OK;

error :
  if(ptr != NULL) free(ptr);
  /* Send dummy byte: 8 Clock pulses of delay */
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);

  /* Return the reponse */
  return retr;
}

/**
  * @brief  Erases the specified memory area of the given SD card.
  * @param  StartAddr: Start byte address
  * @param  EndAddr: End byte address
  * @retval SD status
  */
uint8_t BSP_SD_Erase(uint32_t StartAddr, uint32_t EndAddr)
{
  uint8_t retr = BSP_SD_ERROR;
  SD_CmdAnswer_typedef response;

  /* Send CMD32 (Erase group start) and check if the SD acknowledged the erase command: R1
  response (0x00: no errors) */
  response = SD_SendCmd(SD_CMD_SD_ERASE_GRP_START, StartAddr, 0xFF, SD_ANSWER_R1_EXPECTED);
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);  if (response.r1 == SD_R1_NO_ERROR)
  {
    /* Send CMD33 (Erase group end) and Check if the SD acknowledged the erase command: R1
    response (0x00: no errors) */
    response = SD_SendCmd(SD_CMD_SD_ERASE_GRP_END, EndAddr, 0xFF, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
    if (response.r1 == SD_R1_NO_ERROR)
    {
      /* Send CMD38 (Erase) and Check if the SD acknowledged the erase command: R1 response
      (0x00: no errors) */
      response = SD_SendCmd(SD_CMD_ERASE, 0, 0xFF, SD_ANSWER_R1B_EXPECTED);
      if (response.r1 == SD_R1_NO_ERROR)
      {
        retr = BSP_SD_OK;
      }
      SD_IO_CSState(1);
      SD_IO_WriteByte(SD_DUMMY_BYTE);
    }
  }

  /* Return the reponse */
  return retr;
```

```
}

/**
  * @brief  Returns the SD status.
  * @param  None
  * @retval The SD status.
  */
uint8_t BSP_SD_GetStatus(void)
{
  SD_CmdAnswer_typedef retr;

  /* Send CMD13 (SD_SEND_STATUS) to get SD status */
  retr = SD_SendCmd(SD_CMD_SEND_STATUS, 0, 0xFF, SD_ANSWER_R2_EXPECTED);
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);

  /* Find SD status according to card state */
  if(( retr.r1 == SD_R1_NO_ERROR) && ( retr.r2 == SD_R2_NO_ERROR))
  {
    return BSP_SD_OK;
  }

  return BSP_SD_ERROR;
}

/**
  * @brief  Reads the SD card SCD register.
  *         Reading the contents of the CSD register in SPI mode is a simple
  *         read-block transaction.
  * @param  Csd: pointer on an SCD register structure
  * @retval SD status
  */
uint8_t SD_GetCSDRegister(SD_CSD* Csd)
{
  uint16_t counter = 0;
  uint8_t CSD_Tab[16];
  uint8_t retr = BSP_SD_ERROR;
  SD_CmdAnswer_typedef response;

  /* Send CMD9 (CSD register) or CMD10(CSD register) and Wait for response in the R1 format
  (0x00 is no errors) */
  response = SD_SendCmd(SD_CMD_SEND_CSD, 0, 0xFF, SD_ANSWER_R1_EXPECTED);
  if(response.r1 == SD_R1_NO_ERROR)
  {
    if (SD_WaitData(SD_TOKEN_START_DATA_SINGLE_BLOCK_READ) == BSP_SD_OK)
    {
      for (counter = 0; counter < 16; counter++)
      {
        /* Store CSD register value on CSD_Tab */
        CSD_Tab[counter] = SD_IO_WriteByte(SD_DUMMY_BYTE);
      }

      /* Get CRC bytes (not really needed by us, but required by SD) */
      SD_IO_WriteByte(SD_DUMMY_BYTE);
      SD_IO_WriteByte(SD_DUMMY_BYTE);

      /*************************************************************************
        CSD header decoding
      *************************************************************************/

      /* Byte 0 */
      Csd->CSDStruct = (CSD_Tab[0] & 0xC0) >> 6;
      Csd->Reserved1 =  CSD_Tab[0] & 0x3F;

      /* Byte 1 */
      Csd->TAAC = CSD_Tab[1];

      /* Byte 2 */
      Csd->NSAC = CSD_Tab[2];

      /* Byte 3 */
      Csd->MaxBusClkFrec = CSD_Tab[3];
```

```
      /* Byte 4/5 */
      Csd->CardComdClasses = (CSD_Tab[4] << 4) | ((CSD_Tab[5] & 0xF0) >> 4);
      Csd->RdBlockLen = CSD_Tab[5] & 0x0F;

      /* Byte 6 */
      Csd->PartBlockRead   = (CSD_Tab[6] & 0x80) >> 7;
      Csd->WrBlockMisalign = (CSD_Tab[6] & 0x40) >> 6;
      Csd->RdBlockMisalign = (CSD_Tab[6] & 0x20) >> 5;
      Csd->DSRImpl         = (CSD_Tab[6] & 0x10) >> 4;

      /*****************************************************************************
        CSD v1/v2 decoding
      *****************************************************************************/

      if(flag_SDHC == 0)
      {
        Csd->version.v1.Reserved1 = ((CSD_Tab[6] & 0x0C) >> 2);

        Csd->version.v1.DeviceSize =  ((CSD_Tab[6] & 0x03) << 10)
                                    |  (CSD_Tab[7] << 2)
                                    | ((CSD_Tab[8] & 0xC0) >> 6);
        Csd->version.v1.MaxRdCurrentVDDMin = (CSD_Tab[8] & 0x38) >> 3;
        Csd->version.v1.MaxRdCurrentVDDMax = (CSD_Tab[8] & 0x07);
        Csd->version.v1.MaxWrCurrentVDDMin = (CSD_Tab[9] & 0xE0) >> 5;
        Csd->version.v1.MaxWrCurrentVDDMax = (CSD_Tab[9] & 0x1C) >> 2;
        Csd->version.v1.DeviceSizeMul = ((CSD_Tab[9] & 0x03) << 1)
                                       |((CSD_Tab[10] & 0x80) >> 7);
      }
      else
      {
        Csd->version.v2.Reserved1 = ((CSD_Tab[6] & 0x0F) << 2) | ((CSD_Tab[7] & 0xC0) >> 6);
        Csd->version.v2.DeviceSize= ((CSD_Tab[7] & 0x3F) << 16) | (CSD_Tab[8] << 8) |
        CSD_Tab[9];
        Csd->version.v2.Reserved2 = ((CSD_Tab[10] & 0x80) >> 8);
      }

      Csd->EraseSingleBlockEnable = (CSD_Tab[10] & 0x40) >> 6;
      Csd->EraseSectorSize   = ((CSD_Tab[10] & 0x3F) << 1)
                              |((CSD_Tab[11] & 0x80) >> 7);
      Csd->WrProtectGrSize   = (CSD_Tab[11] & 0x7F);
      Csd->WrProtectGrEnable = (CSD_Tab[12] & 0x80) >> 7;
      Csd->Reserved2         = (CSD_Tab[12] & 0x60) >> 5;
      Csd->WrSpeedFact       = (CSD_Tab[12] & 0x1C) >> 2;
      Csd->MaxWrBlockLen     = ((CSD_Tab[12] & 0x03) << 2)
                              |((CSD_Tab[13] & 0xC0) >> 6);
      Csd->WriteBlockPartial = (CSD_Tab[13] & 0x20) >> 5;
      Csd->Reserved3         = (CSD_Tab[13] & 0x1F);
      Csd->FileFormatGrouop  = (CSD_Tab[14] & 0x80) >> 7;
      Csd->CopyFlag          = (CSD_Tab[14] & 0x40) >> 6;
      Csd->PermWrProtect     = (CSD_Tab[14] & 0x20) >> 5;
      Csd->TempWrProtect     = (CSD_Tab[14] & 0x10) >> 4;
      Csd->FileFormat        = (CSD_Tab[14] & 0x0C) >> 2;
      Csd->Reserved4         = (CSD_Tab[14] & 0x03);
      Csd->crc               = (CSD_Tab[15] & 0xFE) >> 1;
      Csd->Reserved5         = (CSD_Tab[15] & 0x01);

      retr = BSP_SD_OK;
    }
  }

  /* Send dummy byte: 8 Clock pulses of delay */
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);

  /* Return the reponse */
  return retr;
}

/**
  * @brief  Reads the SD card CID register.
  *         Reading the contents of the CID register in SPI mode is a simple
  *         read-block transaction.
  * @param  Cid: pointer on an CID register structure
```

```
  * @retval SD status
  */
uint8_t SD_GetCIDRegister(SD_CID* Cid)
{
  uint32_t counter = 0;
  uint8_t retr = BSP_SD_ERROR;
  uint8_t CID_Tab[16];
  SD_CmdAnswer_typedef response;

  /* Send CMD10 (CID register) and Wait for response in the R1 format (0x00 is no errors) */
  response = SD_SendCmd(SD_CMD_SEND_CID, 0, 0xFF, SD_ANSWER_R1_EXPECTED);
  if(response.r1 == SD_R1_NO_ERROR)
  {
    if(SD_WaitData(SD_TOKEN_START_DATA_SINGLE_BLOCK_READ) == BSP_SD_OK)
    {
      /* Store CID register value on CID_Tab */
      for (counter = 0; counter < 16; counter++)
      {
        CID_Tab[counter] = SD_IO_WriteByte(SD_DUMMY_BYTE);
      }

      /* Get CRC bytes (not really needed by us, but required by SD) */
      SD_IO_WriteByte(SD_DUMMY_BYTE);
      SD_IO_WriteByte(SD_DUMMY_BYTE);

      /* Byte 0 */
      Cid->ManufacturerID = CID_Tab[0];

      /* Byte 1 */
      Cid->OEM_AppliID = CID_Tab[1] << 8;

      /* Byte 2 */
      Cid->OEM_AppliID |= CID_Tab[2];

      /* Byte 3 */
      Cid->ProdName1 = CID_Tab[3] << 24;

      /* Byte 4 */
      Cid->ProdName1 |= CID_Tab[4] << 16;

      /* Byte 5 */
      Cid->ProdName1 |= CID_Tab[5] << 8;

      /* Byte 6 */
      Cid->ProdName1 |= CID_Tab[6];

      /* Byte 7 */
      Cid->ProdName2 = CID_Tab[7];

      /* Byte 8 */
      Cid->ProdRev = CID_Tab[8];

      /* Byte 9 */
      Cid->ProdSN = CID_Tab[9] << 24;

      /* Byte 10 */
      Cid->ProdSN |= CID_Tab[10] << 16;

      /* Byte 11 */
      Cid->ProdSN |= CID_Tab[11] << 8;

      /* Byte 12 */
      Cid->ProdSN |= CID_Tab[12];

      /* Byte 13 */
      Cid->Reserved1 |= (CID_Tab[13] & 0xF0) >> 4;
      Cid->ManufactDate = (CID_Tab[13] & 0x0F) << 8;

      /* Byte 14 */
      Cid->ManufactDate |= CID_Tab[14];

      /* Byte 15 */
      Cid->CID_CRC = (CID_Tab[15] & 0xFE) >> 1;
```

```
      Cid->Reserved2 = 1;

      retr = BSP_SD_OK;
    }
  }

  /* Send dummy byte: 8 Clock pulses of delay */
  SD_IO_CSState(1);
  SD_IO_WriteByte(SD_DUMMY_BYTE);

  /* Return the reponse */
  return retr;
}

/**
  * @brief  Sends 5 bytes command to the SD card and get response
  * @param  Cmd: The user expected command to send to SD card.
  * @param  Arg: The command argument.
  * @param  Crc: The CRC.
  * @param  Answer: SD_ANSWER_NOT_EXPECTED or SD_ANSWER_EXPECTED
  * @retval SD status
  */
SD_CmdAnswer_typedef SD_SendCmd(uint8_t Cmd, uint32_t Arg, uint8_t Crc, uint8_t Answer)
{
  uint8_t frame[SD_CMD_LENGTH], frameout[SD_CMD_LENGTH];
  SD_CmdAnswer_typedef retr = {0xFF, 0xFF , 0xFF, 0xFF, 0xFF};

  /* R1 Lenght = NCS(0)+ 6 Bytes command + NCR(min1 max8) + 1 Bytes answer + NEC(0) =
  15bytes */
  /* R1b identical to R1 + Busy
  information                                                    */
  /* R2 Lenght = NCS(0)+ 6 Bytes command + NCR(min1 max8) + 2 Bytes answer + NEC(0) =
  16bytes */

  /* Prepare Frame to send */
  frame[0] = (Cmd | 0x40);         /* Construct byte 1 */
  frame[1] = (uint8_t)(Arg >> 24); /* Construct byte 2 */
  frame[2] = (uint8_t)(Arg >> 16); /* Construct byte 3 */
  frame[3] = (uint8_t)(Arg >> 8);  /* Construct byte 4 */
  frame[4] = (uint8_t)(Arg);       /* Construct byte 5 */
  frame[5] = (Crc | 0x01);         /* Construct byte 6 */

  /* Send the command */
  SD_IO_CSState(0);
  SD_IO_WriteReadData(frame, frameout, SD_CMD_LENGTH); /* Send the Cmd bytes */

  switch(Answer)
  {
  case SD_ANSWER_R1_EXPECTED :
    retr.r1 = SD_ReadData();
    break;
  case SD_ANSWER_R1B_EXPECTED :
    retr.r1 = SD_ReadData();
    retr.r2 = SD_IO_WriteByte(SD_DUMMY_BYTE);
    /* Set CS High */
    SD_IO_CSState(1);
    HAL_Delay(1);
    /* Set CS Low */
    SD_IO_CSState(0);

    /* Wait IO line return 0xFF */
    while (SD_IO_WriteByte(SD_DUMMY_BYTE) != 0xFF);
    break;
  case SD_ANSWER_R2_EXPECTED :
    retr.r1 = SD_ReadData();
    retr.r2 = SD_IO_WriteByte(SD_DUMMY_BYTE);
    break;
  case SD_ANSWER_R3_EXPECTED :
  case SD_ANSWER_R7_EXPECTED :
    retr.r1 = SD_ReadData();
    retr.r2 = SD_IO_WriteByte(SD_DUMMY_BYTE);
    retr.r3 = SD_IO_WriteByte(SD_DUMMY_BYTE);
    retr.r4 = SD_IO_WriteByte(SD_DUMMY_BYTE);
```

```
    retr.r5 = SD_IO_WriteByte(SD_DUMMY_BYTE);
    break;
  default :
    break;
  }
  return retr;
}

/**
  * @brief  Gets the SD card data response and check the busy flag.
  * @param  None
  * @retval The SD status: Read data response xxx0<status>1
  *         - status 010: Data accecpted
  *         - status 101: Data rejected due to a crc error
  *         - status 110: Data rejected due to a Write error.
  *         - status 111: Data rejected due to other error.
  */
uint8_t SD_GetDataResponse(void)
{
  uint8_t dataresponse;
  uint8_t rvalue = SD_DATA_OTHER_ERROR;

  dataresponse = SD_IO_WriteByte(SD_DUMMY_BYTE);
  SD_IO_WriteByte(SD_DUMMY_BYTE); /* read the busy response byte*/

  /* Mask unused bits */
  switch (dataresponse & 0x1F)
  {
  case SD_DATA_OK:
    rvalue = SD_DATA_OK;

    /* Set CS High */
    SD_IO_CSState(1);
    /* Set CS Low */
    SD_IO_CSState(0);

    /* Wait IO line return 0xFF */
    while (SD_IO_WriteByte(SD_DUMMY_BYTE) != 0xFF);
    break;
  case SD_DATA_CRC_ERROR:
    rvalue =  SD_DATA_CRC_ERROR;
    break;
  case SD_DATA_WRITE_ERROR:
    rvalue = SD_DATA_WRITE_ERROR;
    break;
  default:
    break;
  }

  /* Return response */
  return rvalue;
}


/**
  * @brief  Put the SD in Idle state.
  * @param  None
  * @retval SD status
  */
uint8_t SD_GoIdleState(void)
{
  SD_CmdAnswer_typedef response;
  __IO uint8_t counter = 0;
  /* Send CMD0 (SD_CMD_GO_IDLE_STATE) to put SD in SPI mode and
     wait for In Idle State Response (R1 Format) equal to 0x01 */
  do{
    counter++;
    response = SD_SendCmd(SD_CMD_GO_IDLE_STATE, 0, 0x95, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
    if(counter >= SD_MAX_TRY)
    {
      return BSP_SD_ERROR;
```

```
  }
}
while(response.r1 != SD_R1_IN_IDLE_STATE);


/* Send CMD8 (SD_CMD_SEND_IF_COND) to check the power supply status
   and wait until response (R7 Format) equal to 0xAA and */
response = SD_SendCmd(SD_CMD_SEND_IF_COND, 0x1AA, 0x87, SD_ANSWER_R7_EXPECTED);
SD_IO_CSState(1);
SD_IO_WriteByte(SD_DUMMY_BYTE);
if((response.r1  & SD_R1_ILLEGAL_COMMAND) == SD_R1_ILLEGAL_COMMAND)
{
  /* initialise card V1 */
  do
  {
    /* initialise card V1 */
    /* Send CMD55 (SD_CMD_APP_CMD) before any ACMD command: R1 response (0x00: no errors)
    */
    response = SD_SendCmd(SD_CMD_APP_CMD, 0x00000000, 0xFF, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);

    /* Send ACMD41 (SD_CMD_SD_APP_OP_COND) to initialize SDHC or SDXC cards: R1 response
    (0x00: no errors) */
    response = SD_SendCmd(SD_CMD_SD_APP_OP_COND, 0x00000000, 0xFF, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
  }
  while(response.r1 == SD_R1_IN_IDLE_STATE);
  flag_SDHC = 0;
}
else if(response.r1 == SD_R1_IN_IDLE_STATE)
{
    /* initialise card V2 */
  do {

    /* Send CMD55 (SD_CMD_APP_CMD) before any ACMD command: R1 response (0x00: no errors)
    */
    response = SD_SendCmd(SD_CMD_APP_CMD, 0, 0xFF, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);

    /* Send ACMD41 (SD_CMD_SD_APP_OP_COND) to initialize SDHC or SDXC cards: R1 response
    (0x00: no errors) */
    response = SD_SendCmd(SD_CMD_SD_APP_OP_COND, 0x40000000, 0xFF, SD_ANSWER_R1_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
  }
  while(response.r1 == SD_R1_IN_IDLE_STATE);

  if((response.r1 & SD_R1_ILLEGAL_COMMAND) == SD_R1_ILLEGAL_COMMAND)
  {
    do {
      /* Send CMD55 (SD_CMD_APP_CMD) before any ACMD command: R1 response (0x00: no
      errors) */
      response = SD_SendCmd(SD_CMD_APP_CMD, 0, 0xFF, SD_ANSWER_R1_EXPECTED);
      SD_IO_CSState(1);
      SD_IO_WriteByte(SD_DUMMY_BYTE);
      if(response.r1 != SD_R1_IN_IDLE_STATE)
      {
        return BSP_SD_ERROR;
      }
      /* Send ACMD41 (SD_CMD_SD_APP_OP_COND) to initialize SDHC or SDXC cards: R1 response
      (0x00: no errors) */
      response = SD_SendCmd(SD_CMD_SD_APP_OP_COND, 0x00000000, 0xFF, SD_ANSWER_R1_EXPECTED);
      SD_IO_CSState(1);
      SD_IO_WriteByte(SD_DUMMY_BYTE);
    }
    while(response.r1 == SD_R1_IN_IDLE_STATE);
  }

  /* Send CMD58 (SD_CMD_READ_OCR) to initialize SDHC or SDXC cards: R3 response (0x00: no
  errors) */
```

```
    response = SD_SendCmd(SD_CMD_READ_OCR, 0x00000000, 0xFF, SD_ANSWER_R3_EXPECTED);
    SD_IO_CSState(1);
    SD_IO_WriteByte(SD_DUMMY_BYTE);
    if(response.r1 != SD_R1_NO_ERROR)
    {
      return BSP_SD_ERROR;
    }
    flag_SDHC = (response.r2 & 0x40) >> 6;
  }
  else
  {
    return BSP_SD_ERROR;
  }

  return BSP_SD_OK;
}

/**
  * @brief  Waits a data until a value different from SD_DUMMY_BITE
  * @param  None
  * @retval the value read
  */
uint8_t SD_ReadData(void)
{
  uint8_t timeout = 0x08;
  uint8_t readvalue;

  /* Check if response is got or a timeout is happen */
  do {
    readvalue = SD_IO_WriteByte(SD_DUMMY_BYTE);
    timeout--;

  }while ((readvalue == SD_DUMMY_BYTE) && timeout);

  /* Right response got */
  return readvalue;
}

/**
  * @brief  Waits a data from the SD card
  * @param  data : Expected data from the SD card
  * @retval BSP_SD_OK or BSP_SD_TIMEOUT
  */
uint8_t SD_WaitData(uint8_t data)
{
  uint16_t timeout = 0xFFFF;
  uint8_t readvalue;

  /* Check if response is got or a timeout is happen */

  do {
    readvalue = SD_IO_WriteByte(SD_DUMMY_BYTE);
    timeout--;
  }while ((readvalue != data) && timeout);

  if (timeout == 0)
  {
    /* After time out */
    return BSP_SD_TIMEOUT;
  }

  /* Right response got */
  return BSP_SD_OK;
}
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
  ******************************************************************************
  * @file    stm32_adafruit_sd.h
  * @author  MCD Application Team
  * @version V2.0.1
  * @date    04-November-2015
  * @brief   This file contains the common defines and functions prototypes for
  *          the stm32_adafruit_sd.c driver.
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32_ADAFRUIT_SD_H
#define __STM32_ADAFRUIT_SD_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include <stdint.h>

/** @addtogroup BSP
  * @{
  */
#define __IO    volatile

enum {
      BSP_SD_OK = 0x00,
      MSD_OK = 0x00,
      BSP_SD_ERROR = 0x01,
      BSP_SD_TIMEOUT
};

typedef struct
{
  uint8_t  Reserved1:2;               /* Reserved */
  uint16_t DeviceSize:12;             /* Device Size */
  uint8_t  MaxRdCurrentVDDMin:3;      /* Max. read current @ VDD min */
  uint8_t  MaxRdCurrentVDDMax:3;      /* Max. read current @ VDD max */
  uint8_t  MaxWrCurrentVDDMin:3;      /* Max. write current @ VDD min */
  uint8_t  MaxWrCurrentVDDMax:3;      /* Max. write current @ VDD max */
  uint8_t  DeviceSizeMul:3;           /* Device size multiplier */
} struct_v1;


typedef struct
{
  uint8_t  Reserved1:6;               /* Reserved */
  uint32_t DeviceSize:22;             /* Device Size */
  uint8_t  Reserved2:1;               /* Reserved */
} struct_v2;

/**
  * @brief  Card Specific Data: CSD Register
  */
typedef struct
{
  /* Header part */
  uint8_t  CSDStruct:2;            /* CSD structure */
  uint8_t  Reserved1:6;            /* Reserved */
  uint8_t  TAAC:8;                 /* Data read access-time 1 */
  uint8_t  NSAC:8;                 /* Data read access-time 2 in CLK cycles */
  uint8_t  MaxBusClkFrec:8;        /* Max. bus clock frequency */
  uint16_t CardComdClasses:12;      /* Card command classes */
  uint8_t  RdBlockLen:4;           /* Max. read data block length */
  uint8_t  PartBlockRead:1;        /* Partial blocks for read allowed */
  uint8_t  WrBlockMisalign:1;      /* Write block misalignment */
  uint8_t  RdBlockMisalign:1;      /* Read block misalignment */
  uint8_t  DSRImpl:1;              /* DSR implemented */

  /* v1 or v2 struct */
  union csd_version {
```

```
    struct_v1 v1;
    struct_v2 v2;
  } version;

  uint8_t  EraseSingleBlockEnable:1;  /* Erase single block enable */
  uint8_t  EraseSectorSize:7;         /* Erase group size multiplier */
  uint8_t  WrProtectGrSize:7;         /* Write protect group size */
  uint8_t  WrProtectGrEnable:1;       /* Write protect group enable */
  uint8_t  Reserved2:2;               /* Reserved */
  uint8_t  WrSpeedFact:3;             /* Write speed factor */
  uint8_t  MaxWrBlockLen:4;           /* Max. write data block length */
  uint8_t  WriteBlockPartial:1;       /* Partial blocks for write allowed */
  uint8_t  Reserved3:5;               /* Reserved */
  uint8_t  FileFormatGrouop:1;        /* File format group */
  uint8_t  CopyFlag:1;                /* Copy flag (OTP) */
  uint8_t  PermWrProtect:1;           /* Permanent write protection */
  uint8_t  TempWrProtect:1;           /* Temporary write protection */
  uint8_t  FileFormat:2;              /* File Format */
  uint8_t  Reserved4:2;               /* Reserved */
  uint8_t  crc:7;                     /* Reserved */
  uint8_t  Reserved5:1;               /* always 1*/

} SD_CSD;

/**
  * @brief  Card Identification Data: CID Register
  */
typedef struct
{
  __IO uint8_t  ManufacturerID;       /* ManufacturerID */
  __IO uint16_t OEM_AppliID;          /* OEM/Application ID */
  __IO uint32_t ProdName1;            /* Product Name part1 */
  __IO uint8_t  ProdName2;            /* Product Name part2*/
  __IO uint8_t  ProdRev;              /* Product Revision */
  __IO uint32_t ProdSN;               /* Product Serial Number */
  __IO uint8_t  Reserved1;            /* Reserved1 */
  __IO uint16_t ManufactDate;         /* Manufacturing Date */
  __IO uint8_t  CID_CRC;              /* CID CRC */
  __IO uint8_t  Reserved2;            /* always 1 */
} SD_CID;

/**
  * @brief SD Card information
  */
typedef struct
{
  SD_CSD Csd;
  SD_CID Cid;
  uint32_t CardCapacity;  /* Card Capacity */
  uint32_t CardBlockSize; /* Card Block Size */
} SD_CardInfo;

/**
  * @}
  */

/** @defgroup STM32_ADAFRUIT_SPI_SD_Exported_Constants
  * @{
  */

/**
  * @brief  Block Size
  */
#define SD_BLOCK_SIZE    0x200

/**
  * @brief  SD detection on its memory slot
  */
#define SD_PRESENT               ((uint8_t)0x01)
#define SD_NOT_PRESENT           ((uint8_t)0x00)

/**
  * @}
```

```
  */

/** @defgroup STM32_ADAFRUIT_SD_Exported_Macro
  * @{
  */

/**
  * @}
  */

/** @defgroup STM32_ADAFRUIT_SD_Exported_Functions
  * @{
  */
uint8_t BSP_SD_Init(void);
uint8_t BSP_SD_ReadBlocks(uint32_t *pData, uint32_t ReadAddr, uint16_t BlockSize, uint32_t
NumberOfBlocks);
uint8_t BSP_SD_WriteBlocks(uint32_t *pData, uint32_t WriteAddr, uint16_t BlockSize, uint32_t
NumberOfBlocks);
uint8_t BSP_SD_Erase(uint32_t StartAddr, uint32_t EndAddr);
uint8_t BSP_SD_GetStatus(void);
uint8_t BSP_SD_GetCardInfo(SD_CardInfo *pCardInfo);

/* Link functions for SD Card peripheral*/
void    SD_IO_Init(void);
void    SD_IO_CSState(uint8_t state);
void    SD_IO_WriteReadData(const uint8_t *DataIn, uint8_t *DataOut, uint16_t DataLength);
uint8_t SD_IO_WriteByte(uint8_t Data);

/* Link function for HAL delay */
void HAL_Delay(__IO uint32_t Delay);

#ifdef __cplusplus
}
#endif

#endif /* __STM32_ADAFRUIT_SD_H */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```c
/**
  ******************************************************************************
  * File Name          : stm32f4xx_nucleo.c
  * Description        : funciones para manejo SD.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_nucleo.h"

/**
  * @brief STM32F4xx NUCLEO BSP Driver version number V1.2.5
  */
#define __STM32F4xx_NUCLEO_BSP_VERSION_MAIN   (0x01) /*!< [31:24] main version */
#define __STM32F4xx_NUCLEO_BSP_VERSION_SUB1   (0x02) /*!< [23:16] sub1 version */
#define __STM32F4xx_NUCLEO_BSP_VERSION_SUB2   (0x05) /*!< [15:8]  sub2 version */
#define __STM32F4xx_NUCLEO_BSP_VERSION_RC     (0x00) /*!< [7:0]  release candidate */
#define __STM32F4xx_NUCLEO_BSP_VERSION         ((__STM32F4xx_NUCLEO_BSP_VERSION_MAIN << 24)\
                                               |(__STM32F4xx_NUCLEO_BSP_VERSION_SUB1 << 16)\
                                               |(__STM32F4xx_NUCLEO_BSP_VERSION_SUB2 << 8 )\
                                               |(__STM32F4xx_NUCLEO_BSP_VERSION_RC))

/**
  * @brief LINK SD Card
  */
#define SD_DUMMY_BYTE            0xFF
#define SD_NO_RESPONSE_EXPECTED  0x80


#ifdef HAL_SPI_MODULE_ENABLED
uint32_t SpixTimeout = NUCLEO_SPIx_TIMEOUT_MAX; /*<! Value of Timeout when SPI communication
fails */
static SPI_HandleTypeDef hnucleo_Spi;
#endif /* HAL_SPI_MODULE_ENABLED */


#ifdef HAL_SPI_MODULE_ENABLED
static void       SPIx_Init(void);
static void       SPIx_Write(uint8_t Value);
static void       SPIx_WriteReadData(const uint8_t *DataIn, uint8_t *DataOut, uint16_t
DataLegnth);
static void       SPIx_Error(void);
static void       SPIx_MspInit(SPI_HandleTypeDef *hspi);

/* SD IO functions */
void              SD_IO_Init(void);
void              SD_IO_CSState(uint8_t state);
void              SD_IO_WriteReadData(const uint8_t *DataIn, uint8_t *DataOut, uint16_t
DataLength);
uint8_t           SD_IO_WriteByte(uint8_t Data);


#endif /* HAL_SPI_MODULE_ENABLED */


uint32_t BSP_GetVersion(void)
{
  return __STM32F4xx_NUCLEO_BSP_VERSION;
}

/*******************************************************************************
                            BUS OPERATIONS
*******************************************************************************/


/******************************* SPI ***************************************/
#ifdef HAL_SPI_MODULE_ENABLED

/**
  * @brief  Initializes SPI MSP.
  */
static void SPIx_MspInit(SPI_HandleTypeDef *hspi)
{
```

```
  GPIO_InitTypeDef  GPIO_InitStruct;

  /*** Configure the GPIOs ***/
  /* Enable GPIO clock */
  NUCLEO_SPIx_SCK_GPIO_CLK_ENABLE();
  NUCLEO_SPIx_MISO_MOSI_GPIO_CLK_ENABLE();

  /* Configure SPI SCK */
  GPIO_InitStruct.Pin = NUCLEO_SPIx_SCK_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull  = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
  GPIO_InitStruct.Alternate = NUCLEO_SPIx_SCK_AF;
  HAL_GPIO_Init(NUCLEO_SPIx_SCK_GPIO_PORT, &GPIO_InitStruct);

  /* Configure SPI MISO and MOSI */
  GPIO_InitStruct.Pin = NUCLEO_SPIx_MOSI_PIN;
  GPIO_InitStruct.Alternate = NUCLEO_SPIx_MISO_MOSI_AF;
  GPIO_InitStruct.Pull  = GPIO_PULLDOWN;
  HAL_GPIO_Init(NUCLEO_SPIx_MISO_MOSI_GPIO_PORT, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = NUCLEO_SPIx_MISO_PIN;
  GPIO_InitStruct.Pull  = GPIO_PULLDOWN;
  HAL_GPIO_Init(NUCLEO_SPIx_MISO_MOSI_GPIO_PORT, &GPIO_InitStruct);

  /*** Configure the SPI peripheral ***/
  /* Enable SPI clock */
  NUCLEO_SPIx_CLK_ENABLE();
}

/**
  * @brief  Initializes SPI HAL.
  */
static void SPIx_Init(void)
{
  if(HAL_SPI_GetState(&hnucleo_Spi) == HAL_SPI_STATE_RESET)
  {
    /* SPI Config */
    hnucleo_Spi.Instance = NUCLEO_SPIx;
      /* SPI baudrate is set to 12,5 MHz maximum (APB1/SPI_BaudRatePrescaler = 100/8 = 12,5
      MHz)
       to verify these constraints:
          - ST7735 LCD SPI interface max baudrate is 15MHz for write and 6.66MHz for read
            Since the provided driver doesn't use read capability from LCD, only constraint
            on write baudrate is considered.
          - SD card SPI interface max baudrate is 25MHz for write/read
          - PCLK2 max frequency is 100 MHz
       */
    hnucleo_Spi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
    hnucleo_Spi.Init.Direction = SPI_DIRECTION_2LINES;
    hnucleo_Spi.Init.CLKPhase = SPI_PHASE_2EDGE;
    hnucleo_Spi.Init.CLKPolarity = SPI_POLARITY_HIGH;
    hnucleo_Spi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLED;
    hnucleo_Spi.Init.CRCPolynomial = 7;
    hnucleo_Spi.Init.DataSize = SPI_DATASIZE_8BIT;
    hnucleo_Spi.Init.FirstBit = SPI_FIRSTBIT_MSB;
    hnucleo_Spi.Init.NSS = SPI_NSS_SOFT;
    hnucleo_Spi.Init.TIMode = SPI_TIMODE_DISABLED;
    hnucleo_Spi.Init.Mode = SPI_MODE_MASTER;

    SPIx_MspInit(&hnucleo_Spi);
    HAL_SPI_Init(&hnucleo_Spi);
  }
}

/**
  * @brief  SPI Write a byte to device
  * @param  DataIn: value to be written
  * @param  DataOut: value to be read
  * @param  DataLegnth: length of data
  */
static void SPIx_WriteReadData(const uint8_t *DataIn, uint8_t *DataOut, uint16_t DataLegnth)
{
```

```
  HAL_StatusTypeDef status = HAL_OK;

  status = HAL_SPI_TransmitReceive(&hnucleo_Spi, (uint8_t*) DataIn, DataOut, DataLegnth,
  SpixTimeout);

  /* Check the communication status */
  if(status != HAL_OK)
  {
    /* Execute user timeout callback */
    SPIx_Error();
  }
}

/**
  * @brief  SPI Write a byte to device.
  * @param  Value: value to be written
  */
static void SPIx_Write(uint8_t Value)
{
  HAL_StatusTypeDef status = HAL_OK;
  uint8_t data;

  status = HAL_SPI_TransmitReceive(&hnucleo_Spi, (uint8_t*) &Value, &data, 1, SpixTimeout);

  /* Check the communication status */
  if(status != HAL_OK)
  {
    /* Execute user timeout callback */
    SPIx_Error();
  }
}

/**
  * @brief  SPI error treatment function.
  */
static void SPIx_Error (void)
{
  /* De-initialize the SPI communication BUS */
  HAL_SPI_DeInit(&hnucleo_Spi);

  /* Re-Initiaize the SPI communication BUS */
  SPIx_Init();
}

/*******************************************************************************
                            LINK OPERATIONS
*******************************************************************************/

/******************************** LINK SD ***************************************/
/**
  * @brief  Initializes the SD Card and put it into StandBy State (Ready for
  *         data transfer).
  */
void SD_IO_Init(void)
{
  GPIO_InitTypeDef  GPIO_InitStruct;
  uint8_t counter;

  /* SD_CS_GPIO Periph clock enable */
  SD_CS_GPIO_CLK_ENABLE();

  /* Configure SD_CS_PIN pin: SD Card CS pin */
  GPIO_InitStruct.Pin = SD_CS_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
  HAL_GPIO_Init(SD_CS_GPIO_PORT, &GPIO_InitStruct);

  /*------------Put SD in SPI mode--------------*/
  /* SD SPI Config */
  SPIx_Init();

  /* SD chip select high */
```

```
  SD_CS_HIGH();

  /* Send dummy byte 0xFF, 10 times with CS high */
  /* Rise CS and MOSI for 80 clocks cycles */
  for (counter = 0; counter <= 9; counter++)
  {
    /* Send dummy byte 0xFF */
    SD_IO_WriteByte(SD_DUMMY_BYTE);
  }
}

/**
  * @brief Set the SD_CS pin.
  * @param val: pin value.
  */
void SD_IO_CSState(uint8_t val)
{
  if(val == 1)
  {
    SD_CS_HIGH();
  }
  else
  {
    SD_CS_LOW();
  }
}

/**
  * @brief Write a byte on the SD.
  * @param  DataIn: value to be written
  * @param  DataOut: value to be read
  * @param  DataLength: length of data
  */
void SD_IO_WriteReadData(const uint8_t *DataIn, uint8_t *DataOut, uint16_t DataLength)
{
  /* Send the byte */
  SPIx_WriteReadData(DataIn, DataOut, DataLength);
}

/**
  * @brief  Writes a byte on the SD.
  * @param  Data: byte to send.
  */
uint8_t SD_IO_WriteByte(uint8_t Data)
{
  uint8_t tmp;
  /* Send the byte */
  SPIx_WriteReadData(&Data,&tmp,1);
  return tmp;
}


#endif /* HAL_SPI_MODULE_ENABLED */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/*****************************************************************************
 *
 *                  FTDIChip AN_xxx FT800 with ARM - Version 1.0
 *
 *****************************************************************************
 * FileName:        FT800.h
 * Purpose:         This header file contains the display list and graphics
 *                  processor commands for the FTDI FT800 graphics controller chip
 * Dependencies:    See INCLUDES section below
 * Company:         Future Technology Devices International Ltd.
 *
 * Software License Agreement
 *
 * This code is provided as an example only and is not guaranteed by FTDI.
 * FTDI accept no responsibility for any issues resulting from its use.
 * The developer of the final application incorporating any parts of this
 * sample project is responsible for ensuring its safe and correct operation
 * and for any conequences resulting from its use.
 *
 * Author   Date                 Rev     Comment
 *****************************************************************************
 * BR           2014-03-01  1.0     Initial version - adapted from AN_275 source
 *****************************************************************************

*/

#ifndef __ft_gpu_h
#define __ft_gpu_h

/*****************************************************************************/
/************************ I N C L U D E S ************************************/
/*****************************************************************************/

/*****************************************************************************/
/********************** D E C L A R A T I O N S ******************************/
/*****************************************************************************/

#define FT_DL_SIZE            (8*1024)  //8KB Display List buffer size
#define FT_CMD_FIFO_SIZE      (4*1024)  //4KB coprocessor Fifo size
#define FT_CMD_SIZE           (4)       //4 byte per coprocessor command of EVE

#define FT800_VERSION         "1.9.0"

// FT800 Chip Commands - use with cmdWrite
#define FT800_ACTIVE                    0x00            // Initializes FT800
#define FT800_STANDBY                   0x41            // Place FT800 in Standby (clk
running)
#define FT800_SLEEP                     0x42            // Place FT800 in Sleep (clk off)
#define FT800_PWRDOWN                   0x50            // Place FT800 in Power Down (core
off)
#define FT800_CLKEXT                    0x44            // Select external clock source
#define FT800_CLK48M                    0x62            // Select 48MHz PLL
#define FT800_CLK36M                    0x61            // Select 36MHz PLL
#define FT800_CORERST                   0x68            // Reset core - all registers default

// FT800 Memory Commands - use with ft800memWritexx and ft800memReadxx
#define MEM_WRITE                           0x80            // FT800 Host Memory Write
#define MEM_READ                            0x00            // FT800 Host Memory Read


// Refer to the FT800 Datasheet

// FT800 Memory Map Addresses
#define RAM_CMD               0x108000UL
#define RAM_DL                0x100000UL
#define RAM_G                 0x000000UL
#define RAM_PAL               0x102000UL
#define RAM_REG               0x102400UL

// FT800 Register Addresses
#define REG_CLOCK             0x102408UL
#define REG_CMD_DL            0x1024ecUL
#define REG_CMD_READ          0x1024e4UL
```

```
#define REG_CMD_WRITE         0x1024e8UL
#define REG_CPURESET          0x10241cUL
#define REG_CSPREAD           0x102464UL
#define REG_DITHER            0x10245cUL
#define REG_DLSWAP            0x102450UL
#define REG_FRAMES            0x102404UL
#define REG_FREQUENCY         0x10240cUL
#define REG_GPIO              0x102490UL
#define REG_GPIO_DIR          0x10248cUL
#define REG_HCYCLE            0x102428UL
#define REG_HOFFSET           0x10242cUL
#define REG_HSIZE             0x102430UL
#define REG_HSYNC0            0x102434UL
#define REG_HSYNC1            0x102438UL
#define REG_ID                0x102400UL
#define REG_INT_EN            0x10249cUL
#define REG_INT_FLAGS         0x102498UL
#define REG_INT_MASK          0x1024a0UL
#define REG_MACRO_0           0x1024c8UL
#define REG_MACRO_1           0x1024ccUL
#define REG_OUTBITS           0x102458UL
#define REG_PCLK              0x10246cUL
#define REG_PCLK_POL          0x102468UL
#define REG_PLAY              0x102488UL
#define REG_PLAYBACK_FORMAT   0x1024b4UL
#define REG_PLAYBACK_FREQ     0x1024b0UL
#define REG_PLAYBACK_LENGTH   0x1024a8UL
#define REG_PLAYBACK_LOOP     0x1024b8UL
#define REG_PLAYBACK_PLAY     0x1024bcUL
#define REG_PLAYBACK_READPTR  0x1024acUL
#define REG_PLAYBACK_START    0x1024a4UL
#define REG_PWM_DUTY          0x1024c4UL
#define REG_PWM_HZ            0x1024c0UL
#define REG_RENDERMODE        0x102410UL
#define REG_ROTATE            0x102454UL
#define REG_SNAPSHOT          0x102418UL
#define REG_SNAPY             0x102414UL
#define REG_SOUND             0x102484UL
#define REG_SWIZZLE           0x102460UL
#define REG_TAG               0x102478UL
#define REG_TAG_X             0x102470UL
#define REG_TAG_Y             0x102474UL
#define REG_TAP_CRC           0x102420UL
#define REG_TAP_MASK          0x102424UL
#define REG_TOUCH_ADC_MODE    0x1024f4UL
#define REG_TOUCH_CHARGE      0x1024f8UL
#define REG_TOUCH_DIRECT_XY   0x102574UL
#define REG_TOUCH_DIRECT_Z1Z2 0x102578UL
#define REG_TOUCH_MODE        0x1024f0UL
#define REG_TOUCH_OVERSAMPLE  0x102500UL
#define REG_TOUCH_RAW_XY      0x102508UL
#define REG_TOUCH_RZ          0x10250cUL
#define REG_TOUCH_RZTHRESH    0x102504UL
#define REG_TOUCH_SCREEN_XY   0x102510UL
#define REG_TOUCH_SETTLE      0x1024fcUL
#define REG_TOUCH_TAG         0x102518UL
#define REG_TOUCH_TAG_XY      0x102514UL
#define REG_TOUCH_TRANSFORM_A 0x10251cUL
#define REG_TOUCH_TRANSFORM_B 0x102520UL
#define REG_TOUCH_TRANSFORM_C 0x102524UL
#define REG_TOUCH_TRANSFORM_D 0x102528UL
#define REG_TOUCH_TRANSFORM_E 0x10252cUL
#define REG_TOUCH_TRANSFORM_F 0x102530UL
#define REG_TRACKER           0x109000UL
#define REG_VCYCLE            0x10243cUL
#define REG_VOFFSET           0x102440UL
#define REG_VOL_PB            0x10247cUL
#define REG_VOL_SOUND         0x102480UL
#define REG_VSIZE             0x102444UL
#define REG_VSYNC0            0x102448UL
#define REG_VSYNC1            0x10244cUL

// Graphics Engine Commands
```

```
// Refer to the FT800 Programmers Guide
#define CMDBUF_SIZE             4096UL
#define CMD_APPEND              0xffffff1eUL
#define CMD_BGCOLOR             0xffffff09UL
#define CMD_BUTTON              0xffffff0dUL
#define CMD_CALIBRATE           0xffffff15UL
#define CMD_CLOCK               0xffffff14UL
#define CMD_COLDSTART           0xffffff32UL
#define CMD_DIAL                0xffffff2dUL
#define CMD_DLSTART             0xffffff00UL
#define CMD_FGCOLOR             0xffffff0aUL
#define CMD_GAUGE               0xffffff13UL
#define CMD_GETMATRIX           0xffffff33UL
#define CMD_GETPTR              0xffffff23UL
#define CMD_GETPROPS            0xffffff25UL
#define CMD_GRADCOLOR           0xffffff34UL
#define CMD_GRADIENT            0xffffff0bUL
#define CMD_INFLATE             0xffffff22UL
#define CMD_INTERRUPT           0xffffff02UL
#define CMD_KEYS                0xffffff0eUL
#define CMD_LOADIDENTITY        0xffffff26UL
#define CMD_LOADIMAGE           0xffffff24UL
#define CMD_LOGO                0xffffff31UL
#define CMD_MEMCPY              0xffffff1dUL
#define CMD_MEMCRC              0xffffff18UL
#define CMD_MEMSET              0xffffff1bUL
#define CMD_MEMWRITE            0xffffff1aUL
#define CMD_MEMZERO             0xffffff1cUL
#define CMD_NUMBER              0xffffff2eUL
#define CMD_PROGRESS            0xffffff0fUL
#define CMD_REGREAD             0xffffff19UL
#define CMD_ROTATE              0xffffff29UL
#define CMD_SCALE               0xffffff28UL
#define CMD_SCREENSAVER         0xffffff2fUL
#define CMD_SCROLLBAR           0xffffff11UL
#define CMD_SETFONT             0xffffff2bUL
#define CMD_SETMATRIX           0xffffff2aUL
#define CMD_SKETCH              0xffffff30UL
#define CMD_SLIDER              0xffffff10UL
#define CMD_SNAPSHOT            0xffffff1fUL
#define CMD_SPINNER             0xffffff16UL
#define CMD_STOP                0xffffff17UL
#define CMD_SWAP                0xffffff01UL
#define CMD_TEXT                0xffffff0cUL
#define CMD_TOGGLE              0xffffff12UL
#define CMD_TRACK               0xffffff2cUL
#define CMD_TRANSLATE           0xffffff27UL

// Display list commands to be embedded in Graphics Processor
#define DL_ALPHA_FUNC           0x09000000UL // requires OR'd arguments
#define DL_BITMAP_HANDLE        0x05000000UL // requires OR'd arguments
#define DL_BITMAP_LAYOUT        0x07000000UL // requires OR'd arguments
#define DL_BITMAP_SIZE          0x08000000UL // requires OR'd arguments
#define DL_BITMAP_SOURCE        0x01000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_A       0x15000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_B       0x16000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_C       0x17000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_D       0x18000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_E       0x19000000UL // requires OR'd arguments
#define DL_BITMAP_TFORM_F       0x1A000000UL // requires OR'd arguments
#define DL_BLEND_FUNC           0x0B000000UL // requires OR'd arguments
#define DL_BEGIN                0x1F000000UL // requires OR'd arguments
#define DL_CALL                 0x1D000000UL // requires OR'd arguments
#define DL_CLEAR                0x26000000UL // requires OR'd arguments
#define DL_CELL                 0x06000000UL // requires OR'd arguments
#define DL_CLEAR_RGB            0x02000000UL // requires OR'd arguments
#define DL_CLEAR_STENCIL        0x11000000UL // requires OR'd arguments
#define DL_CLEAR_TAG            0x12000000UL // requires OR'd arguments
#define DL_COLOR_A              0x0F000000UL // requires OR'd arguments
#define DL_COLOR_MASK           0x20000000UL // requires OR'd arguments
#define DL_COLOR_RGB            0x04000000UL // requires OR'd arguments
#define DL_DISPLAY              0x00000000UL
#define DL_END                  0x21000000UL
```

```
#define DL_JUMP                 0x1E000000UL // requires OR'd arguments
#define DL_LINE_WIDTH           0x0E000000UL // requires OR'd arguments
#define DL_MACRO                0x25000000UL // requires OR'd arguments
#define DL_POINT_SIZE           0x0D000000UL // requires OR'd arguments
#define DL_RESTORE_CONTEXT      0x23000000UL
#define DL_RETURN               0x24000000UL
#define DL_SAVE_CONTEXT         0x22000000UL
#define DL_SCISSOR_SIZE         0x1C000000UL // requires OR'd arguments
#define DL_SCISSOR_XY           0x1B000000UL // requires OR'd arguments
#define DL_STENCIL_FUNC         0x0A000000UL // requires OR'd arguments
#define DL_STENCIL_MASK         0x13000000UL // requires OR'd arguments
#define DL_STENCIL_OP           0x0C000000UL // requires OR'd arguments
#define DL_TAG                  0x03000000UL // requires OR'd arguments
#define DL_TAG_MASK             0x14000000UL // requires OR'd arguments
#define DL_VERTEX2F             0x40000000UL // requires OR'd arguments
#define DL_VERTEX2II            0x02000000UL // requires OR'd arguments

// Command and register value options
#define CLR_COL                 0x4
#define CLR_STN                 0x2
#define CLR_TAG                 0x1
#define DECR                    4UL
#define DECR_WRAP               7UL
#define DLSWAP_DONE             0UL
#define DLSWAP_FRAME            2UL
#define DLSWAP_LINE             1UL
#define DST_ALPHA               3UL
#define EDGE_STRIP_A            7UL
#define EDGE_STRIP_B            8UL
#define EDGE_STRIP_L            6UL
#define EDGE_STRIP_R            5UL
#define EQUAL                   5UL
#define GEQUAL                  4UL
#define GREATER                 3UL
#define INCR                    3UL
#define INCR_WRAP               6UL
#define INT_CMDEMPTY            32UL
#define INT_CMDFLAG             64UL
#define INT_CONVCOMPLETE        128UL
#define INT_PLAYBACK            16UL
#define INT_SOUND               8UL
#define INT_SWAP                1UL
#define INT_TAG                 4UL
#define INT_TOUCH               2UL
#define INVERT                  5UL
#define KEEP                    1UL
#define L1                      1UL
#define L4                      2UL
#define L8                      3UL
#define LEQUAL                  2UL
#define LESS                    1UL
#define LINEAR_SAMPLES          0UL
#define LINES                   3UL
#define LINE_STRIP              4UL
#define NEAREST                 0UL
#define NEVER                   0UL
#define NOTEQUAL                6UL
#define ONE                     1UL
#define ONE_MINUS_DST_ALPHA     5UL
#define ONE_MINUS_SRC_ALPHA     4UL
#define OPT_CENTER              1536UL    // 0x6000
#define OPT_CENTERX             512UL     // 0x0200
#define OPT_CENTERY             1024UL    // 0x0400
#define OPT_FLAT                256UL     // 0x0100
#define OPT_MONO                1UL
#define OPT_NOBACK              4096UL    // 0x1000
#define OPT_NODL                2UL
#define OPT_NOHANDS             49152UL   // 0xC168
#define OPT_NOHM                16384UL   // 0x4000
#define OPT_NOPOINTER           16384UL   // 0x4000
#define OPT_NOSECS              32768UL   // 0x8000
#define OPT_NOTICKS             8192UL    // 0x2000
#define OPT_RIGHTX              2048UL    // 0x0800
```

```
#define OPT_SIGNED            256UL     // 0x0100
#define PALETTED              8UL
#define PLAYCOLOR             0x00a0a080
#define FTPOINTS              2UL       // "POINTS" is a reserved word
#define RECTS                 9UL
#define REPEAT                1UL
#define REPLACE               2UL
#define RGB332                4UL
#define RGB565                7UL
#define SRC_ALPHA             2UL
#define TEXT8X8               9UL
#define TEXTVGA               10UL
#define TOUCHMODE_CONTINUOUS  3UL
#define TOUCHMODE_FRAME       2UL
#define TOUCHMODE_OFF         0UL
#define TOUCHMODE_ONESHOT     1UL
#define ULAW_SAMPLES          1UL
#define ZERO                  0UL

// Useful Macros
#define RGB(r, g, b)          ((((r) << 16) | (g) << 8) | (b))
#define SQ(v)                 ((v) * (v))
#define MIN(x,y)              ((x) > (y) ? (y) : (x))
#define MAX(x,y)              ((x) > (y) ? (x) : (y))
#define NOTE(n, sharp)        (((n) - 'C') + ((sharp) * 128))
#define F16(s)                (((s) * 65536))
#define INVALID_TOUCH_XY      0x8000
#define ABS(x)                ((x) > (0) ? (x) : (-x))

#define VERTEX2F(x,y) ((1UL<<30)|(((x)&32767UL)<<15)|(((y)&32767UL)<<0))
#define VERTEX2II(x,y,handle,cell)
((2UL<<30)|(((x)&511UL)<<21)|(((y)&511UL)<<12)|(((handle)&31UL)<<7)|(((cell)&127UL)<<0))
#define BITMAP_SOURCE(addr) ((1UL<<24)|(((addr)&1048575UL)<<0))
#define CLEAR_COLOR_RGB(red,green,blue)
((2UL<<24)|(((red)&255UL)<<16)|(((green)&255UL)<<8)|(((blue)&255UL)<<0))
#define TAG(s) ((3UL<<24)|(((s)&255UL)<<0))
#define COLOR_RGB(red,green,blue)
((4UL<<24)|(((red)&255UL)<<16)|(((green)&255UL)<<8)|(((blue)&255UL)<<0))
#define BITMAP_HANDLE(handle) ((5UL<<24)|(((handle)&31UL)<<0))
#define CELL(cell) ((6UL<<24)|(((cell)&127UL)<<0))
#define BITMAP_LAYOUT(format,linestride,height)
((7UL<<24)|(((format)&31UL)<<19)|(((linestride)&1023UL)<<9)|(((height)&511UL)<<0))
#define BITMAP_SIZE(filter,wrapx,wrapy,width,height)
((8UL<<24)|(((filter)&1UL)<<20)|(((wrapx)&1UL)<<19)|(((wrapy)&1UL)<<18)|(((width)&511UL)<<9)|(
((height)&511UL)<<0))
#define ALPHA_FUNC(func,ref) ((9UL<<24)|(((func)&7UL)<<8)|(((ref)&255UL)<<0))
#define STENCIL_FUNC(func,ref,mask)
((10UL<<24)|(((func)&7UL)<<16)|(((ref)&255UL)<<8)|(((mask)&255UL)<<0))
#define BLEND_FUNC(src,dst) ((11UL<<24)|(((src)&7UL)<<3)|(((dst)&7UL)<<0))
#define STENCIL_OP(sfail,spass) ((12UL<<24)|(((sfail)&7UL)<<3)|(((spass)&7UL)<<0))
#define POINT_SIZE(size) ((13UL<<24)|(((size)&8191UL)<<0))
#define LINE_WIDTH(width) ((14UL<<24)|(((width)&4095UL)<<0))
#define CLEAR_COLOR_A(alpha) ((15UL<<24)|(((alpha)&255UL)<<0))
#define COLOR_A(alpha) ((16UL<<24)|(((alpha)&255UL)<<0))
#define CLEAR_STENCIL(s) ((17UL<<24)|(((s)&255UL)<<0))
#define CLEAR_TAG(s) ((18UL<<24)|(((s)&255UL)<<0))
#define STENCIL_MASK(mask) ((19UL<<24)|(((mask)&255UL)<<0))
#define TAG_MASK(mask) ((20UL<<24)|(((mask)&1UL)<<0))
#define BITMAP_TRANSFORM_A(a) ((21UL<<24)|(((a)&131071UL)<<0))
#define BITMAP_TRANSFORM_B(b) ((22UL<<24)|(((b)&131071UL)<<0))
#define BITMAP_TRANSFORM_C(c) ((23UL<<24)|(((c)&16777215UL)<<0))
#define BITMAP_TRANSFORM_D(d) ((24UL<<24)|(((d)&131071UL)<<0))
#define BITMAP_TRANSFORM_E(e) ((25UL<<24)|(((e)&131071UL)<<0))
#define BITMAP_TRANSFORM_F(f) ((26UL<<24)|(((f)&16777215UL)<<0))
#define SCISSOR_XY(x,y) ((27UL<<24)|(((x)&511UL)<<9)|(((y)&511UL)<<0))
#define SCISSOR_SIZE(width,height) ((28UL<<24)|(((width)&1023UL)<<10)|(((height)&1023UL)<<0))
#define CALL(dest) ((29UL<<24)|(((dest)&65535UL)<<0))
#define JUMP(dest) ((30UL<<24)|(((dest)&65535UL)<<0))
#define BEGIN(prim) ((31UL<<24)|(((prim)&15UL)<<0))
#define COLOR_MASK(r,g,b,a)
((32UL<<24)|(((r)&1UL)<<3)|(((g)&1UL)<<2)|(((b)&1UL)<<1)|(((a)&1UL)<<0))
#define CLEAR(c,s,t) ((38UL<<24)|(((c)&1UL)<<2)|(((s)&1UL)<<1)|(((t)&1UL)<<0))
#define END() ((33UL<<24))
```

```
#define SAVE_CONTEXT() ((34UL<<24))
#define RESTORE_CONTEXT() ((35UL<<24))
#define RETURN() ((36UL<<24))
#define MACRO(m) ((37UL<<24)|(((m)&1UL)<<0))
#define DISPLAY() ((0UL<<24))

#define POINTS                2UL
#define BITMAPS               1UL

#endif  //__ft_gpu_h
```