UNIVERSIDAD NACIONAL de MAR DEL PLATA

Departamento Ingeniería Eléctrica

Diseño y construcción de un sistema de monitoreo remoto de instalaciones automáticas

Autor: Yerro Avincetto, Nahuel

Director del Proyecto: Dr. Ing. Strack, Jorge Luis

Co-Director del Proyecto: Ing. Branda, Julio

Evaluadores:

Ing. Murcia, Guillermo

Ing. Belliski, Gustavo

Trabajo Final de Grado para acceder al título de Ingeniero Eléctrico

Mar del Plata, 29 del mes de Noviembre, 2019

RINFI

REPOSITORIO INSTITUCIONAL

INTEMA | FACULTAD DE INGENIERÍA | UNMDP

RINFI se desarrolla en forma conjunta entre el INTEMA y la Biblioteca de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

Tiene como objetivo recopilar, organizar, gestionar, difundir y preservar documentos digitales en Ingeniería, Ciencia y Tecnología de Materiales y Ciencias Afines.

A través del Acceso Abierto, se pretende aumentar la visibilidad y el impacto de los resultados de la investigación, asumiendo las políticas y cumpliendo con los protocolos y estándares internacionales para la interoperabilidad entre repositorios


Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.

Resumen

Este trabajo tiene como objetivo solucionar una problemática puntual en las instalaciones del horno crematorio del Cementerio Parque de la ciudad. El mismo requiere de un sistema de monitoreo que se encargue de realizar notificaciones de eventos importantes como el apagado del horno debido a una falla o a la falta de suministro de energía electrica o gas. Para la solución de la misma se plantea la instalación de un sistema de monitoreo remoto, el cual permitirá reaccionar de manera más veloz ante la ocurrencia de eventualidades y reducir al mínimo cualquier problemática que pueda ocurrir. Para ello se analizan las posibles alternativas del mercado y las aptitudes que tienen las mismas para solucionar la problemática en función de las condiciones establecidas para la solución del problema. Este análisis lleva a la opción del uso de una tarjeta de desarrollo de hardware y software de código abierto basada en un microcontrolador, conocida como Arduino.

Se procede a la investigación de componentes y opciones para generar un dispositivo que permita satisfacer los objetivos planteados. Esto da lugar al uso de comunicación remota mediante la tecnología GSM y comunicación industrial con el PLC instalado mediante Modbus. El uso de una tarjeta SD permite el almacenamiento de datos adquiridos de funcionamiento y la posibilidad del uso de archivos de configuración para el dispositivo.

El uso de las tecnologías mencionadas permiten la creación de un dispositivo que permite el monitoreo remoto de las instalaciones. El mismo se caracteriza por su bajo costo y por la capacidad de adaptarse ante los cambios que se puedan realizar en las instalaciones del lugar.

INDICE

INDICE DE TABLAS

INDICE DE FIGURAS

CAPÍTULO N°1: INTRODUCCION

1.1 Objetivo del capítulo

Se explicarán las bases y objetivos del proyecto, para ello se indicará información de las condiciones de las instalaciones y de la problemática a resolver.

1.2 Problemática

En el crematorio del Cementerio Parque se presenta la necesidad de diseñar un sistema de monitoreo, con la finalidad de verificar remotamente el correcto funcionamiento del mismo. Esto es debido a que ante la existencia de algún desperfecto que genere el apagado del mismo para volver a encenderlo y ponerlo a punto de funcionamiento se puede tardar hasta doce horas. Considerando la naturaleza del proceso realizado por el mismo esto es un gran problema, entre otras cosas porque los cambios bruscos de temperatura (enfriamiento excesivo en pocas horas) podrían generar serios deterioros estructurales en las paredes del horno.

Figura 1.2.1 Puertas del horno

1.3 Instalaciones

En el lugar de trabajo se dispone de un horno pirolítico, es decir que permite la descomposición de materia orgánica a altas temperaturas (700°C - 1000°C). El mismo posee dos cámaras, realizando el quemado en dos etapas. La primera de ellas posee dos quemadores y se utiliza para hacer la ignición. Esta etapa se lleva a cabo con una baja presencia de oxigeno generando gases ricos en combustible debido a la materia orgánica quemada. En la segunda hay únicamente un quemador y se realiza la postcombustión. La última etapa se genera en un entorno enriquecido en oxígeno quemando los gases para que al liberarlos a la atmósfera estos no resulten dañinos para el medio ambiente.

Cada uno de los quemadores posee un sistema independiente de control siendo análogos entre sí. Permiten la realización de maniobras de encendido y apagado, así también como la verificación de presencia de gas y energía, temperatura dentro de los parámetros de seguridad, etc. En caso de alguna falla el quemador es apagado, si la naturaleza de la misma es debido a temperatura elevada, el quemador se reiniciara luego de un tiempo de haberse apagado.

1.4 Régimen de funcionamiento

Durante horarios laborales el horno se mantiene a la temperatura de trabajo y el quemador de la cámara secundaria es encendido sólo durante el proceso de cremación. Los quemadores fuera de los horarios de trabajo de la cámara principal se mantienen encendidos funcionado a bajas temperaturas y el quemador de la cámara secundaria es apagado. Esto se debe a varios motivos. Los tiempos necesarios para encender y alcanzar la temperatura de funcionamiento son prolongados, el consumo de combustible resulta menor cuando funciona permanentemente que cuando funciona de forma intermitente y se genera un menor desgaste del mismo debido a la reducción de dilataciones y contracciones causadas por variaciones de temperatura.

1.5 Objetivos

Se pide la instalación de un sistema que cumpla con los siguientes objetivos

- Monitoreo remoto. (Informe del estado de las instalaciones)
- Reinstalación y reconfiguración de un Controlador lógico programable (PLC, en inglés Programmable Logic Controller) disponible en el lugar con la finalidad de centralizar la información.
- Flexibilidad, sistema que permita modificaciones futuras (se planea instalaciones de nuevos sensores).
- Robustez y confiabilidad, el sistema debe mantener su funcionamiento ante un corte del suministro de energía eléctrica.
- Bajo costo.

1.6 Objetivos adicionales

Funcionalidades adicionales deseables para el sistema a instalar

- Sistema de alarma
- Registro de datos
- Capacidad de operaciones remotas de dispositivos ajenos al horno (luces, chicharras, etc.)

1.7 Propuesta

A fin de resolver la problemática presentada, se propone en este proyecto de grado, diseñar un sistema de monitoreo versátil, de bajo costo, que cumpla con los requerimientos presentados. Para esta tarea se analizará cuáles son las alternativas comerciales existentes a fin de seleccionar el hardware y software más apropiados para cumplir con los objetivos planteados.

A continuación, en el capítulo 2 se presenta un análisis de las posibilidades tecnológicas asequibles en el mercado. Luego, en el capítulo 3 se presentarán conceptos asociados a la tecnología adoptada a tener en cuenta a lo largo del proyecto. En el capítulo 4 se describe el PLC utilizado y su programación. En forma similar, en el capítulo 5 de describe la placa Arduino y sus periféricos, utilizados para el monitoreo del PLC, así como la programación de los mismos. Luego, en el capítulo 6 se presenta el manual de usuario del proyecto diseñado. Para finalizar, en el capítulo 7 se aborda un análisis económico, comparando el costo del sistema propuesto frente al de otras alternativas comerciales. Por último, en el capítulo 8 se realizan las principales discusiones y conclusiones finales del proyecto.

CAPÍTULO N°2: POSIBILIDADES TECNOLOGICAS

2.1 Objetivo del capítulo

En este capítulo se investigarán diferentes métodos actuales del mercado que puedan llegar a cumplir las necesidades del proyecto indicando sus funcionalidades. Se detallarán posibilidades para la creación de un dispositivo y en base a ello se optará por el más recomendado.

2.2 Alternativas del mercado

Hoy en día existen múltiples soluciones a la hora de monitorear un sistema a distancia. Cada una de estas opciones determinará los dispositivos adicionales a instalar y el medio por el cual se realizará la comunicación remota.

2.2.1 Medio de comunicación

Para que un mensaje sea entregado es necesario un medio por el cual transmitirlo. Para montar un servicio de monitoreo es necesario un medio de comunicación. Hoy en día existen varias posibilidades, dentro de las más comunes se pueden encontrar dos:

- Ethernet
- Radiofrecuencia

2.2.1.1 Ethernet

Se plantea el uso de la red internet por conexión directa Ethernet. Este medio permite el acceso desde una amplia gama de dispositivos con acceso a internet, permitiendo así, controlar un sistema desde un ordenador o un Smartphone. Este método requerirá un software especializado para monitorear el sistema instalado y en adición se recomienda el uso de una Red Privada Virtual (VPN, en inglés Virtual Private Network) para proteger la información enviada de posibles filtraciones o ataques.

2.2.1.2 Radiofrecuencia

Se trata de la comunicación inalámbrica entre dispositivos utilizando señales de frecuencia de radio. Dentro de las tecnologías de comunicación más habituales se encuentran GSM, GPRS y Zigbee

- GPRS: Implica la conexión a internet mediante la red del servicio general de paquetes vía radio (GPRS, en inglés General Packet Radio Service), siendo esta tecnología la utilizada por los teléfonos móviles.
- GSM: El Sistema global para las comunicaciones (GSM, en inglés Global Sistema for Mobile comunicación) es la tecnología que permite el intercambio de información a través del servicio de mensajes de texto corto (SMS, en inglés Short Message Service).
- Zigbee: Se trata de un conjunto de protocolos de comunicación inalámbrica para su utilización con radiodiodifusión digital de bajo consumo.

Dentro de los medios planteados se descartan Ethernet y Zigbee. La primera de las opciones es debido a que las instalaciones no disponen acceso a la red mediante cableado. La posibilidad del uso de la tecnología Zigbee se descarta debido a la necesidad de un dispositivo de recepción y de las limitaciones de distancias.

La tecnología GSM requerirá que el usuario disponga un teléfono celular para realizar la comunicación y puede que se necesite una interface la cual permita una mejor interpretación de la información, así también en caso de que utilice con la finalidad de control puede que sea requerida la capacidad de codificación del mensaje para ser interpretado por el dispositivo. La tecnología GPRS simplemente requerirá que el usuario utilice un dispositivo con acceso a internet.

2.2.2 Dispositivos adicionales a instalar

Según el sistema instalado se requerirá la instalación de dispositivos adicionales para permitir la comunicación remota. Estos pueden ser módulos para el PLC o dispositivos independientes. Los módulos deben permitir la conexión a través de algún medio (GSM/GPRS, Ethernet, etc.), mientras que los dispositivos independientes deberán contar con la posibilidad de comunicarse con el PLC y la posibilidad de comunicación remota.

Actualmente se disponen sistemas individuales de control, por lo cual se estaría utilizando el PLC disponible para reunir la información en un único punto y al mismo tiempo poder obtener un control centralizado.

El PLC a utilizar es S7-200 CPU 216-2 (Ver Capítulo 4). El mismo fabricante dispone de diferentes posibilidades que permiten realizar tales tareas, dentro de ellos se encuentran el uso de un módem como MD720-3 o un módulo para el PLC como CP243-1 IT. El módem mencionado podría permitir una comunicación utilizando la red GSM/GPRS, permitiendo la transferencia de datos por mensaje de texto y/o internet. Por otro lado, el módulo mencionado permite la conexión directa del PLC a internet, posibilitando así el envío de correos electrónicos y el uso de páginas web HTML (Lenguaje de Marcas de Hipertexto, en inglés HyperText Markup Language). Del mismo modo en el mercado se pueden encontrar dispositivos con funcionalidades similares de otras marcas.

El uso de un módem o módulo posee la ventaja de un costo relativamente bajo y disponibilidad de varios medios de comunicación. Aunque existe la posibilidad de encontrar problemas de compatibilidad o falta de controladores al utilizarlos con el PLC disponible.

Fig. 2.2.2.1 MD720-3

Fig. 2.2.2.2 CP243-1 IT

Existen PLC's que disponen de funciones de comunicación remota que podrían utilizarse como maestros o reemplazar completamente el uso del X-Messenger, que consiste en un PLC con funciones de monitoreo y control de forma remota utilizando GSM como medio. Permite protocolo Modbus, registro de datos, en memoria SD, configuración mediante SMS, conexión Ethernet, entre otras opciones.

Esta opción permite una reducción en los componentes a instalar (solo el PLC) y permite una gran variedad de posibilidades dependiendo del dispositivo a utilizar. En contraparte, el costo de este método es moderado.

Fig. 2.2.2.3 X-Messenger

Utilizando un ordenador con una conexión directa al PLC instalado se podría generar un control y monitoreo remoto con una conexión a internet. En adición se requeriría la búsqueda de software que permita la comunicación al exterior.

El uso de un ordenador permite mayor flexibilidad y capacidad de personalización. Pero, el mismo requiere cierta complejidad a la hora de montar el servidor y un costo moderado.

Finalmente, existe otra opción, la creación de un dispositivo que se adecúe a las necesidades utilizando una tarjeta de desarrollo. Esta opción puede requerir mayor complejidad a la hora de programar el dispositivo, pero permite un mayor grado de flexibilidad en cuanto a las funciones y usos del dispositivo, como así también con un costo reducido. En el mercado actual existen diversas opciones, pero nos basaremos en los dos más populares (Raspberry pi y Arduino).

2.3 Raspberry pi Vs Arduino

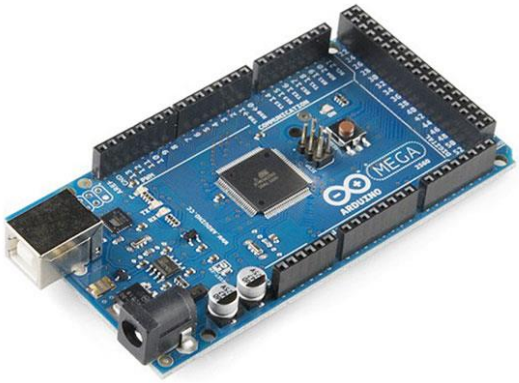


Fig. 2.3.1 Arduino MEGA 2560

Fig. 2.3.2 Raspberry pi 1 B

Si bien ambas opciones a simple vista pueden parecer similares y permiten resolver mismos problemas, son muy diferentes uno de otro. Arduino es un micro controlador programable en su propio lenguaje similar a C, mientras que Raspberry pi es una pequeña computadora funcional la cual permite la ejecución de sistemas operativos como Linux, Windows 10, etc.

En cuestiones de tamaño ambas poseen dimensiones similares, su costo es reducido en ambos casos aunque la Raspberry pi es ligeramente más costosa. La principal diferencia es lo que se encuentra en su interior.

Debido a que existen diversos modelos con capacidades muy diferentes tanto para Arduino como para Raspberry pi. Se darán datos de dos modelos particulares, la Arduino MEGA 2560 y la Raspberry pi modelo B. Los modelos fueron elegidos simplemente para poder cuantificar y tomar noción de sus diferencias.

En la siguiente tabla se pueden apreciar algunas de sus características

Dispositivo	Arduino MEGA 2560	Raspberry Pi 1 B
Memoria	0.008 Mb	512 Mb
Velocidad de reloj	16 MHz	700 MHz
Multitarea	No	Sí
Memoria Flash	256 Kb	Tarjeta SD
Sistema operativo	Ninguno	Linux
Entorno de desarrollo integrado (IDE)	Arduino	Scratch, IDLE, cualquiera con soporte Linux
Cantidad de pines digitales programables (GPIO)	54	40

Tabla 2.3.3 Comparación Arduino MEGA 2560 con Raspberry Pi 1 B

De esto se puede notar que Raspberry pi posee las capacidades de una pequeña computadora, pudiendo de ese modo decirse que es superior. Sin embargo, Arduino presenta mayor robustez y simpleza en programación, haciendo que sea más práctica para aplicaciones donde no se requiere gran potencia. Para aclarar lo dicho, si se desea realizar un dispositivo que encienda un LED de forma intermitente mediante una salida digital en Arduino bastaran menos de diez líneas de código cargadas desde un simple software, mientras en Raspberry pi será necesario instalar un sistema operativo, cargar librerías antes de siquiera comenzar a diseñar el programa. Por lo cual se puede decir que si bien Raspberry Pi es más poderosa, su complejidad puede resultar contraproducente en ciertas tareas.

En ambos casos existen grandes comunidades que giran en torno a ellas y en general son utilizadas para la enseñanza, por lo cual se encuentra fácilmente material sobre las mismas. Esto fue gracias a que ambas placas de desarrollo cuentan con hardware y software completamente abiertos a toda la comunidad y a sus bajos costos.

Entonces, si se debe elegir entre ambas se debe tener en cuenta la aplicación. Raspberry Pi tiene utilidad cuando se pueda requerir el uso de una computadora. Permitiendo el uso de software complejos facilitando el almacenamiento y procesamiento de grandes cantidades de datos, ejecución de multiprocesos o tareas, interacción con diversos dispositivos a través de múltiples tipos de conectividad incluidos en la misma placa (bluetoth, wifi, Ethernet, USB, puerto serie, canales de audio, etc.) y ejecución de interfaces gráficas de alta complejidad que permitan la interacción con el usuario. Por otra parte Arduino es mejor opción cuando se requiera interactuar con otro dispositivo a una baja tasa de transferencia de datos pero con una muy baja latencia en la comunicación, es decir, aquellas aplicaciones donde se requiera un micro controlador exclusivamente dedicado a atender la comunicación con otro dispositivo, capaz de ejecutar un programa de comandos secuenciales, ejecutar rutinas de servicios de interrupciones tanto internas como externas, lectura y escritura de canales analógicos y digitales, etc.

2.4 Conclusiones del capítulo

Teniendo en cuenta lo expresado y considerando las limitaciones establecidas para solucionar el problema, se ha considerado que el uso de una tarjeta de desarrollo es la opción más conveniente. De los métodos presentados es el más económico y flexible. Esto permite la creación de un dispositivo que se adecue perfectamente a las necesites del problema presentado. De las tarjetas de desarrollo presentadas se ha optado por Arduino, dado que se no se estima requerir una gran potencia y ni complejidad. Como medio de comunicación se utilizará la tecnología GSM debido a que el volumen de datos a transmitir es reducido y presenta menor complejidad que la tecnología GPRS.

CAPÍTULO N°3: TECNOLOGIAS INVOLUCRADAS

3.1 Objetivos del capítulo

En el capítulo se presentarán diferentes nociones a tener en cuenta a lo largo del proyecto, permitiendo así mayor entendimiento de términos aplicados, funcionamiento y decisiones tomadas.

3.2 Arduino

Arduino es una plataforma de código abierto basada en hardware y software libre y fácil de usar, por esos motivos es utilizada principalmente por estudiantes. Las placas Arduino son capaces de leer señales de entrada (luz en un sensor, botón, un mensaje de Twitter, etc.) Y transformarlo en una señal de salida (activando un motor, prendiendo un LED, publicando un mensaje online, etc.). Se puede lograr que la placa realice dichas tareas enviando una serie de instrucciones al microcontrolador de la placa. Para ello se utiliza el lenguaje de programación Arduino (Basado en Wiring), y el software de Arduino (IDE, entorno de desarrollo integrado, en inglés Integrated Development Environment), basado en Processing.

A lo largo de los años Arduino ha funcionado como cerebro de cientos de proyectos, desde objetos simples hasta instrumentos avanzados. Debido a sus posibilidades se ha generado una amplia comunidad de creadores en torno a la plataforma de código abierto, cuyas contribuciones han dado una gran cantidad de conocimiento fácilmente accesible el cual es de ayuda tanto como para novatos como para expertos.

Arduino ha nacido en el Ivrea Interaction Design Institute como una herramienta versátil para prototipos, apuntada a estudiantes sin conocimiento en electrónica o programación. Tan pronto como alcanzó una comunidad más amplia, la placa Arduino comenzó a cambiar para adaptarse a las nuevas necesidades y desafíos. Como ya se mencionó, Arduino tiene una filosofía de hardware Open Source por lo cual los usuarios pueden construir independientemente y eventualmente adaptarse a sus necesidades particulares. El software también es de código abierto, y se encuentra en continuo crecimiento gracias a las contribuciones de usuarios alrededor del mundo. [1]

3.2.1 Placas Arduino de todo tipo

Existen un gran número de modelos de placas Arduino. Cada modelo posee características propias, como mayor memoria RAM (Memoria de acceso aleatorio, en inglés Random Access memory), mayor cantidad de pines de entrada/salida, menor tamaño, etc. Para ilustrar esto a continuación se mencionaran algunas de las posibilidades junto con una breve descripción. [1]

- Arduino UNO: Es la placa estándar y la más conocida y documentada. Salió a la luz en septiembre de 2010 sustituyendo su predecesor Duemilanove con varias mejoras de hardware que consisten básicamente en el uso de un USB HID propio en lugar de utilizar un conversor FTDI para la conexión USB. Es 100% compatible con los modelos Duemilanove y Diecimila. Viene con un Atmega328p con 32Kbytes de ROM para el programa.

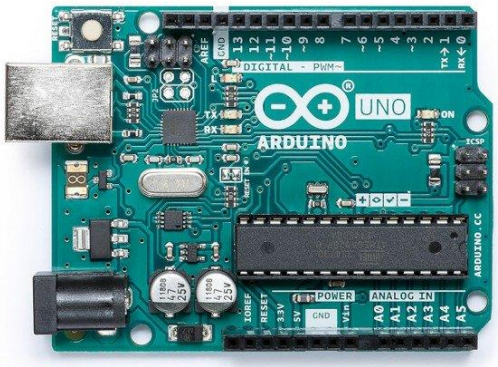


Fig. 3.2.1.1 Arduino UNO

- Arduino MEGA: Es con mucha diferencia la más potente de las placas que cuentan con un microcontrolador de 8 bits y la que más pines entrada/salida tiene, apto para trabajos de mayor complejidad aunque se tenga que sacrificar un poco el espacio. Cuenta con el microcontrolador Atmega2560 con más memoria para el programa, más RAM y más pines que el resto de los modelos de 8 bits.

Fig. 3.2.1.2 Arduino MEGA 2560

- Arduino DUE: Arduino con la mayor capacidad de procesamiento, basado en un microcontrolador de 32 bit y arquitectura ARM: Atmel SAM3X8E ARM Cortex-M3 CPU. Este Arduino opera a 3.3V y gran parte de los shields, sensores, actuadores para Arduino y compatible son a 5V lo cual es un limitante, pero cada vez se ven más elementos donde se puede elegir el voltaje entre 3.3 y 5V.

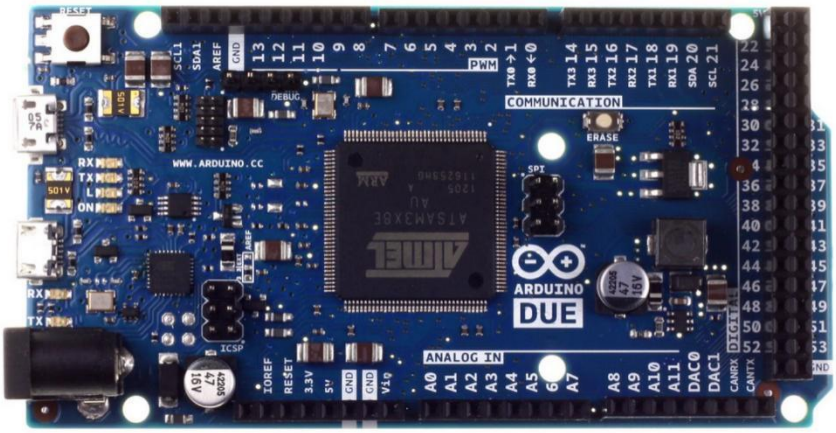


Fig. 3.2.1.3 Arduino DUE

- Arduino Leonardo: La diferencia de este Arduino con el resto es que trae un único MCU ATmega32u4 que tiene integrado la comunicación USB, lo que elimina la necesidad de un segundo procesador. Esto tiene otras implicaciones en el compartimento del Arduino al conectarlo al ordenador, lo que no lo hace apto para iniciarse con él.

Fig. 3.2.1.4 Arduino Leonardo

- Arduino Mini: Versión miniaturizada de la placa Arduino UNO basado en el ATMega328. Mide tan sólo 30x18mm y permite ahorrar espacio en los proyectos que lo requieran. Las funcionalidades son las mismas que Arduino UNO. Necesita un programador para conectarlo al ordenador.

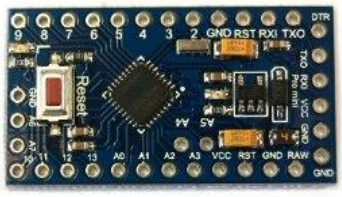


Fig. 3.2.1.5 Arduino Mini

- Arduino Lilypad: Diseñado para dispositivos “wearables” y e-textiles. Para coser con hilo conductor e instalarlo sobre prendas.

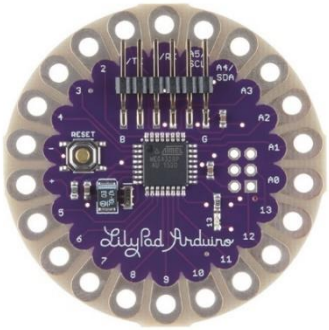


Fig. 3.2.1.6 Arduino Lilypad

3.2.2 Shields

Si se quieren ampliar las capacidades de las placas Arduino, se puede recurrir al uso de “Shields”. Estos son módulos de expansión en forma de placa impresa que se pueden conectar a la parte superior de la placa Arduino y dotarlo con capacidades adicionales, usualmente especializadas. Por otro lado, los shields pueden ser colocados unos sobre otros de manera modular. [2]

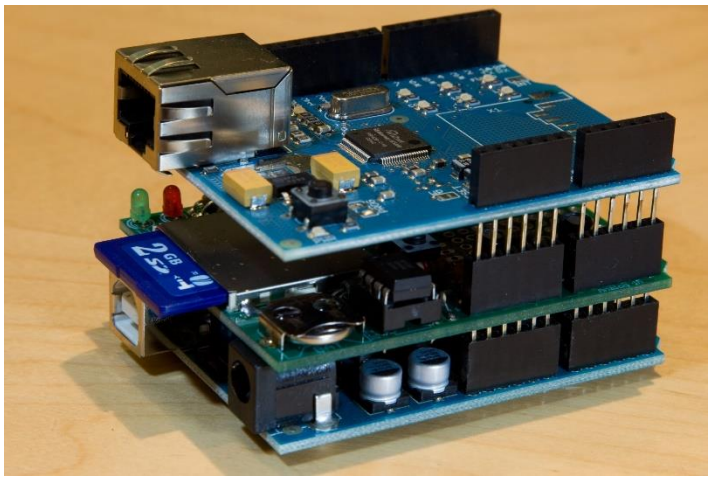


Fig. 3.2.2.1 Arduino con Shields montados

Los shields se pueden comunicar con el Arduino por algunos de los pines digitales o analógicos o bien por algún bus como el SPI, I2C o puerto serie, así como usar algunos pines como interrupción. Además estas shields se alimentan generalmente a través del Arduino mediante los pines de 5V y GND. Adicionalmente pueden existir condiciones especiales para su uso (inhabilitación de algún I/O, uso de algún bus especial, etc.) o la necesidad del uso de librerías especiales.

Algunos ejemplos:

- Ethernet Shield 2: Permite que la placa se conecte a internet. Basada en Wiznet W5500 provee una red IP capaz de TPC y UDP. Al mismo tiempo soporta hasta ocho conexiones simultáneas. Se debe utilizar la librería de Ethernet para realizar una conexión a internet mediante este shield.

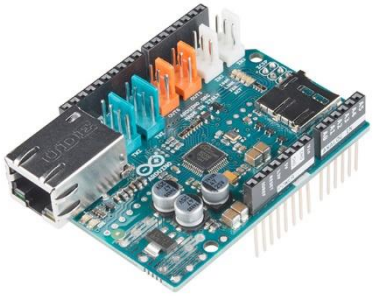


Fig. 3.2.2.2 Shield Ethernet

- Arduino Wifi Shield: Permite conectar un Arduino a Internet mediante WiFi y hace uso de la librería WiFi Library. También dispone de un slot para una tarjeta micro SD.

Fig. 3.2.2.3 Shield Wifi

- **Arduino GSM Shield**: Conecta Arduino a Internet mediante GPRS, usando una tarjeta SIM. También permite enviar y recibir mensajes y llamadas de voz (SMSs). Este dispositivo incorpora conexión GPRS/GSM a una placa base de hardware libre, dando como resultado un dispositivo de comunicación de bajo costo. En este caso por el consumo de esta shield, se hace necesario alimentar a Arduino mediante una fuente externa y no desde el USB ya que no es capaz de proporcionar suficiente energía. También es posible hacer llamadas de voz haciendo unas pequeñas modificaciones, añadiendo un micrófono y un altavoz.

Fig. 3.2.2.4 Shield GSM

3.3 Introducción a GSM

Formalmente conocida como *"Group Special Mobile"* (GSM, Grupo Especial Móvil) aunque también llamada *Global System for Mobile communications* (Sistema Global para las Comunicaciones Móviles), por el influjo del mundo anglosajón, es un estándar mundial para teléfonos móviles digitales creado por la CEPT (Conferencia Europea de Administraciones de Correos y Telecomunicaciones) y posteriormente desarrollado por el ETSI (Instituto Europeo de Normas de Telecomunicaciones) como un estándar para los teléfonos móviles europeos, con la intención de desarrollar una normativa que fuera adoptada mundialmente. El estándar es abierto, no propietario y evolutivo (aún en desarrollo) y es el estándar predominante en Europa, así como el mayoritario en el resto del

mundo (alrededor del 80% de los usuarios de teléfonos móviles del mundo en 2004 usaban GSM). GSM difiere de sus antecesores principalmente en que tanto los canales de voz como las señales son digitales. Para lograr así un moderado nivel de seguridad. [3]

El único servicio ofrecido por GSM y que no se encuentra en los sistemas analógicos más antiguos es el que nos interesa para este proyecto, el servicio de mensajes cortos o SMS. SMS es un servicio bidireccional para mensajes alfanuméricos cortos (hasta 160 bytes).

3.3.1 Red GSM

Una red GSM se encuentra formada por los siguientes componentes que integran la red pública móvil terrestre *(PLMN-Public Land Mobile Net-work) [4]*:

- La estación móvil *(MS: Mobile Station)*. Es el punto de entrada a la red móvil inalámbrica. Es el equipo físico usado por el usuario GSM para acceder a los servicios proporcionados por la red.

- El módulo de identidad del abonado *(SIM: Subscriber Identity Module)*. GSM distingue entre la identidad del abonado y la del equipo móvil. El SIM está asociado con el abonado, se trata de un chip que el usuario debe introducir en el terminal GSM.

- La estación transmisora-receptora de base o estación transceptora de base *(BTS-Base Transceiver Station)*. Se encarga de proporcionar, vía radio, la conectividad entre la red y las estaciones móviles.

- El controlador de estaciones base *(BSC-Base Station Controller)*. Se encarga de todas las funciones centrales y de control del subsistema de estaciones base *(BSS: Base Station Subsystem)* que está constituido por el BSC y las BTSs.

- La unidad de Transcodificación *(TRAU-Transcoding Rate and Adaptation Unit)*. Se encarga de comprimir la información en el interfaz aéreo cuando se hace necesario. La TRAU forma parte del subsistema BSS. Permite que tasas de datos GSM (8, 16 ,32 Kbps) puedan ser enviadas hacia la interfaz RDSI (Red digital de servicios integrados) del MSC que sólo acepta tasas de 64 Kbps.

- El centro de conmutación de servicios móviles o centro de conmutación de móviles (MSC-*Mobile Services Switching Center*). Se encarga de direccionar el tráfico de llamadas entrantes y salientes, y de la asignación de canales de usuario en la interfaz entre el MSC y las BSC.

- El registro general de abonados *(HLR-Home Location Register)*. Es una base de datos que contiene y administra la información de los abonados, mantiene y actualiza la posición del móvil y la información de su perfil de servicio.

- El registro de abonados itinerantes *(VLR-Visitor Location Register)*. Diseñado para NO sobrecargar el HLR. Guarda localmente la misma información que el HLR, cuando el abonado se encuentra en modo de itinerancia (roaming).

- El centro de autentificación *(AuC-Authentication Center)*. Genera y almacena información relativa a la seguridad, genera las claves usadas para autentificación y encriptación.

- Registro de Identidad de Equipos *(EIR: Equipment Identity Register)*. Los terminales móviles tienen un identificador único, el IMEI *(International Mobile Equipment Identity)*, el EIR se utiliza para mantener una relación de las identidades de los equipos abonados; a través de él resulta posible identificar aquellos usuarios autorizados.

- El GMSC: *Gateway Mobile Switching Center*. Es el punto hacia el cual es encaminada una terminación de llamada cuando no se tiene conocimiento de la ubicación de la estación móvil. Este componente tiene la responsabilidad por el encaminamiento de la llamada al MSC correcto.

- SMS-G. Este término es usado para describir colectivamente a dos Gateways que soportan el servicio de mensajería corta *(Short Message Services Gateways)* descritos en las recomendaciones GSM. El SMS-GMSC *(Short Message Service Gateway Mobile Switching Service)* encargado de la terminación de los mensajes cortos y el IWMSC *(Short Message Service Inter-Working Mobile Switching Center)* encargado de originar los mensajes cortos.

- Las conexiones originadas o dirigidas hacia otras redes son manejadas por un gateway dedicado, el GMSC *(Gateway Mobile Switching Center)*.

En la Figura se muestra la arquitectura del sistema GSM. Sus componentes pueden ser agrupados en tres subsistemas: El subsistema de estaciones base *(BSS: Base Station Subsytem)*, el subsistema de conmutación y gestión *(SMSS: Switching and Management Subsystem)* y el subsistema de operación y mantenimiento *(OMSS: Operation and Maintenance Subsystem).* [4]

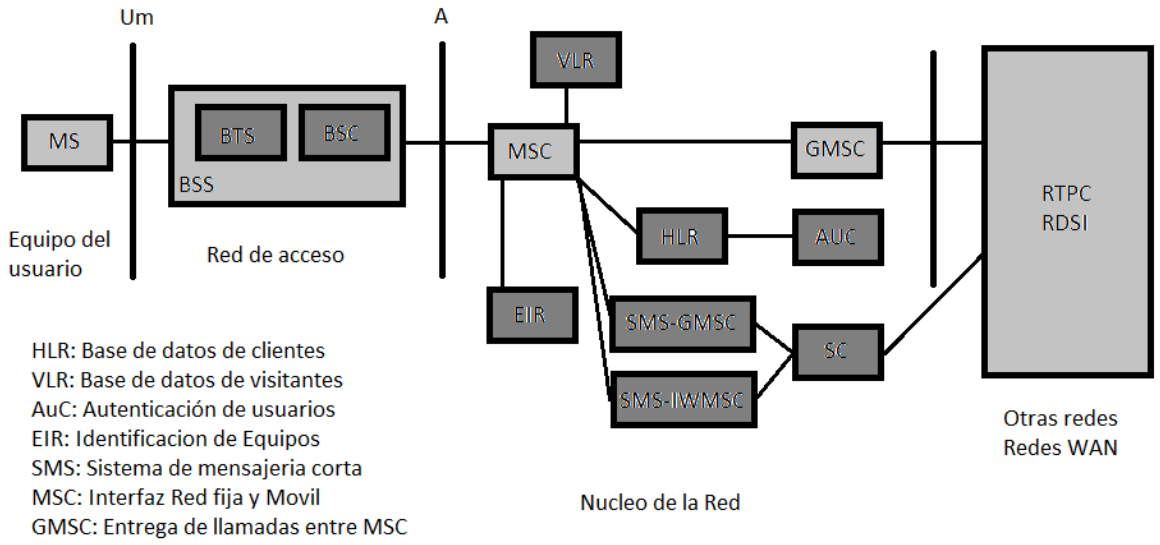


Fig. 3.3.1.1 Arquitectura sistema GSM

3.3.2 Comandos AT

En 1977 Dennis Hayes surgió con un lenguaje de comandos conocidos como AT (Attention Commands). Los comandos se originaron debido a la necesidad de configurar un modem, sin embargo la simplicidad y la fácil implementación llevo a que estos se apliquen en una amplia gama de dispositivos. Hoy en día el uso de comandos AT incluye fax, llamadas de voz, SMS, entre otras. [5]

La telefonía móvil GSM ha adoptado el lenguaje como estándar. Debido a esto, todos los teléfonos móviles GSM poseen comandos AT específicos que pueden ser utilizados para configurar y proporcionar instrucciones a las terminales. Dentro de la documentación técnica de los terminales GSM pueden encontrarse los comandos para realizar diversas acciones, tales como envío de mensajes SMS, iniciar llamadas entre otras.[3]

Los comandos AT inician su cadena con AT y finalizan con un retorno de carro <CR>. Luego de recibir el comando este es procesado y se devuelve una cadena de caracteres como resultado.

Ejemplo:

Comando enviado: AT+CMGS="+542236xxxxxx"<CR> Donde <CR> es retorno de carro

Texto adicional: Esto es un mensaje de texto<Ctrl+z> Donde <Ctrl+z> es fin del mensaje

Respuesta del dispositivo: +CMGS

El ejemplo anterior envía un SMS al número "+542236xxxxxx" con el mensaje ": Esto es un mensaje de texto" y en caso de ser exitoso el dispositivo responderá "+CMGS" [6]

Comandos AT usuales para manejo de SMS [3]

- AT+CPMS: Selecciona lugar de almacenamiento de los SMS.

- AT+CMGF: Selecciona formato de los mensajes SMS.
  - Modo texto
  - Modo PDU
- AT+CMGR: Leer un mensaje SMS almacenado.
- AT+CMGL: Listar los mensajes almacenados.
- AT+CMGS: Enviar mensaje SMS.
- AT+CMGW: Almacenar mensaje en memoria.
- AT+CMSS: Enviar mensaje almacenado.

3.4 Modelo OSI

El modelo OSI (Open Systems Interconection), se desarrolló por la Organización Internacional de Estandarización ISO (International Organization for Standarization) como una arquitectura para comunicaciones, con el objetivo de ser el marco de referencia en el desarrollo de protocolos estándares. El modelo OSI consta de siete capas [7]:

1. Aplicación
2. Presentación
3. Sesión
4. Transporte
5. Red
6. Enlace de Datos
7. Física

Fig. 3.4.1 Capas del modelo OSI

-Capa de Aplicación: Proporciona el acceso al entorno OSI para los usuarios, también proporciona servicios de información distribuida.

-Capa de Presentación: Proporciona a los procesos de aplicación independencia respecto a las diferencias en la representación de los datos (sintaxis).

-Capa de Sesión: Proporciona el control de la comunicación entre las aplicaciones; establece, gestiona y cierra las conexiones (sesiones) entre las aplicaciones cooperadoras.

-Capa de Transporte: Proporciona seguridad, transferencia transparente de datos entre los puntos finales; proporciona además procedimientos de recuperación de errores y control de flujo origen-destino.

-Capa de Red: Proporciona independencia a los niveles superiores respecto a las técnicas de conmutación y de transmisión utilizadas para conectar los sistemas; es responsable del establecimiento, mantenimiento y cierre de las conexiones.

-Capa de Enlace de Datos: Proporciona un servicio de transferencia de datos seguro a través del enlace físico; envía bloques de datos (tramas) llevando a cabo la sincronización, el control de errores y de flujo necesarios.

-Capa de Física: Se encarga de la transmisión de cadenas de bits no estructurados sobre el medio físico; está relacionada con las características mecánicas, eléctricas, funcionales y de procedimiento para acceder al medio físico. [7]

3.5 Modbus

Modbus es un protocolo de comunicaciones basado en la arquitectura del tipo maestro/esclavo o cliente/servidor, donde el cliente necesita adquirir datos del servidor o enviar datos al servidor. Modbus es un protocolo de capa de aplicación. Es decir, su objetivo es el de definir reglas que permitan organizar e interpretar datos, con lo que se erige como un sistema que soporta el envío de mensajes, sin que tenga importancia la capa física. Esto le confiere una gran versatilidad, hasta el punto de que se pueden transferir mensajes a través de otras redes mediante técnicas en encapsulación y desencapsulación. [7]

Desde que Modicon creó Modbus en 1979 para sus PLC’s no ha dejado de extenderse su uso haciéndose estándar de facto por tener las siguientes características [8]:

- Es público y gratuito
- Es fácil de implementar y no requiere mucho desarrollo
- Maneja bloques de datos sin restricciones

3.5.1 Protocolo Modbus sobre distintas capas físicas

Existen distintas capas físicas del protocolo Modbus:

- Puerto serie (RS-232,RS-485)
- Ethernet (Modbus/TCP (Protocolo de control de transmisión, en inglés Transmission Control Protocol)): es muy semejante al formato RTU (unidad terminal remota, en inglés Remote Terminal Unit), pero estableciendo la transmisión mediante paquetes TCP/IP (Protocolo de internet, en inglés Internet Protocol)

Dentro de la opción puerto serie existen dos tipos de representación:

- Modbus RTU es una representación binaria compacta de los datos. Finaliza la trama con el control de redundancia cíclica (CRC).
- Modbus ASCII es una representación legible del protocolo pero menos eficiente. Utiliza en el final el control de redundancia longitudinal (LRC).

Por ultimo existe una versión extendida del protocolo y propiedad de Modicon:

- Modbus Plus (Modbus+ o MB+), dada la naturaleza de la red precisa un coprocesador dedicado para el control de la misma. Con una velocidad de 1 Mbit/s en un par de trenzado sus especificaciones son muy semejantes al estándar EIA/RS-485 aunque no guarda compatibilidad con este.

Los protocolos Modbus RTU y Modbus ASCII se implementan a través del puerto serie, a diferencia de Modbus/TCP que se implementa a través del puerto Ethernet. [7]

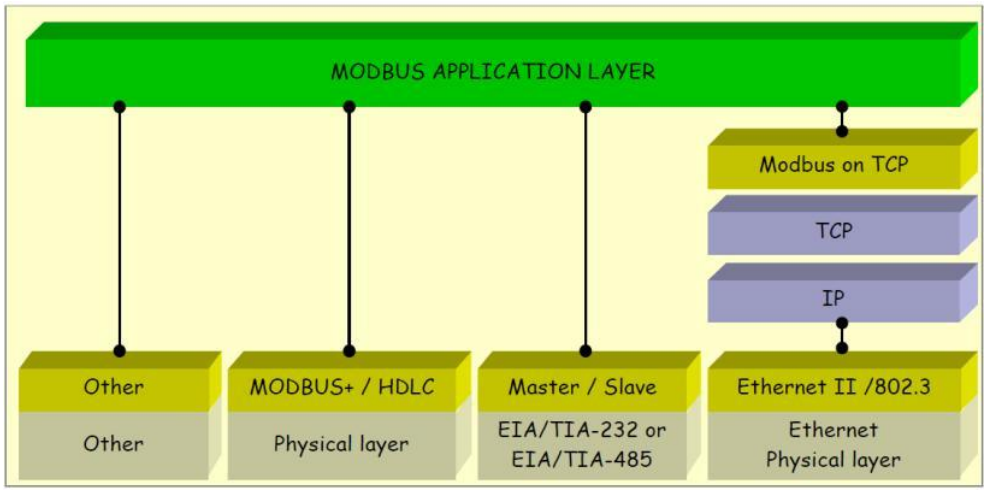


Fig. 3.5.1.1 Estructura de capas físicas Modbus

3.5.2 Modos de transmisión serial

Los controladores pueden ser configurados para funcionar en cualquiera de los dos modos de transmisión: ASCII o RTU. La elección de los modos ASCII o RTU indica

como los paquetes de información serán codificados y enviados, para luego ser recibidos y decodificados. Esto implica que la comunicación sea exitosa solo si todos los dispositivos de la red poseen la misma configuración. [9]

3.5.2.1 Modo ASCII

Cuando es utilizado el modo ASCII (American Estándar Code for Information Interchange), en cada byte de 8-bits son enviados dos caracteres ASCII. La mayor ventaja de este modo es que permite intervalos de hasta un segundo entre caracteres sin que causen errores.

El formato de cada byte en modo ASCII es:

-Sistema de codificación: Hexadecimal, caracteres ASCII 0-9, A-F
Un carácter hexadecimal contenido en cada carácter ASCII del mensaje
-Bits por Byte: 1 bit de comienzo
7 bits de datos, bit menos significativo primero
1 bit para paridad par o impar; no bits para no paridad
1 bit de detención si la paridad es usada; 2 bits en caso contrario
-Comprobación de errores: Comprobación redundante de longitud (LRC) [9]

3.5.2.2 Modo RTU

Cuando se utiliza el modo RTU (Remote Terminal Unit), cada byte de 8-bits en el mensaje contiene 2 caracteres hexadecimales de 4-bits. La mayor ventaja de este modo es la mayor densidad de caracteres la cual permite mayor transferencia de datos utilizando la misma velocidad de trasferencia en baudios. Cada mensaje debe ser transmitido de manera continua.

El formato de cada byte en modo RTU es:

-Sistema de codeo: Binario de 8-bits, Hexadecimal 0-9, A-F
Dos caracteres hexadecimales contenidos en cada 8-bits del mensaje
-Bits por Byte: 1 bit de comienzo
8 bits de datos, bit menos significativo primero
1 bit para paridad par o impar; no bits para no paridad
1 bit de detención si la paridad es usada; 2 bits en caso contrario
-Comprobación de errores: Código de redundancia cíclica (CRC) [9]

De ambas opciones, la última es la utilizada en el proyecto. Esto se debe a la capacidad de transferir mayor cantidad de datos en menos tiempo.

3.5.2.3 Formato de la trama en RTU

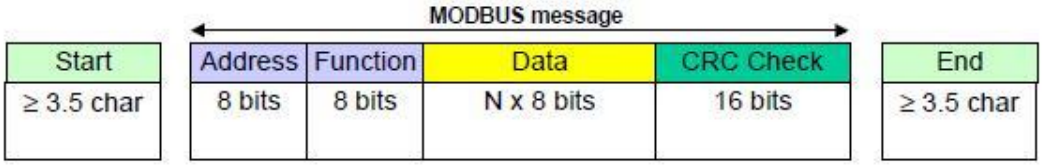


Fig. 3.5.2.3.1 Estructura de paquete Modbus

En el modo RTU, los mensajes comienzan con un intervalo de silencio equivalente al envío de 3.5 caracteres. Esto es fácilmente implementado con un múltiplo de tiempo de envío de caracteres en función de la velocidad de transferencia en baudios (indicado previamente como 3.5char).

Los caracteres permitidos en todos los campos son hexadecimales 0-9, A-F. Dispositivos conectados a la red monitorean el bus de datos continuamente, incluso durante los tiempos de “silencio”. Cuando el primer campo es recibido (Dirección), cada dispositivo decodifica el mensaje para saber a qué dirección corresponde. [9]

Continuando al último carácter transmitido, un intervalo similar de al menos 3.5 caracteres marca el final del mensaje. Un nuevo mensaje puede comenzar luego de ese intervalo.

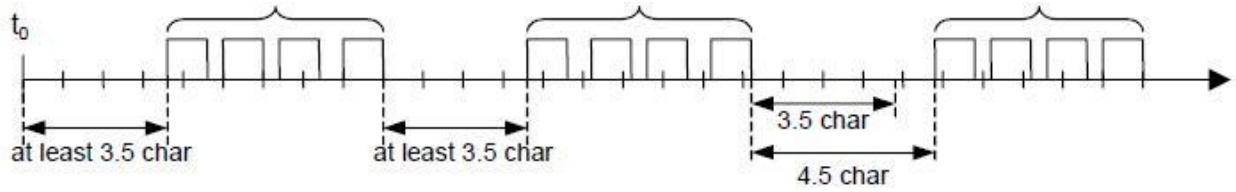


Fig. 3.5.2.3.2 Distancia entre paquetes

La trama entera del mensaje debe ser transmitida de manera continua. Si un intervalo de silencio de más de 1.5 caracteres ocurre antes de que se complete, los dispositivos que reciben el mensaje ignoran lo recibido y asumen que lo próximo que recibirán será la dirección de un nuevo mensaje.

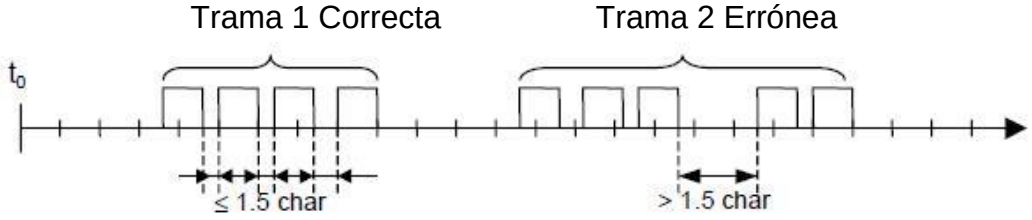


Fig. 3.5.2.3.3 Separación correcta dentro de trama

Similarmente, si un nuevo mensaje comienza antes de que pasen los 3.5 caracteres en tiempo luego del último mensaje, los dispositivos consideraran que es la continuación del mensaje previo. Esto indicara un error, dado que el valor en el campo CRC no será válido para los mensajes combinados.

Nótese que los tiempos entre trama y trama deben ser de al menos 3.5 char, esto quiere decir que el silencio inicial y el final no se suman, sino que el silencio final es el silencio inicial del próximo paquete. [8]

3.5.2.4 Campos de la trama

Dirección: Los dispositivos que funcionan como esclavos se les asignan una dirección entre 1 y 247. El maestro inicia la comunicación enviando un byte con la dirección del esclavo para el cual se destina el mensaje (modo unicast). El maestro también puede enviar un mensaje destinado a la dirección “0” (cero), lo que significa que el mensaje es destinado a todos los esclavos de la red (modo broadcast). [9]

-Modo unicast: El maestro direcciona a un único esclavo, una vez el esclavo ha recibido y procesado la petición retorna al maestro la respuesta, este último mensaje es iniciado con la dirección del esclavo que responde. En este modo de transmisión Modbus consiste en dos mensajes: la petición del maestro y la respuesta del esclavo. Cada esclavo ha de tener una dirección única.

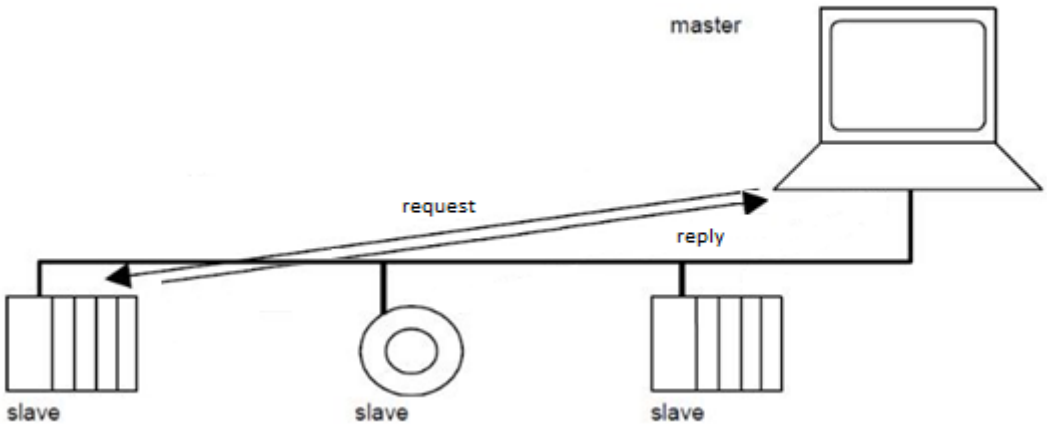


Fig. 3.5.2.4.1 Modo unicast

-Modo broadcast: el maestro puede enviar peticiones a todos los esclavos que estén conectados. Para este modo es necesario escribir comandos para la petición. Todos los elementos deben aceptar la función de escritura de broadcast. La dirección 0 se reserva para identificar el intercambio de broadcast, ya que los esclavos no emiten mensaje de respuesta y se desconoce si el mensaje ha llegado a su destino. [7]

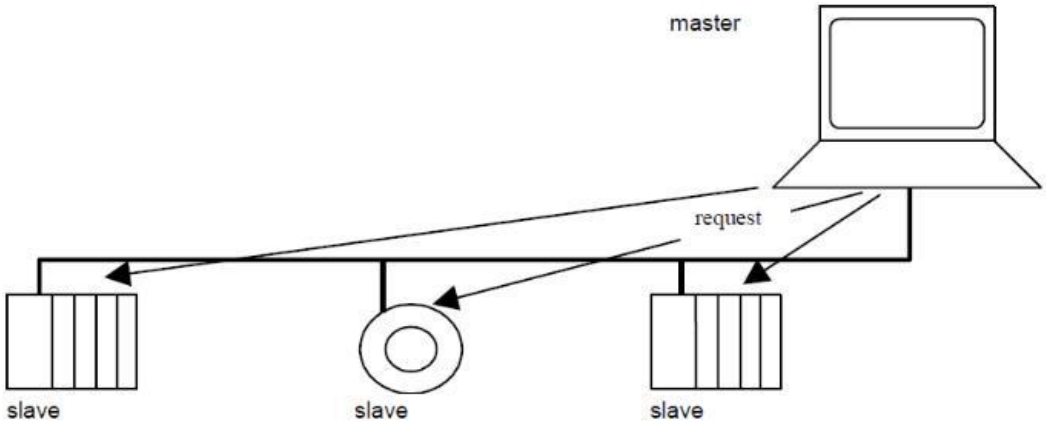


Fig. 3.5.2.4.2 Modo broadcast

Código de la función: El campo dentro de la trama de la función contiene 8 bits (RTU). El rango valido de valores es de 1 – 255 decimal. De esos, no todos son utilizados (ver Funciones).

Cuando un mensaje es enviado desde un maestro a un esclavo el campo del código de función le informa al esclavo que acción realizar. Un ejemplo de ellos seria, leer el estado ON/OFF de un grupo de bobinas. Cuando el esclavo responde el campo de función indica que no hubo problemas o bien que hubo un error.

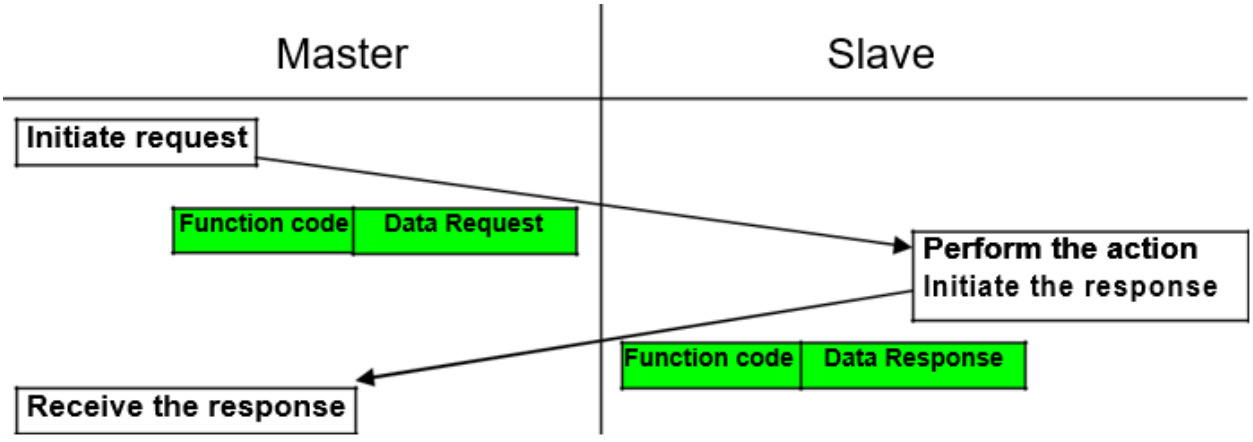


Fig. 3.5.2.4.3 Diagrama de comunicación [11]

Para una respuesta normal, el esclavo hará eco del valor de la función llevada a cabo. En caso de error, el esclavo retornara un valor equivalente al original pero con el bit más significativo definido en 1.

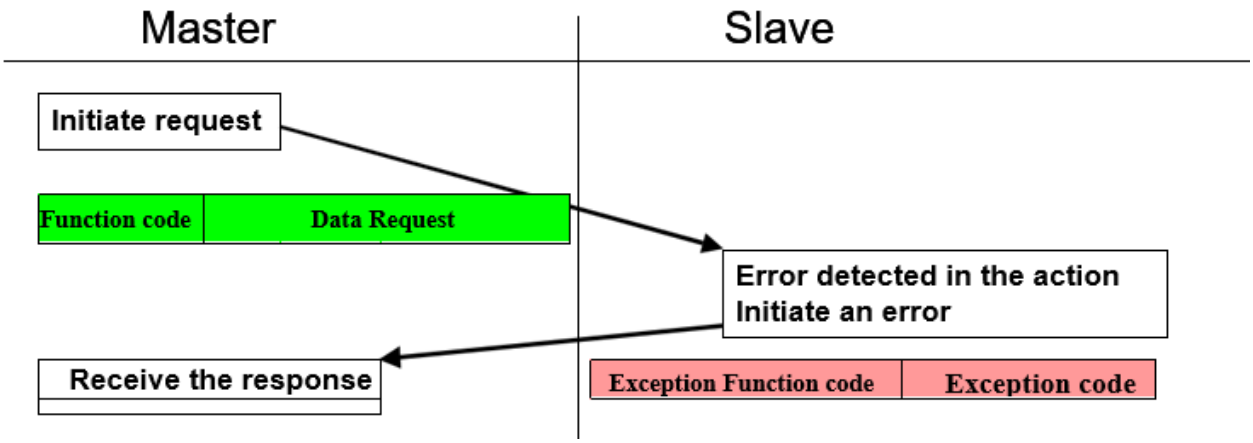


Fig. 3.5.2.4.4 Diagrama de error en comunicación [11]

EJ: 0000 0011 (hexadecimal 03): Función enviada: lectura de un grupo de registros.
1000 0011 (hexadecimal 83): Función respuesta: error en lectura de un grupo de registros.

Campo de datos: Es constituido por conjuntos de pares de dígitos hexadecimales, en el rango de 00-FF. El campo contiene información adicional que el esclavo debe utilizar para llevar a cabo la función indicada. Por ejemplo: puede incluir la dirección y el número de registros a leer (función 03). Si no hay problemas, el esclavo responde con la información solicitada por el maestro. En caso de un error, el campo contendrá el código de error.

CRC (Código de redundancia cíclica): Se aplica a la trama completa. Es un algoritmo que aplicado por el transmisor a la trama (sin incluir delimitadores) da como resultado un campo de dos bytes que se incluye en la trama antes de enviarla, de esta manera el receptor aplica el mismo algoritmo y comprueba si el resultado coincide con el campo de CRC que el transmisor incluyó en la trama. En caso contrario se ignora el mensaje. El esclavo no respondería al maestro y es el maestro el que tomará la decisión oportuna cuando expire su temporizador.

Se detalla en “Comprobación de errores de comunicación” el cálculo de CRC [9]

3.5.2.5 Comprobación de errores de comunicación

Paridad: Para transmitir un Byte de información se necesita agregar unos bits de inicio y de fin para que el receptor sepa cuando empieza y cuando acaba la información. Se puede utilizar un modo de comprobación de errores en estos bits, el uso de Paridad (Par/Impar). El bit de paridad se pone en “1” o a “0” de manera que el número total de bits a “1” en el Byte que se está enviando coincida con el modo de paridad elegido.

EJ: si se transmite “10010001” y el modo configurado es paridad par, como hay tres bits en “1”, para que el número total de bits sea par el bit de paridad tomara el valor “1”. En caso contrario se pondría en “0”.

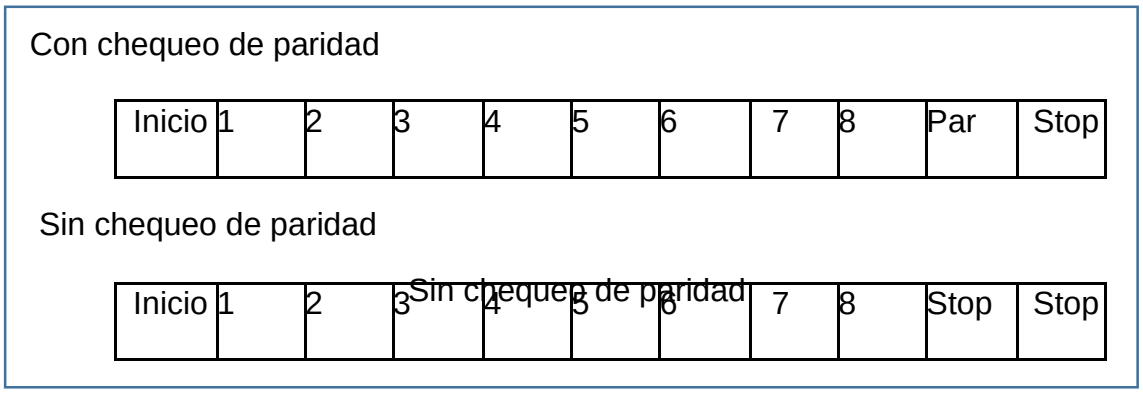


Fig. 3.5.2.5.1 Variación por paridad

En el modo de no paridad como se puede observar se sustituye dicho bit por otro bit de stop. [8]

Pasos para el cálculo de CRC

1. Se somete al primer byte del mensaje (solamente los bits de datos - start bit, paridad y stop bit no son utilizados) a una lógica XOR (O exclusivo) con los 8 bits menos significativos de la variable CRC, retornando el resultado en la propia variable CRC.

2. Entonces, la variable CRC es desplazada una posición a la derecha, en dirección al bit menos significativo, la posición del bit más significativo es rellenada con 0 (cero);

3. Después de este desplazamiento, el bit de flag (bit que fue desplazado para fuera de la variable CRC) se analiza, ocurriendo lo siguiente:

4. Si el valor del bit fuera 0 (cero), nada se ha hecho;

5. Si el valor del bit fuera 1 (uno), el contenido de la variable CRC es sometida a una lógica XOR con un valor constante de A001h y el resultado es regresado a la variable CRC.

6. Se repiten los pasos 2 y 3 hasta que se hayan realizado ocho desplazamientos;

7. Se repiten los pasos de 1 a 4, utilizando el próximo byte del mensaje, hasta que todo el mensaje haya sido procesado. El contenido final de la variable CRC es el valor del campo CRC que es transmitido en el final del mensaje. La parte menos significativa es transmitida primero (CRC-) y en seguida la parte más significativa (CRC+). [7]

Ejemplo de trama enviada y respuesta obtenida de petición de 3 registros del esclavo “06”

Nombre del campo	Ejemplo (HEX)	8bits modo RTU
Cabecera		Ninguno
Dirección esclavo	06	0000 0110
Función	03	0000 0011
Dirección inicio Hi	00	0000 0000
Dirección inicio Lo	6B	0110 1011
Num de Registros Hi	00	0000 0000
Num de Registros Lo	03	0000 0011
Error Check		CRC (16 bits)
Fin de trama		Ninguno
Total:		8 bytes

. 3.5.2.5.2 Trama de petición

Nombre del campo	Ejemplo (HEX)	8bits modo RTU	
Cabecera		Ninguno	
Dirección esclavo	06	0000 0110	
Función	03	0000 0011	
Numero de bytes de datos	06	0000 0110	
Dato 0 Hi	02	0000 0010	
Dato 0 Lo	2B	0010 1011	
Dato 1 Hi	00	0000 0000	
Dato 1 Lo	00	0000 0000	
Dato 2 Hi	00	0000 0000	
Dato 2 Lo	63	0110 0011	
Error Check		CRC (16 bits)	
Fin de trama		CR LF	ninguno

Tabla 3.5.2.5.3 Trama de respuesta

3.5.3 Funciones

Existen tres categorías de funciones [8]

- Publicas: Son códigos bien definidos, aprobados, testeados y documentados por la comunidad MODBUS
- Definidos por el usuario: Son implementaciones específicas que un usuario crea para su aplicación y que no está soportada por el estándar.
- Reservados: Algunas compañías tienen funciones reservadas para sus productos que no son accesibles públicamente.

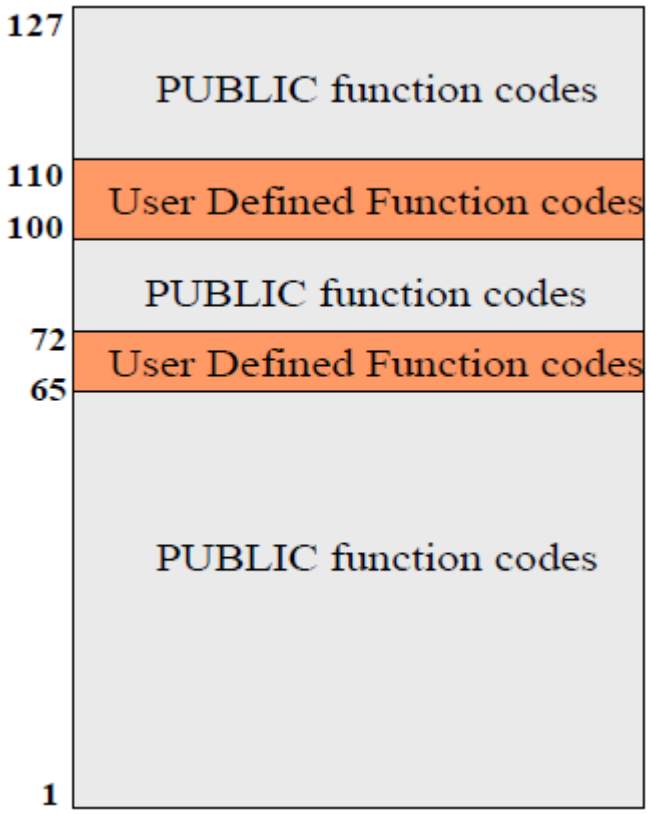


Fig. 3.5.3.1 Esquemas defunciones [11]

A continuación se detallara una lista con las funciones más habituales utilizadas.

Function Code	Type	Description
01	Read Coil Status	Requests status of discrete coils
02	Read Input Status	Requests status of discrete inputs
03	Read Holding Registers	Requests content of holding registers
04	Read Input Registers	Requests content of input registers
05	Force Single Coil	Writes to a discrete coil
06	Preset Single Register	Writes to a single holding register
08	Diagnostics (Serial only)	See Table 1.12
15	Force Multiple Coils	Writes to multiple coils
16	Preset Multiple Registers	Writes to multiple holding registers
17	Read Slave ID	Requests address of slave device

Fig. 3.5.3.2 Funciones

Dentro del proyecto se han utilizado las siguientes funciones:

- Leer estado de bobinas (Función 01): Solicita el estado de una o más bobinas.

Read Coil Status Query
Device Address
Function Code
Address of beginning coil
Total number of coils to be read
CRC

Fig. 3.5.3.3 Esquema de petición de función 01

Read Coil Status Response
Device Address
Function Code
Number of data bytes in message
Data from coils (8 coils/bits per byte; most significant bits contain the higher coils. Zeroes will be sent as placeholders if necessary).
CRC

Fig. 3.5.3.4 Esquema de respuesta de función 01

- Leer estado de entradas (Función 02): Solicita el estado de una o más entradas.

Read Input Status Query
Device Address
Function Code
Address of beginning discrete input to be read
Total number of inputs to be read
CRC

Fig. 3.5.3.5 Esquema de petición de función 02

Read Input Status Response
Device Address
Function Code
Number of data bytes in message
Data from inputs (8 inputs/bits per byte; most significant bits contain the higher inputs. Zeroes will be sent as placeholders if necessary).
CRC

Fig. 3.5.3.6 Esquema de respuesta de función 02

3.5.4 Comunicación estándar RS-485

Considerando el protocolo de comunicación entre los dispositivos, es necesario definir cómo se lleva a cabo la comunicación entre los mismos. El protocolo de comunicación RS-485 o EIA-485 se hizo estándar en 1983 por la TIA/EIA. Es un estándar de comunicación en bus para la capa física del modelo OSI y define las características eléctricas de los transmisores y receptores. La transmisión es serial y asíncrona, lo cual quiere decir que los bits se van transmitiendo uno detrás de otro y sin una señal que sincronice transmisor y receptor. El medio físico es un par trenzado (A, B) que admite hasta 256 estaciones. La comunicación es semidúplex y se pueden cubrir hasta 1200 metros con un mismo bus sin pérdida de información gracias a la transmisión diferencial que cancela gran cantidad de ruido. Las velocidades de transmisión oscilan entre los 300 y los 19200 bit/s.

RS-485	
Estándar	TIA/EIA-485-A
Medio físico	Par trenzado
Topología de red	Punto a punto, punto a multipunto, multi-drop
Modo de comunicación	Semiduplex, dúplex
Máximo de dispositivos	Originalmente 32, actualmente 256 e incluso más usando repetidores
Modo de operación	Diferencial
Niveles de tensión	-7V / +12V
“1” Lógico	Tensión positiva (B-A > +200mV)
“0” Lógico	Tensión negativa (B-A < -200mV)

Tabla 3.5.4.1 Características RS-485

La transmisión diferencial se consigue transmitiendo en cada cable la misma señal desfasada 180 grados, de manera que el ruido afecta por igual ambos cables e invirtiendo una de las señales se consigue que el ruido se cancele al sumarlas como se puede ver en la siguiente figura:

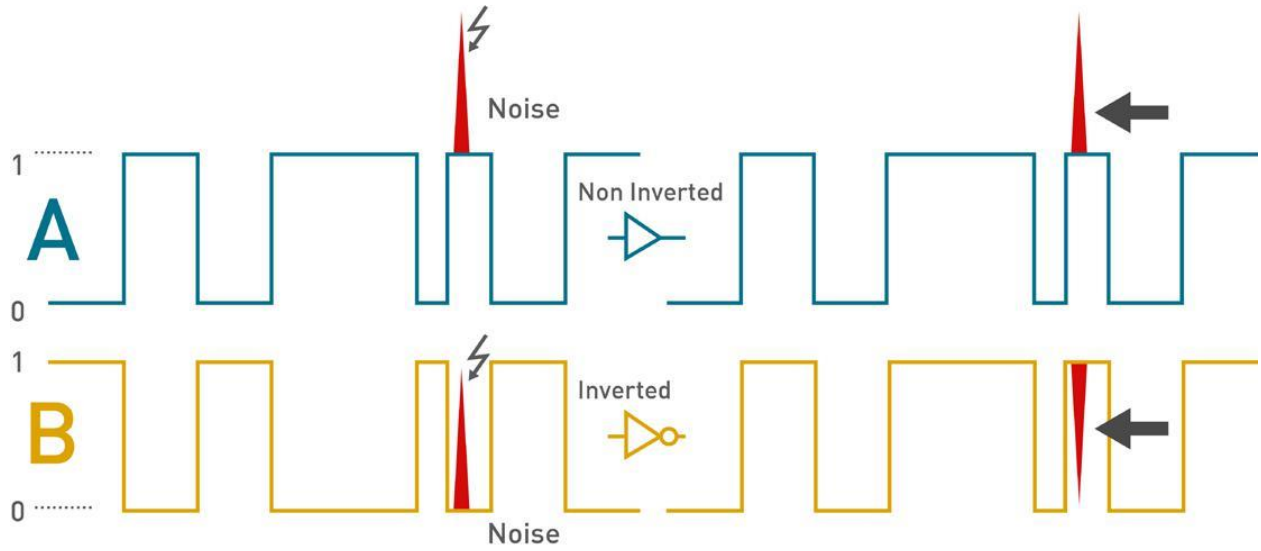


Fig. 3.5.4.2 Efectos del ruido

El esquema de la topología de red se puede ver en la siguiente figura:

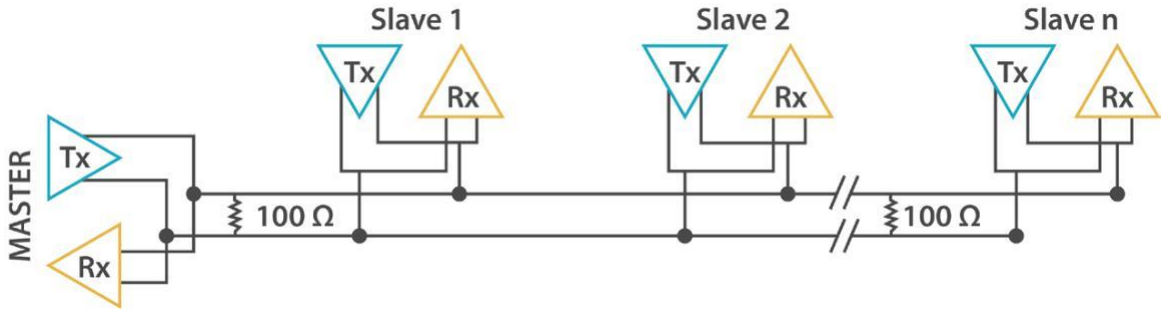


Fig. 3.5.4.3 Topología de red

Como se puede observar hacen falta sendas resistencias de terminación para que no se produzcan reflexiones en el bus. El valor de estas resistencias depende de la impedancia característica del cable y deberá ser calculado apropiadamente o consultado con el fabricante. El número máximo de dispositivos que se pueden conectar a la red es de 32, pudiéndose aumentar a 256 funcionando con entrada en alta impedancia e incluso a miles si se usan también regeneradores de señal. [8]

CAPÍTULO N°4:
PLC

4.1 Objetivo del capítulo

En el presente capítulo se explicará sobre el PLC utilizado en el proyecto. Indicando de ese modo características del dispositivo, interface de comunicación, configuración de protocolo Modbus y su programación.

4.2 S7-200 CPU 216-2

El PLC utilizado en el proyecto es el S7-200 CPU 216-2 de Siemens, se trata de un modelo que posee cierta antigüedad y se encuentra discontinuado. Esto puede representar un desafío considerando la incompatibilidad que podría ocurrir con tecnologías actuales, así también como la limitación de métodos para realizar el intercambio de datos con el dispositivo a diseñar.

A continuación se encuentra una tabla con algunas propiedades correspondientes a la serie S7-200. [10]

Función	CPU 212	CPU 214	CPU 215	CPU 216
Tamaño físico	160 mm x 80 mm x 62 mm	197 mm x 80 mm x 62 mm	218 mm x 80 mm x 62 mm	218 mm x 80 mm x 62 mm
Memoria				
Programa (EEPROM)	512 palabras	2K palabras	4K palabras	4K palabras
Datos de usuario	512 palabras	2K palabras	2,5K palabras	2,5K palabras
Marcas internas	128	256	256	256
Cartucho de memoria	No	Sí (EEPROM)	Sí (EEPROM)	Sí (EEPROM)
Cartucho de pila opcional	No	200 días (típ.)	200 días (típ.)	200 días (típ.)
Respaldo (condensador de alto rendimiento)	50 horas (típ).	190 horas (típ).	190 horas (típ).	190 horas (típ).
Entradas/salidas (E/S)				
E/S integradas	8 DI / 6 DQ	14 DI / 10 DQ	14 DI / 10 DQ	24 DI / 16 DQ
Módulos de ampliación (máx).	2 módulos	7 módulos	7 módulos	7 módulos
Imagen del proceso de E/S	64 DI / 64 DQ	64 DI / 64 DQ	64 DI / 64 DQ	64 DI / 64 DQ
E/S analógicas (ampliación)	16 AI / 16 AQ	16 AI / 16 AQ	16 AI / 16 AQ	16 AI / 16 AQ
Filtros de entrada	No	Sí	Sí	Sí

Operaciones				
Velocidad de ejecución booleana	1,2 µs/operación	0,8 µs/operación	0,8 µs/operación	0,8 µs/operación
Contadores / temporizadores	64/64	128/128	256/256	256/256
Bucles FOR/NEXT	No	Sí	Sí	Sí
Aritmética en coma fija	Sí	Sí	Sí	Sí
Aritmética en coma flotante	No	Sí	Sí	Sí
PID	No	No	Sí	Sí
Funciones adicionales				
Contadores rápidos	1 S/W	1 S/W, 2 H/W	1 S/W, 2 H/W	1 S/W, 2 H/W
Potenciómetros analógicos	1	2	2	2
Salidas de impulsos	No	2	2	2
Interrupciones de comunicación	1 emisor / 1 receptor	1 emisor / 1 receptor	1 emisor / 2 receptores	2 emisores / 4 receptores
Interrupciones temporizadas	1	2	2	2
Entradas de interrupción de hardware	1	4	4	4
Reloj de tiempo real	No	Sí	Sí	Sí
Comunicación				
Interfaces	1 (RS-485)	1 (RS-485)	2 (RS-485)	2 (RS-485)
Protocolos asistidos Interface 0:	PPI, Freeport	PPI, Freeport	PPI, Freeport, MPI	PPI, Freeport, MPI
Interface 1:	N/A	N/A	DP, MPI	PPI, Freeport, MPI
Punto a punto	Sólo esclavo	Sí	Sí	Sí

Tabla. 4.2.1 Propiedades de modelos S7-200 [10]

De las tablas anteriores, es importante tener en cuenta las posibilidades para la comunicación con el dispositivo de monitoreo, siendo de interés la interface y los protocolos a utilizar. La interface de comunicación sin el uso de módulos adicionales es únicamente RS-485. Luego, los protocolos disponibles serán PPI (Interfaz punto a punto, en inglés Point to Point Interface), MPI (Interfaz multipunto, en inglés Multi-Point Interface) o Freeport.

Existen librerías para el uso de Modbus en S7-200, pero la CPU 216-2 no es compatible con ellas. No obstante, Siemens suministra una serie de ejemplos de programación en Step7-micro/WIN bajo el nombre “All_TIPS_1208”. Encontrándose así, un ejemplo de comunicación Modbus sin la necesidad del uso de librerías adiciones. Permitiendo de ese modo con algunas modificaciones, el uso de Modbus para nuestro CPU.

CPU	Inter-face	Esclavo PPI	Maestro PPI	Esclavo PROFIBUS-DP	Esclavo MPI	Freeport	Velocidad de transferencia
212	0	Sí	No	No	No	Sí	9,6 kbit/s,19,2 kbit/s
214	0	Sí	Sí	No	No	Sí	9,6 kbit/s,19,2 kbit/s
215	0	Sí	Sí	No	Sí	Sí	9,6 kbit/s,19,2 kbit/s
	DP, DPV2	No	No	Sí	Sí	No	9,6 kbits/s, 19,2 kbits/s, 93,75 kbits/s, 187,5 kbits/s, 500 kbits/s, 1 Mbit/s,1,5 Mbit/s, 3 Mbit/s, 6 Mbit/s, 12 Mbit/s
216	0	Sí	Sí	No	Sí	Sí	9,6 kbit/s,19,2 kbit/s
	1	Sí	Sí	No	Sí	Sí	9,6 kbit/s,19,2 kbit/s

Tabla 4.2.2 Protocolos de comunicación [10]

4.3 Interface de Comunicación

Los puertos disponibles para la comunicación utilizan conectores D de nueve pines conforme al estándar PROFIBUS de la norma europea EN 50170. La siguiente figura muestra el conector que ofrece el enlace físico para la interface de comunicación.

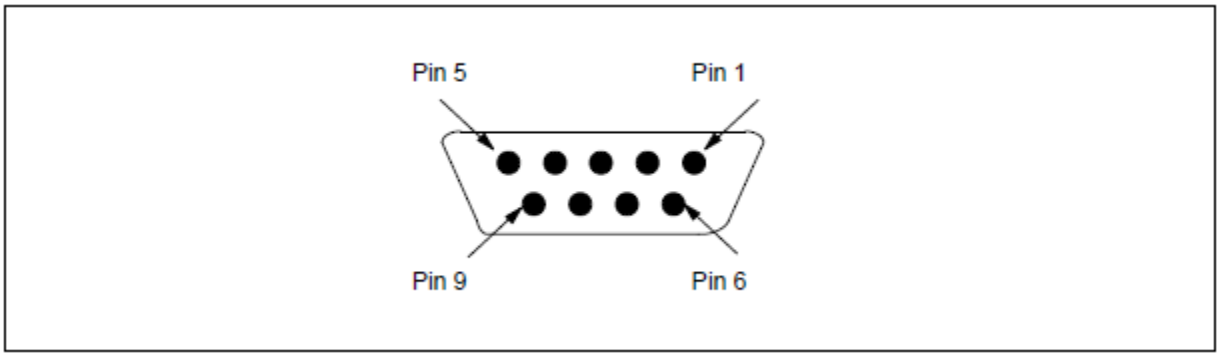


Fig. 4.3.1 Conector DB9 [10]

Donde los pines son asignados según la siguiente tabla

Pin	Denominación PROFIBUS	Interfaces 0 y 1	Interface DP
1	Blindaje	Hilo lógico	Hilo lógico
2	Hilo de retorno 24 V	Hilo lógico	Hilo lógico
3	Señal B RS-485	Señal B RS-485	Señal B RS-485
4	Request-to-Send	Sin conexión	Request-to-send[1]
5	Hilo de retorno 5 V	Hilo lógico	Isolated +5 V Return[2]
6	+5 V	+5 V, 100 Ω series limit	+5 V, con separación galvánica, 90 mA
7	+24 V	+24 V	+24 V
8	Señal A RS-485	Señal A RS-485	Señal A RS-485
9	No aplicable	Sin conexión	Sin conexión
Carcasa del enchufe	Blindaje	Hilo lógico (CPU 212/214) Tierra (CPU 215/216)	Tierra

Tabla 4.3.2 Usos de los pines [10]

De ella se destaca que para el uso de interface RS-485, se tendrá que la señal A es obtenida del PIN 8 y la señal B del 3.

4.4 Programación del PLC

4.4.1 Software de programación

Para programar el S7-200 se ha utilizado el software de programación para PLC de Siemens, “Step 7 Micro/WIN”.

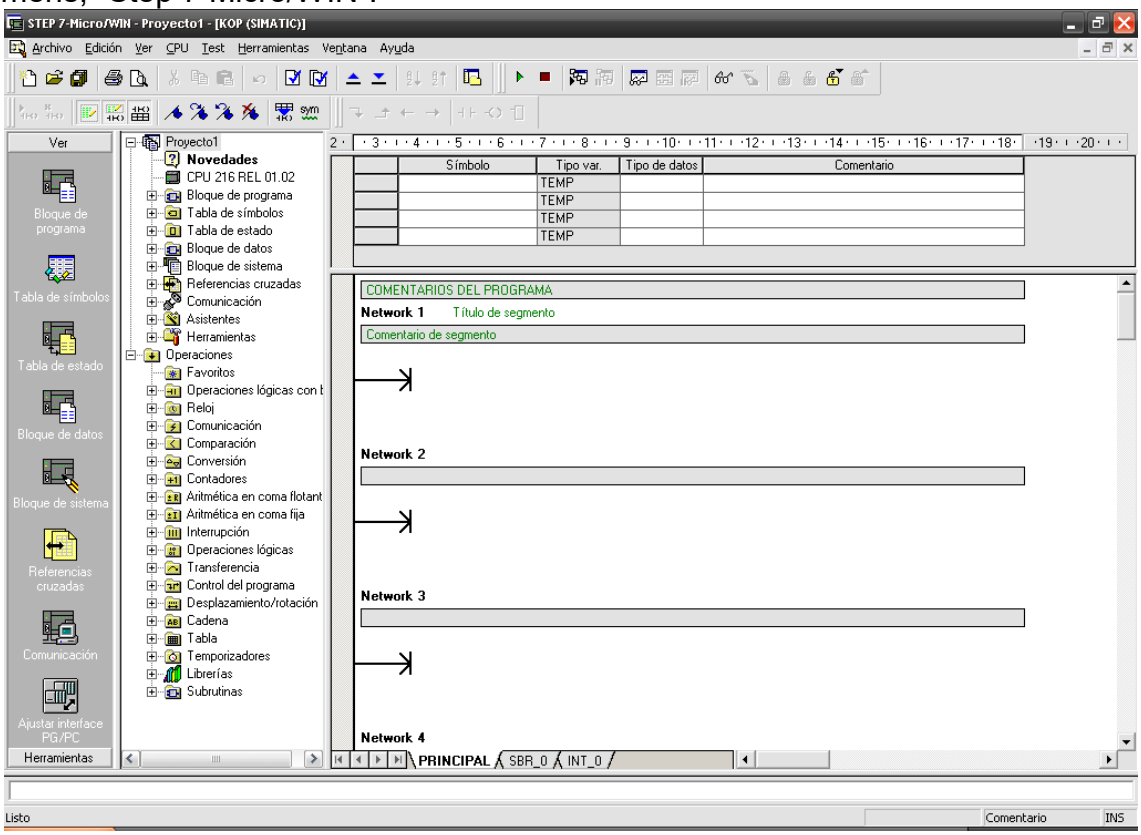
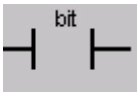


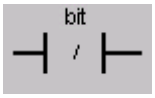
Fig. 4.4.1.1 Step 7 Micro/WIN

La programación se lleva a cabo en el lenguaje “ladder”, utilizando de esa forma los siguientes componentes:

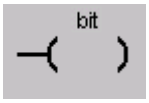
- Contacto normal abierto (NA): El contacto se cierra (ON) al recibir un bit igual a (1).



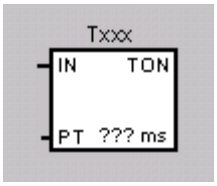
- Contacto Normal cerrado (NC): El contacto se abre (OFF) al recibir un bit igual a (1).



- Salida (bobina de un relé interno o externo): Escribe el bit recibido en la salida asignada.



- Temporizador con retardo a la conexión (TON): El temporizador (Txxx) se activa (1) si la entrada (IN) se encuentra activa (1) por un tiempo mayor al definido por el valor de preselección (PT). El temporizador se desactiva (0) si la entrada (IN) se encuentra desactiva (0).



El tiempo para la activación dependerá del valor ingresado en PT multiplicado por la base de tiempo. Los valores asignables a la base de tiempo son 1ms, 10ms y 100ms.

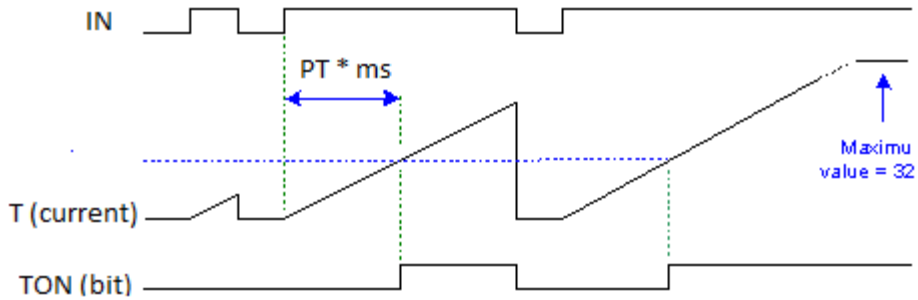


Fig. 4.4.1.2 Funcionamiento TON

4.4.2 Protocolo de comunicación

Para el uso de Modbus en nuestro PLC se parte del ejemplo mencionado con anterioridad. En el mismo se incluye un grupo de subrutinas y rutinas de interrupción con las cuales se puede crear un esclavo Modbus RTU usando funciones Freeport del S7-200.

El programa incluye las siguientes funciones de Modbus:

- 1: Lectura de salidas (Bobinas)
- 2: Lectura de entradas (Contactos)
- 3: Lectura de registros(Memoria V)
- 4: Lectura de registros de entrada.
- 5: Escritura de una salida.
- 6: Escritura de registro.
- 15: Escritura de múltiples salidas.
- 16: Escritura de múltiples registros.

En la pestaña SBR50, la cual corresponde la configuración de inicio del Driver Modbus RTU, se debe encontrar el bloque que configure SBM30 (Puerto 0). Esta configuración permitirá definir la comunicación serial, de ese modo se determinará la paridad, la velocidad de transferencia y bits por carácter.

Para configurar el mismo según lo deseado se utiliza la información suministrada en el manual de automatización del S7-200.

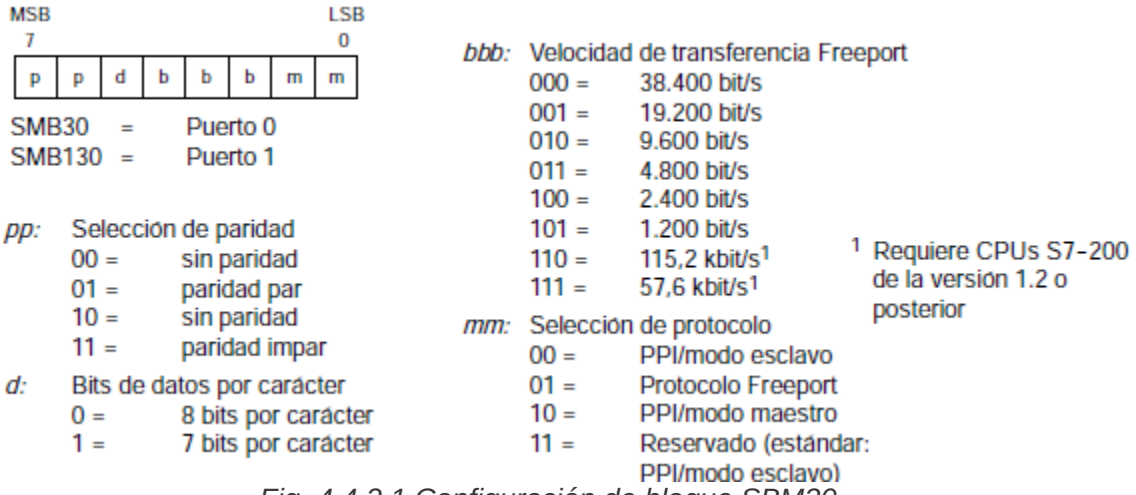


Fig. 4.4.2.1 Configuración de bloque SBM30

De la anterior imagen, se verá que el byte que de configuración se separa en 4 bloques. Por ejemplo, si se desea configurar para el funcionamiento con paridad Impar (11), 8 bits por carácter (0), con una velocidad de transferencia de 19.200 bits/s (001) y protocolo Freeport (01). Se obtendrá 11000101, que en hexadecimal es C5. Este valor será el utilizado en el proyecto.

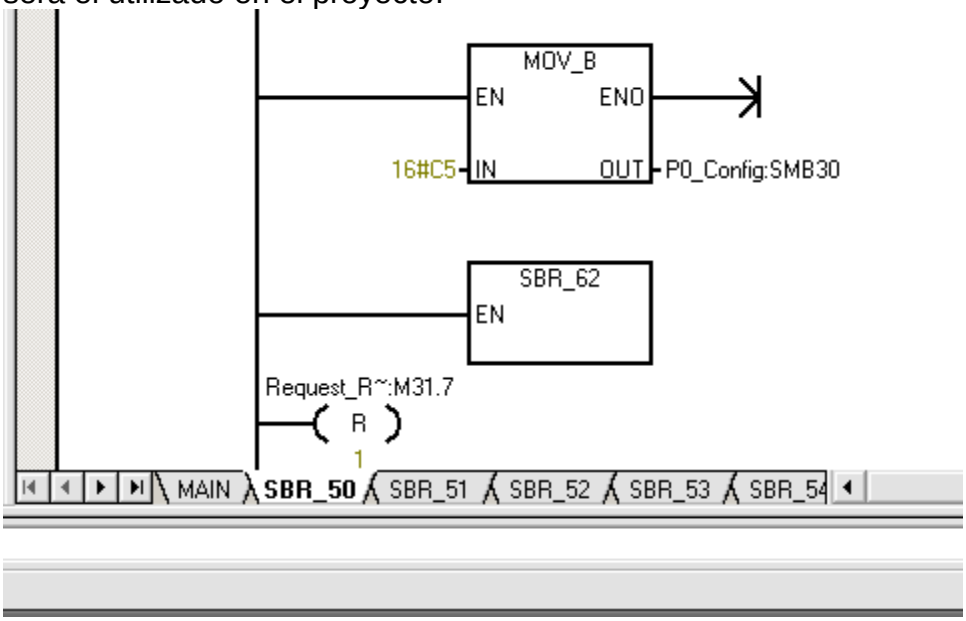


Fig. 4.4.2.2 Bloque SBM30

Nótese que el valor ingresado es “16#C5”, el cual significa que se ingresa el valor hexadecimal C5.

4.4.3 Tareas definidas dentro del PLC:

- Nexo entre sistema actual y dispositivo de monitoreo
- Control de Arduino (flags)
- Sistema de parada de emergencia
- Sistema de alarma

4.4.3.1 Tarea de monitoreo

Para la primera de las tareas basta con conectar a las entradas del PLC aquellas señales a monitorear. La próxima tabla indica cuales son las entradas y salidas asignadas

Entradas	
I0.1	Aviso quemador Q1 ENCENDIDO
I0.2	Aviso quemador Q1 FALLA
I0.3	Aviso quemador Q2 ENCENDIDO
I0.4	Aviso quemador Q2 FALLA
I0.5	Aviso quemador Q3 ENCENDIDO
I0.6	Aviso quemador Q3 FALLA
I0.7	Presencia de alimentación eléctrica
I1.1	Control de puertas Bloqueadas
I1.2	Apagado de emergencia físico (botón de golpe de puño)
I1.3	Apagado de emergencia (Arduino)
I1.4	Sensor de presencia de personas en taller fuera de horario(alarma)
I1.5	
I1.6	
I1.7	
I2.1	
I2.2	Sensor de temperatura interno Q1
I2.3	Sensor de temperatura interno Q2
I2.4	Sensor de temperatura interno Q3
I2.5	Reset quemador 1
I2.6	Reset quemador 2
I2.7	Reset quemador 3

Salidas	
Q0.1	Resetear el quemador Q1
Q0.2	Resetear el quemador Q2
Q0.3	Resetear el quemador Q3
Q0.4	Habilitar puertas del horno
Q0.5	Salida Alarma

Q0.6	Encender luces
Q0.7	Parada de emergencia
Q1.1	
Q1.2	
Q1.3	
Q1.4	
Q1.5	
Q1.6	
Q1.7	

Tabla 4.4.3.1.1 Entradas y Salidas definidas dentro del PLC

4.4.3.2 Control de Arduino (flags)

El objetivo será definir valores que permitan a Arduino verificar la posible desincronización debido a la pérdida de paquetes. Para cumplir con la tarea se espera que las entradas I0.0, I1.0 e I2.0 permanezcan desactivadas (0) y las salidas Q0.0 y Q1.0 permanezcan activas (1).

Para definir las salidas se utilizaron los contactos NC del relé interno M0.1 la cual solamente es utilizada en esta parte del programa, por lo que siempre permanece en estado (0). Al usarse un contacto NC se define de ese modo a Q0.0 y Q1.0 como activas (1).

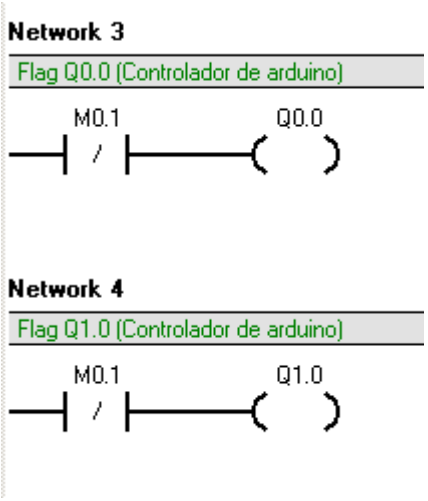


Fig.4.4.3.2.1 Flags

4.4.3.3 Parada de emergencia

La misma será activable con cualquiera de las dos entradas asignadas I1.2 (Botón de golpe de puño) y I1.3 (Señal proveniente de Arduino) la cual será activada con una señal que se mantendrá con el tiempo. Dicha señal estará directamente conectada con la salida QX.X

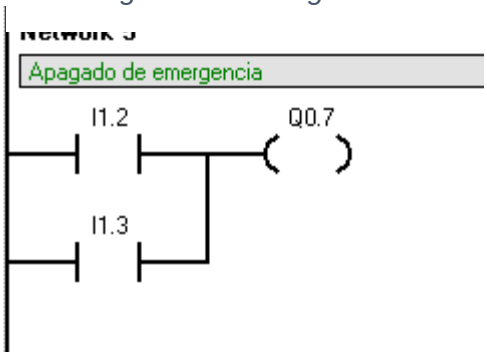


Fig. 4.4.3.3.1 Apagado de emergencia

4.4.3.4 Sistema de alarma

Se ha configurado un sistema de alarma considerando un sensor colocado en una puerta, el mismo se ha programado con la idea de darle cierta tolerancia para evitar que sea activada por golpes o vibraciones.

La entrada I1.4 corresponde al sensor en cuestión. Al activarse y mantenerse en ese estado por al menos 500ms, generara la activación del temporizador T33. El retraso de activación permite obtener cierta tolerancia ante eventos como vibraciones o golpes.

El relé interno (M0.0) activará el temporizador T37 luego de 120s. (Ver Network 8 para activación de M0.0)

Al cerrarse T33, se activará el relé interno (M0.0) y la alarma (Q0.5). M0.0 cumplirá la función de retención cuando T33 se desactive. Luego de 120s T37 se activará generando que el normalmente cerrado de T37 de abra, deteniendo de ese modo el relé interno (M0.0) y la Alarma (Q0.5). El sistema luego de eso podrá activarse nuevamente en caso de necesidad.

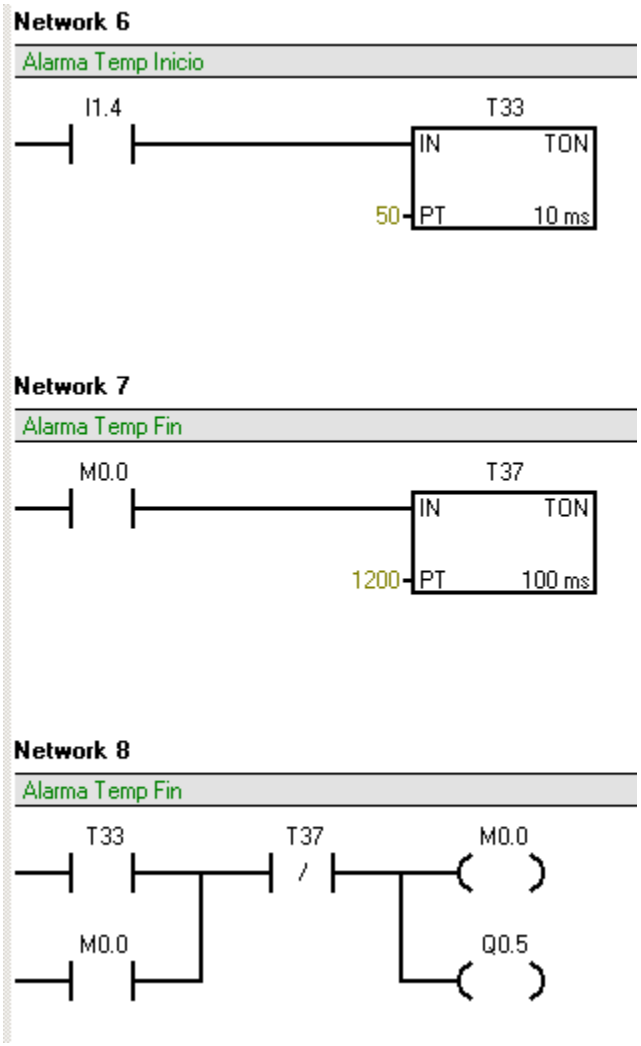


Fig. 4.4.3.4 Sistema de Alarma

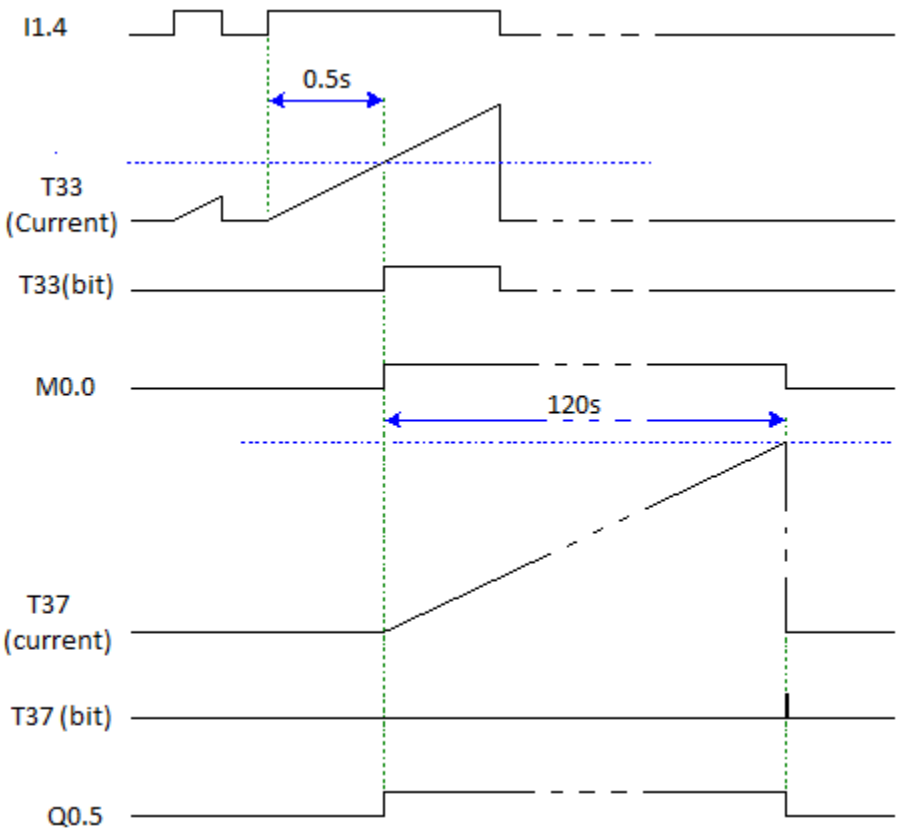


Fig. 4.4.3.4.2 Funcionamiento de alarma

4.4.3.5 Horario de trabajo

El objetivo de este bloque es destrabar las puertas de los hornos (Q0.4) e indicar a Arduino que se estaría operando durante horario laboral.

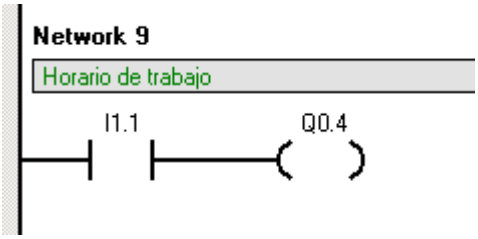


Fig. 4.4.3.5.1 Horario de trabajo

4.4.3.6 Reset de hornos

El siguiente sistema es análogo a los 3 hornos. Cuando se recibe una activación en la entrada de reinicio, si el sistema se encuentra en fallo (I0.2 para el horno 1) se activará la salida de reinicio (Q0.1 Horno 1). Al transcurrir 50ms el temporizador se activará generando que el normalmente cerrado del mismo sea abierto. Entonces, la señal de salida será un pulso de 0.5s de duración.

Tabla 4.4.3.6.1 Entradas / Salidas

	Horno 1	Horno 2	Horno 3
Entrada reset	I2.5	I2.6	I2.7
Entrada Falla	I0.2	I0.4	I0.6
Temporizador	T34	T35	T36
Salida reset	Q0.1	Q0.2	Q0.3

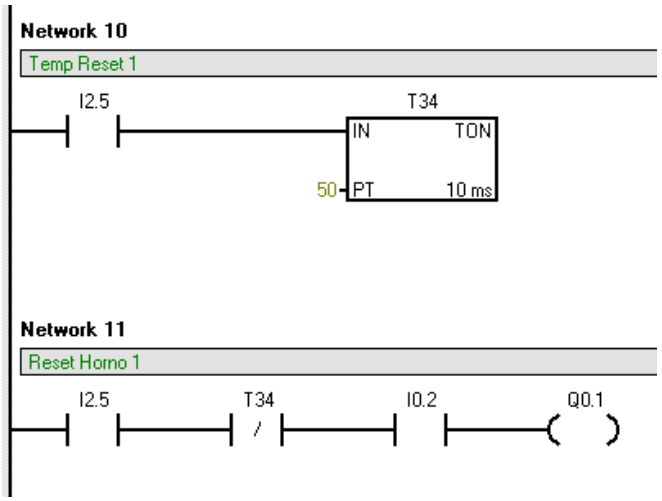


Fig. 4.4.3.6.2 Reset

4.4.4 Carga de programa

Finalmente, una vez realizado el programa es necesario cargarlo al PLC. Se debe posicionar el PLC en modo STOP y conectarlo con la computadora utilizando el cable de conexión PPI (previa verificación de la velocidad configurada en el mismo). Una vez conectado se debe hacer doble clic en “Comunicación”.

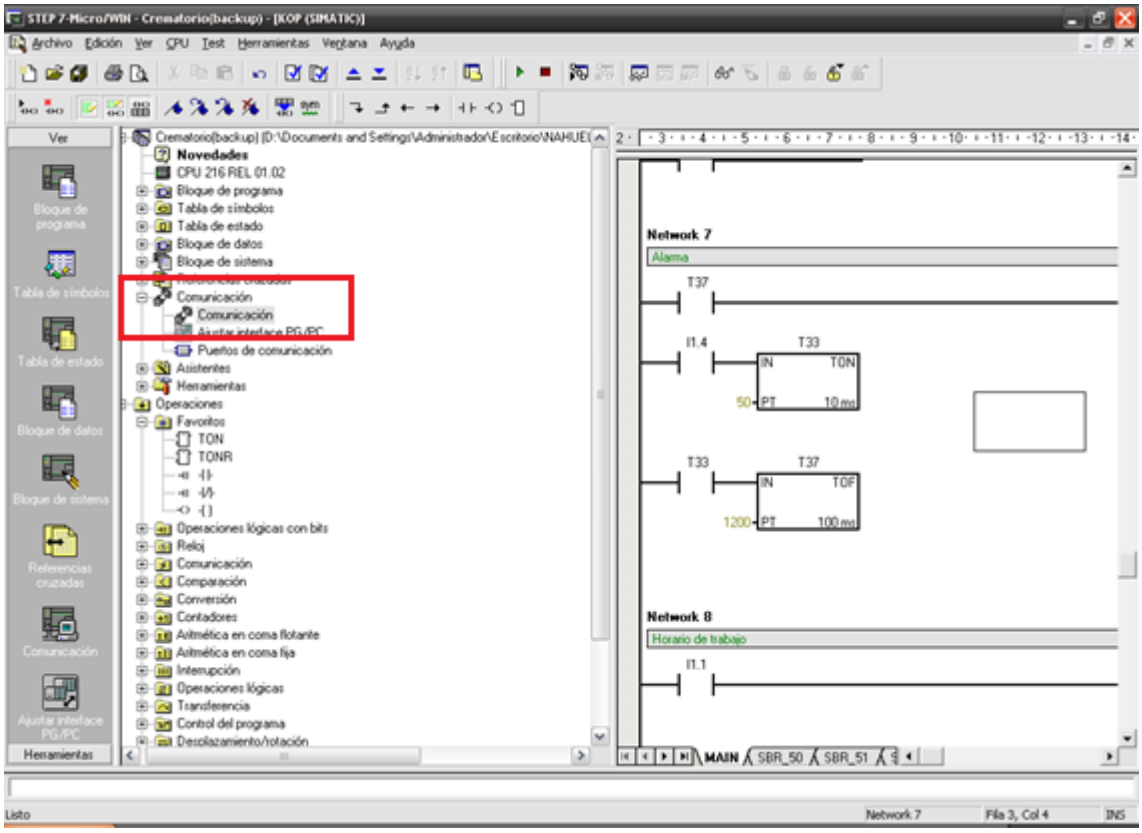


Fig. 4.4.4.1 Step 7 Micro/WIN comunicación

Lográndose ver la siguiente ventana

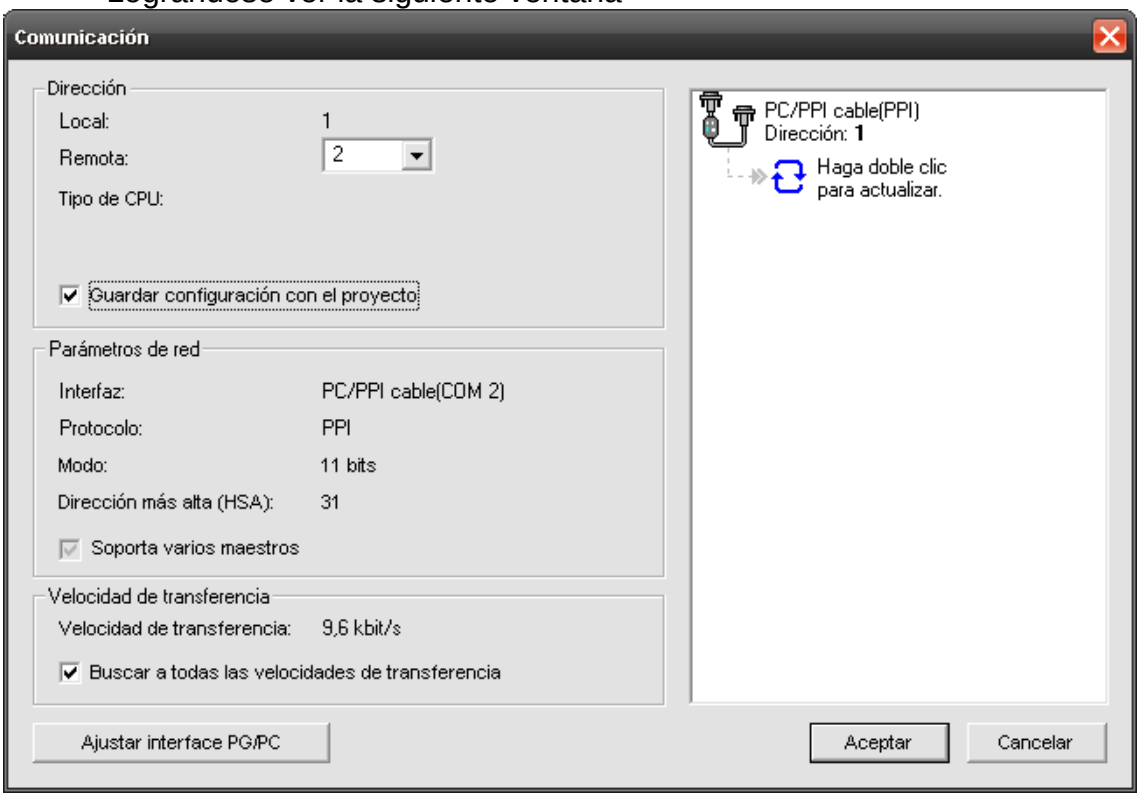


Fig. 4.4.4.2 Step 7 Micro/WIN Ventana de Comunicación

Haciendo doble clic en “Haga doble clic para actualizar”

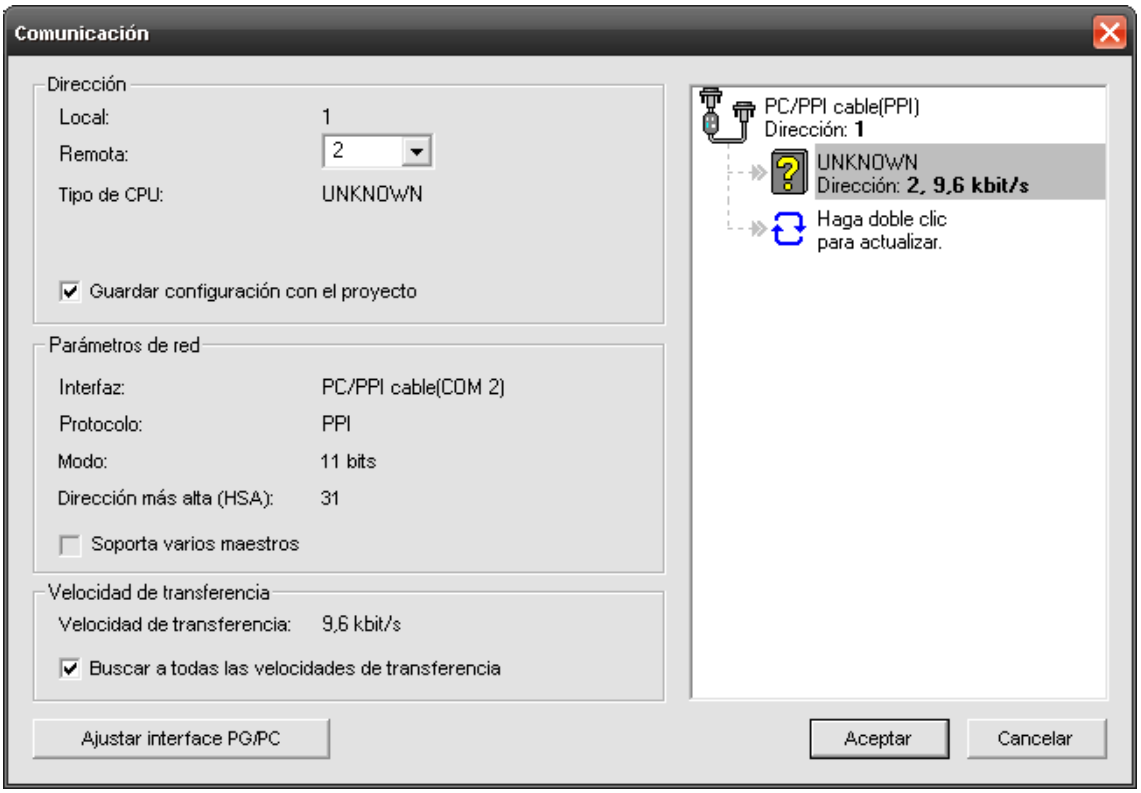


Fig. 4.4.4.3 Step 7 Micro/WIN Ventana de Comunicación, dispositivo encontrado

Nota: Se recomienda conectar el PLC en alguno de los puertos COM del 2 al 9

Doble clic en “Unknown” COM: 2, 9600 kbit/s para elegir nuestro dispositivo, que en nuestro caso es el PLC S7-200 CPU 216-2. Luego de ello se podrá cargar el programa, para ello hay que hacer clic en el botón “Cargar a CPU” ubicado en la parte superior de la ventana principal. Finalmente hacer clic en “Cargar a CPU” en la ventana de carga.

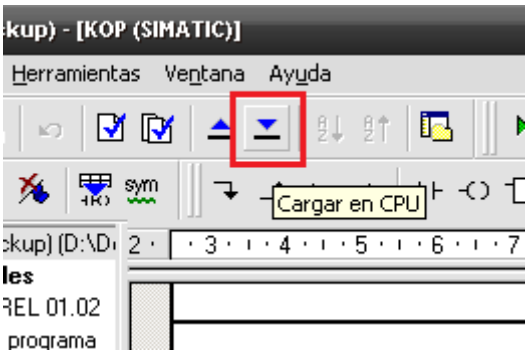


Fig. 4.4.4.4 Step 7 Micro/WIN Botón “Cargar en CPU”

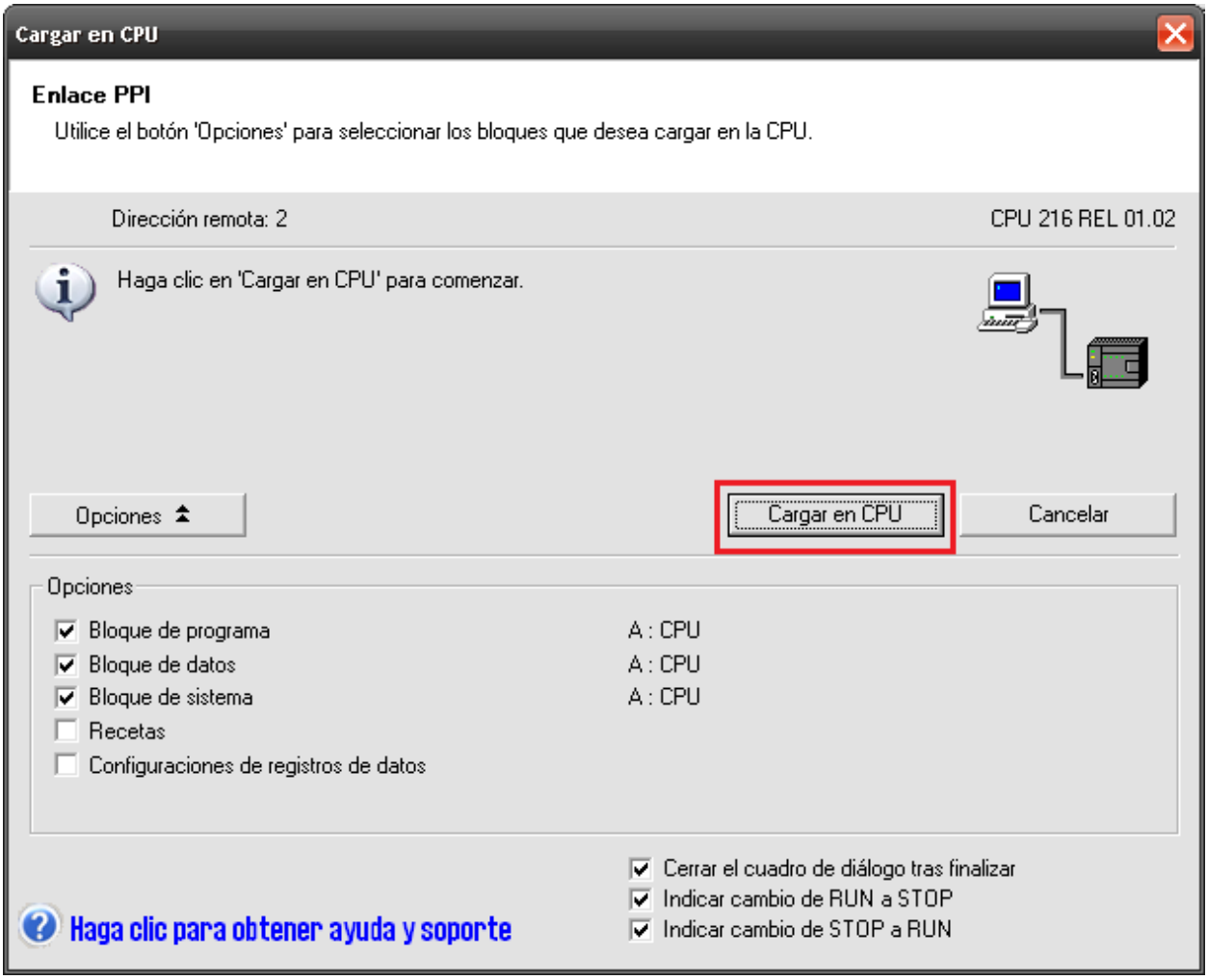


Fig. 4.4.4.5 Step 7 Micro/WIN Ventana de cargar en CPU

4.4.5 Posibles actualizaciones

El programa diseñado en este capítulo está diseñado en base a la instalación actual. En el lugar existe la posibilidad de expandir el sistema actual agregando sensores varios para mayor seguridad, como podría ser un sensor detector de humo o sensores de movimiento para detectar posibles intrusos. Esto implicaría un cambio en el programa, el mismo será compatible con el dispositivo de monitoreo siempre que se respete que las entradas I0.0, I1.0 e I2.0 permanezcan sin uso y las salidas Q0.0 y Q1.0 permanezcan activas. Por supuesto, también se debe programar adecuadamente utilizando lo visto en el capítulo 7 (Manual de usuario).

CAPÍTULO N°5: ARDUINO

5.1 Objetivos del capítulo

El capítulo presenta información sobre el proceso de armado y programación del dispositivo de monitoreo teniendo en cuenta el corazón del mismo, Arduino. Explicando el software utilizado para la creación del programa, las conexiones realizadas con módulos y shields que permiten la realización del proyecto.

5.2 Arduino Mega 2650

Fig. 5.2.1 Arduino MEGA 2650

Es una placa basada en el micro controlador Atmega2560 tiene 54 pines digitales de Entradas/Salidas (de los cuales 15 pueden ser utilizados como salidas PWM (Modulación por ancho de pulsos, en inglés Pulse Width Modulation)), 16 salidas analógicas, 4 UARTs (Transmisor-Receptor Asíncrono Universal), oscilador de cristal de 16 MHz, conexión USB, conector de alimentación, entrada ICSP (In Chip Serial Programmer) y un botón de reset. La placa Mega 2560 es compatible con la mayoría de shields designados para Uno y las antiguas Duemilanove o Diecimila. Permitiendo de este modo grandes posibilidades para el proyecto. [1]

5.3 Módulos y Shields

Modulo conversor MAX485 de serial TTL a RS485:

Permite la conversión bidireccional de señal TTL (lógica transistor a transistor, en inglés transistor-transistor logic) de Arduino a RS485, por medio del uso de un transductor MAX485.

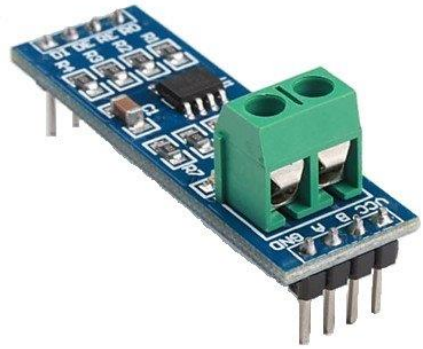


Fig. 5.3.1 MAX485

Display LCD 2004A

La pantalla alfanumérica LCD (pantalla de cristal líquido, en inglés liquid crystal displays) posee 4 líneas de 20 caracteres para mostrar letras, números y símbolos. Dispone de retroiluminación color azul y caracteres blancos.

Fig. 5.3.2 LCD 2004A

Controlador I2C LCD

Modulo adaptador de LCD a interfaz I2C (Circuito inter-integrado, en inglés Inter-Integrated Circuit) el cual permite manejar el display LCD con el uso de 2 pines (SDA y SCL). Especialmente utilizado para controlar LCD Alfanuméricos.

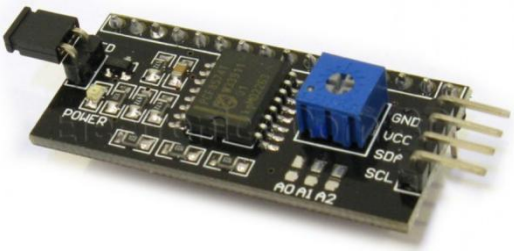


Fig. 5.3.3 I2C

Icomsat V1.1 SIM900

Shield GSM/GPRS para Arduino, basado en el módulo GSM/GPRS cuatribanda SIM900. Controlado mediante comandos AT

Fig. 5.3.4 Shield GSM/GPRS

Deek-Robot Data logger Shield

Shield el cual permite el uso de un lector de tarjetas SD integrado y un RTC (Real Time Clock) con pila. En adición el mismo posee un área de prototipos permitiendo así el montaje de componentes adicionales.

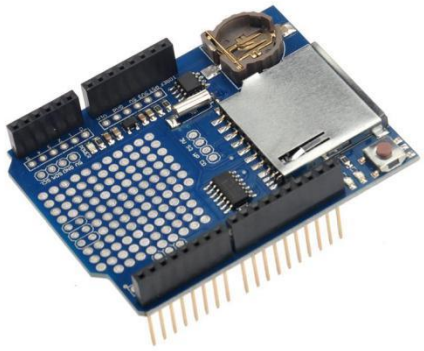


Fig. 5.3.5 Shield SD

5.4 Arduino IDE

Para la programación de las placas Arduino suele ser utilizado el programa Arduino IDE (Integrated Development Enviroment) el cual es una aplicación multiplataforma (Windows, Linux, macOS). El IDE de Arduino permite la programación muy similar a C. Para el desarrollo del proyecto se ha utilizado la versión Arduino IDE 1.6.11.

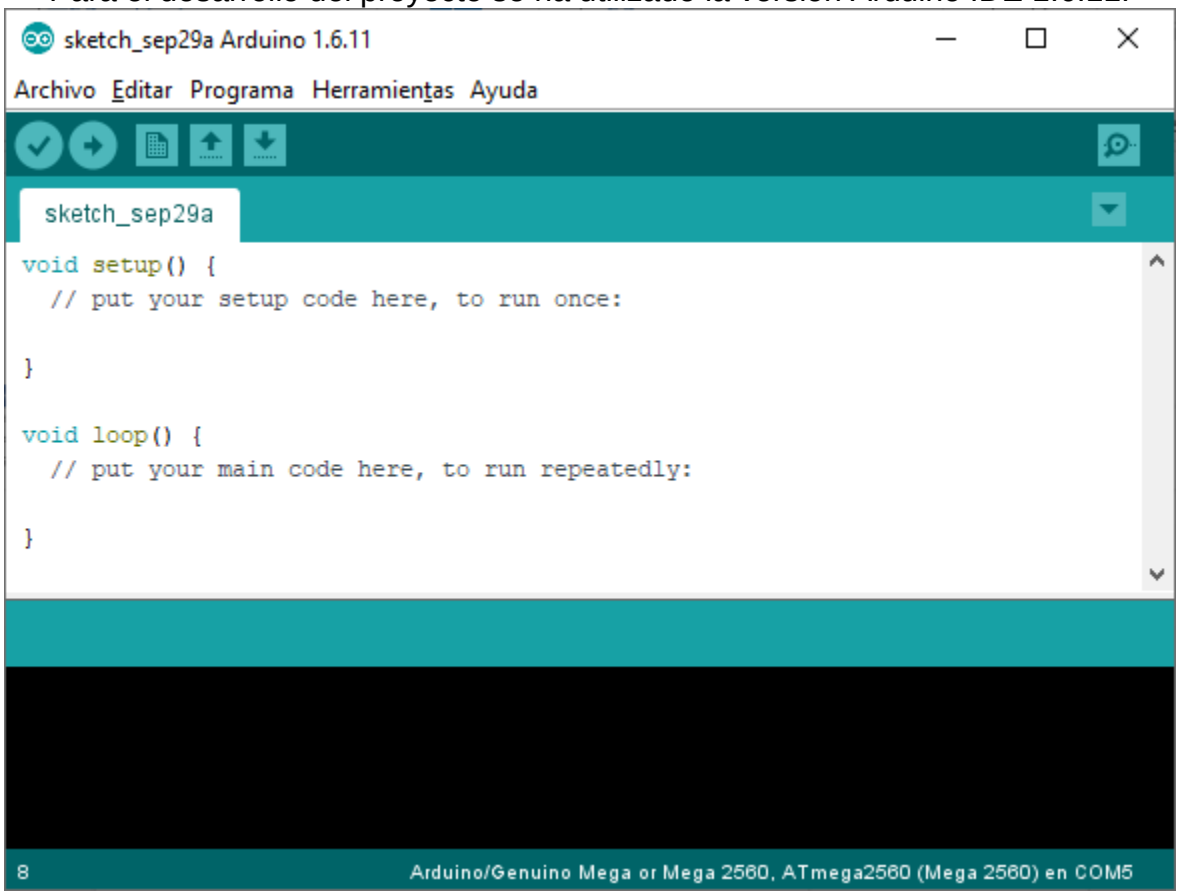


Fig. 5.4.1 Arduino 1.6.11

5.5 Funcionamiento del programa en Arduino

Para poder comprender el funcionamiento, es necesario tomar algunas nociones de la estructura básica de los programas en Arduino. En general los mismos son diseñados utilizando la misma estructura, la cual se divide en tres partes (cinco si se incluyen funciones propias e interrupciones).

- Declaraciones (librerías, variables, constantes, etc.)
- Setup
- Loop
- Funciones (opcional)
- Interrupciones (opcional)

De tal modo que se puede ver de la siguiente manera

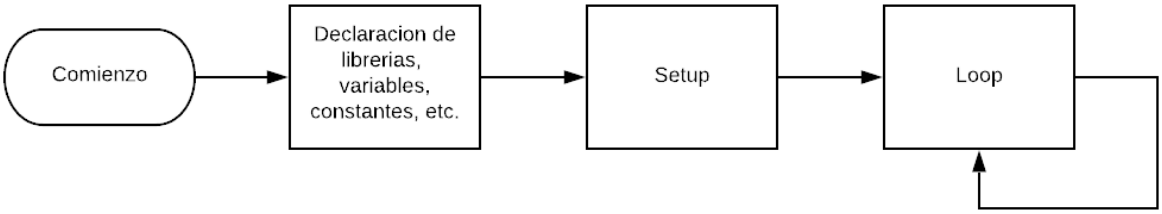


Fig. 5.5.1 Diagrama del programa

El bloque de código dentro de “Declaraciones” se utiliza para indicar las librerías de funciones a utilizar, las variables globales las cuales se usarán en el resto del programa, asignaciones de valores a constantes, etc. Las declaraciones son vitales, dado que el programa no podrá funcionar con algo que no haya sido declarado anteriormente.

El código dentro del bloque Setup será ejecutado una única vez, de tal forma que es utilizado para dar valores iniciales a las variables, ejecutar funciones de configuración, entre otras tareas las cuales estén destinadas a poner a punto el sistema para funcionar.

Loop es el bloque de código que se repite cíclicamente indefinidamente, es aquí donde se encuentra la estructura principal del programa.

Finalmente fuera de esta estructura se pueden encontrar diferentes bloques, los cuales pueden ser “Funciones” o “Interrupciones”. El objetivo principal de las funciones es simplificar otros bloques reduciendo las repeticiones de código. Las interrupciones cumplen la tarea de ejecutar algunas líneas de código cuando ocurre algún evento (EJ: Activación de una entrada).

Setup

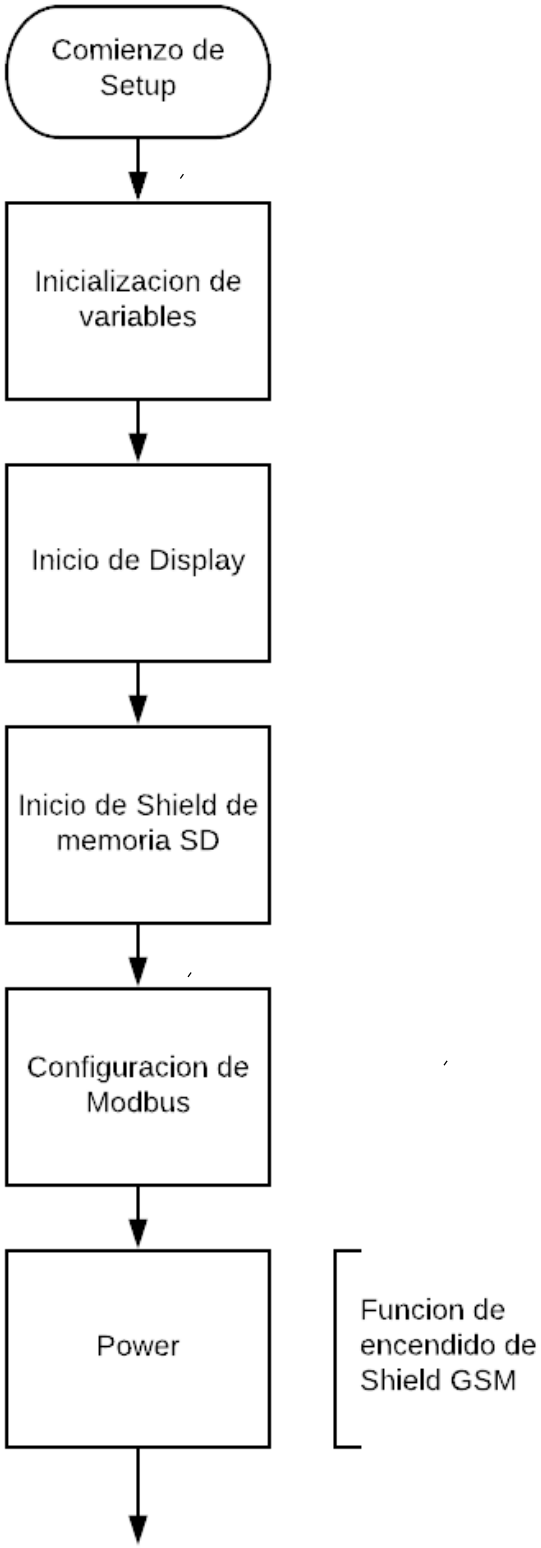
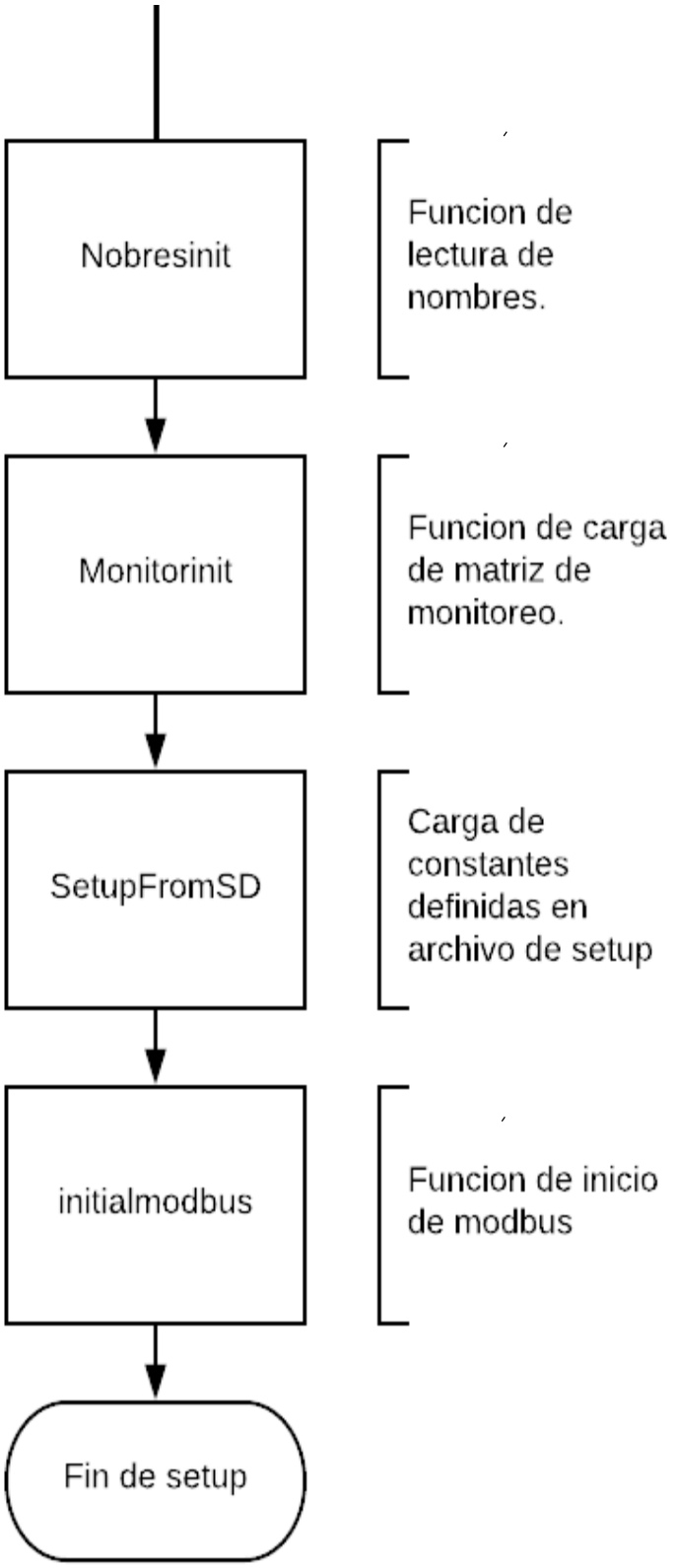


Fig. 5.5.2 Diagrama de Setup 1

Se detallará brevemente las operaciones realizadas durante el bloque conocido como Setup

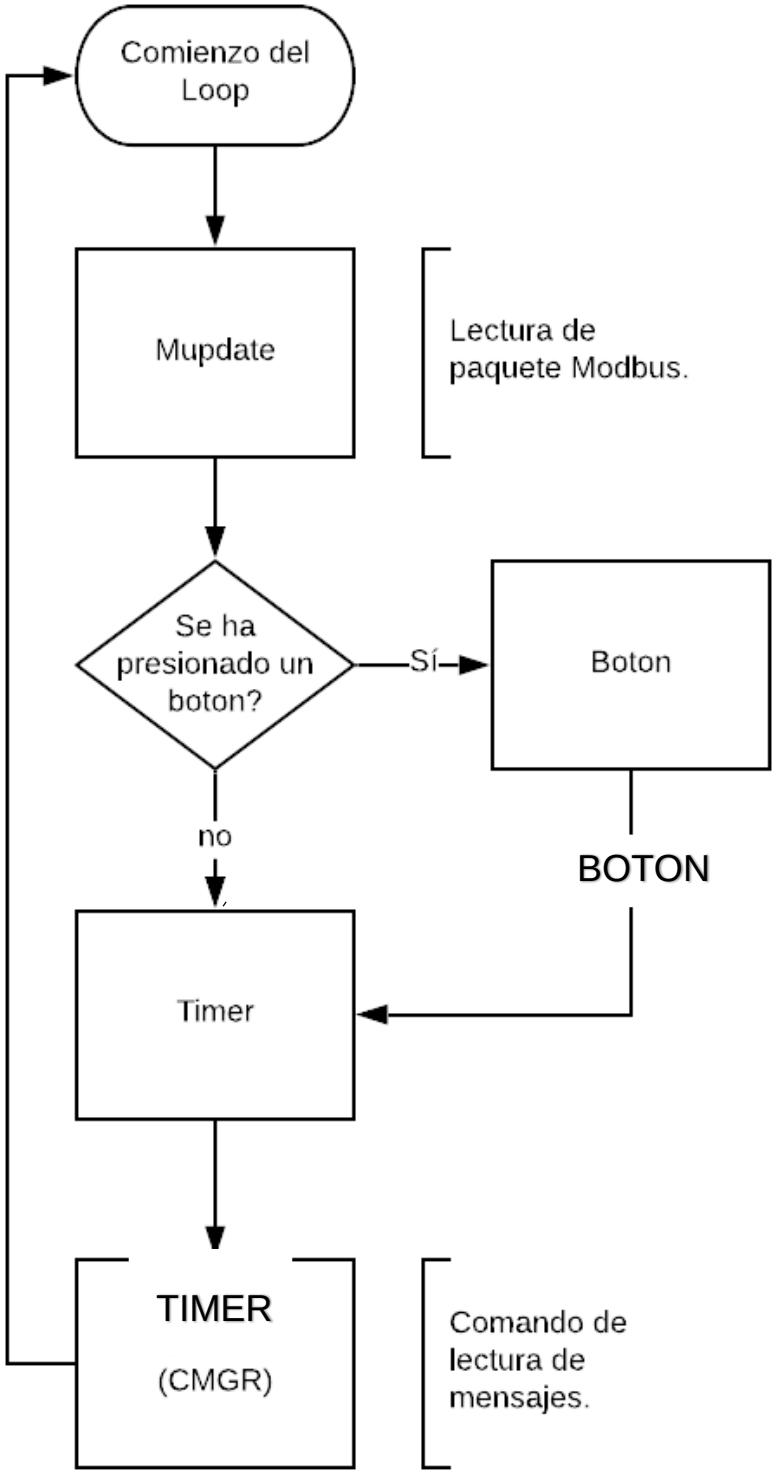
- Se comenzara dando un valor inicial a las variables, para evitar la posibilidad de un inesperado valor causado por memoria residual.
- Luego se dará inicio al display, el cual será la interface principal entre un posible usuario en el lugar.
- El shield de memoria SD se inicia con la finalidad de comenzar a registrar en caso de algún tipo de evento inesperado, como podría ser el fallo de conexión a la red de telefonía móvil.
- Declaración de los paquetes Modbus a leer, los cuales serán las bobinas correspondientes a las 24 entradas y 16 salidas.
- Power: Función cuya tarea es dar inicio al Shield GSM permitiendo la comunicación remota.



- Nobresinit: Función cuya tarea es leer los nombres de entradas y salidas y asignarlos a una variable.
- Monitorinit: Función que crea una matriz que determina si una entrada o salida debe ser monitoreada.
- SetupFromSD: Función cuya tarea es cargar la contraseña del usuario y el número de teléfono de destino.
- Initialmodbus: Función que realiza la lecturas iniciales del PLC mediante Modbus y prepara para futuras lecturas.

Las funciones poseen sus propios disgramas de flujos los cuales se encuentran más adeltante en el capátulo, los códigos de los mismos pueden ser encontrados en el anexo.

Fig.5.5.3 Diagrama de Setup 2



Loop

Código principal del programa el cual se repetirá continuamente.

- Mupdate: Función que envía y recibe un único paquete de datos vía Modbus y actualiza el valor de las entradas Arduino.
- BOTON: Función que opera según el botón presionado realiza acción correspondiente actualizando el display.
- TIMER: Función la cual verifica el tiempo transcurrido de operación y toma acciones en base al mismo, ajustando valores de variables, actualizando la pantalla, etc.
- SendAT: Función de envió de comando AT (de lectura de mensajes), verificando si se ha recibido uno y actuando en función a ello.

Fig.5.5.4 Diagrama de loop

Funciones
Initalmodbus

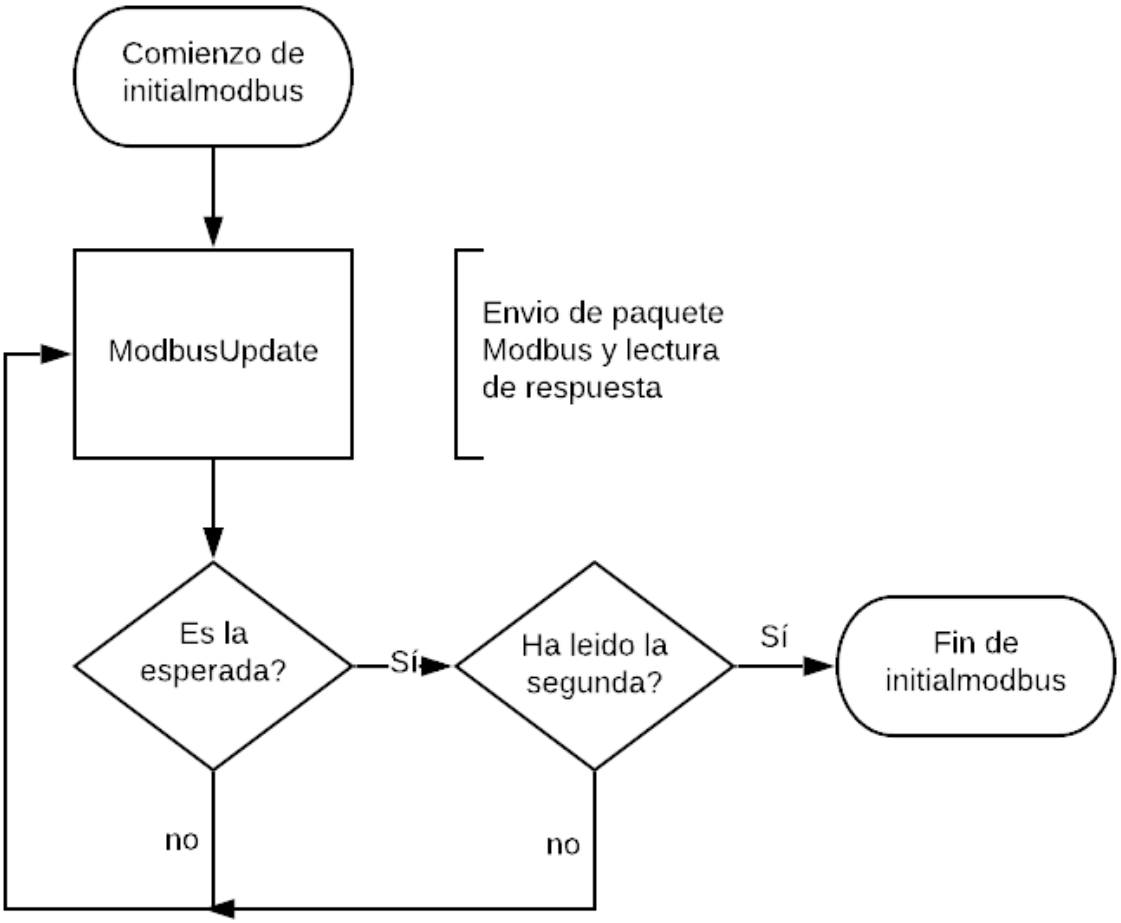


Fig. 5.5.5 Diagrama de Initialmodbus

Esta función cumple una tarea muy importante, la cual es preparar la secuencia para ser leída y asegurar que el próximo paquete sea el primero de las entradas. Hay en total 5 paquetes, 3 de los cuales leen entradas del PLC (I0.0 a I0.7 – I1.0 a I1.7 – I2.0 a I2.7) y 2 los cuales leen salidas (Q0.0 a Q0.7 - Q1.0 a Q1.7), estos 5 paquetes son leídos de manera secuencial en el orden presentado.

Debido a que no se asegura que el primer paquete sea el esperado se recurre a un pequeño artilugio. Las entradas correspondientes a I0.0 – I1.0 – I2.0 deben permanecer siempre en “0”, mientras que las salidas Q0.0 y Q1.0 deben permanecer en “1”. Por lo cual, al leer paquetes de entrada vera que el primer bit será 0 y al leer paquetes de salida el primer bit será 1, de esta forma luego de leer 2 paquetes con el primer bit en “1” (dos salidas) se sabrá que el próximo paquete a ser leído será el primero de entradas. Por otra parte, se puede verificar la existencia de pérdida de paquetes en caso de cambio en alguna de dichas entradas/salidas.

Nombreinit

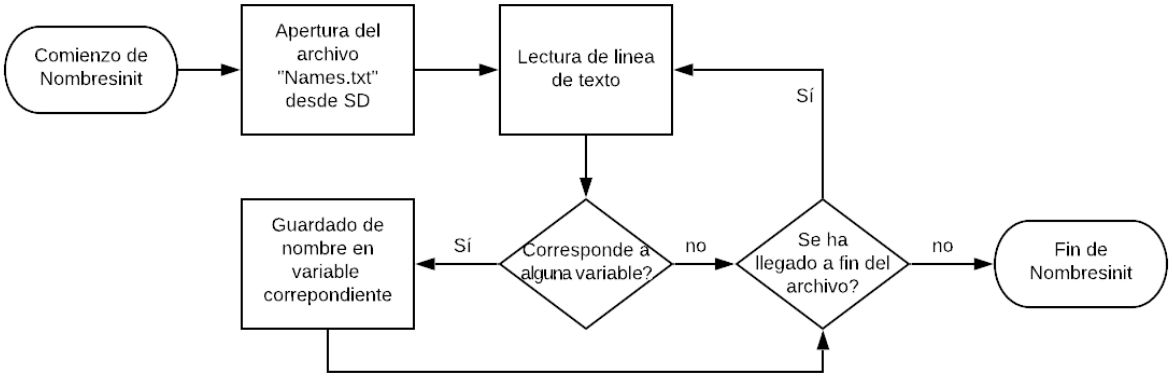


Fig. 5.5.6 Diagrama de Nombreinit

La función principal es cargar las variables correspondientes a los nombres de las entradas/salidas desde el archivo de texto “Names.txt” ubicado en la memoria SD.

Mupdate

Cumple con una de las tareas más importantes del programa, la cual es leer los paquetes provenientes del PLC y decodificarlos para analizar los cambios de entradas/salidas. Según los cambios realizados se verifica si se debe tomar acción y en consecuencia ejecutar la función “action”.

En caso de haber una falla, se comienza a ejecutar un contador y si al pasar 10 minutos se sigue sin obtener ningún paquete de manera exitosa, reporta el fallo vía mensaje de texto. Si luego de haber leído más de 3 fallas se lee exitosamente datos desde el PLC, se ejecuta la función “initiallmodbus” para volver a sincronizar los paquetes leídos con los esperados.

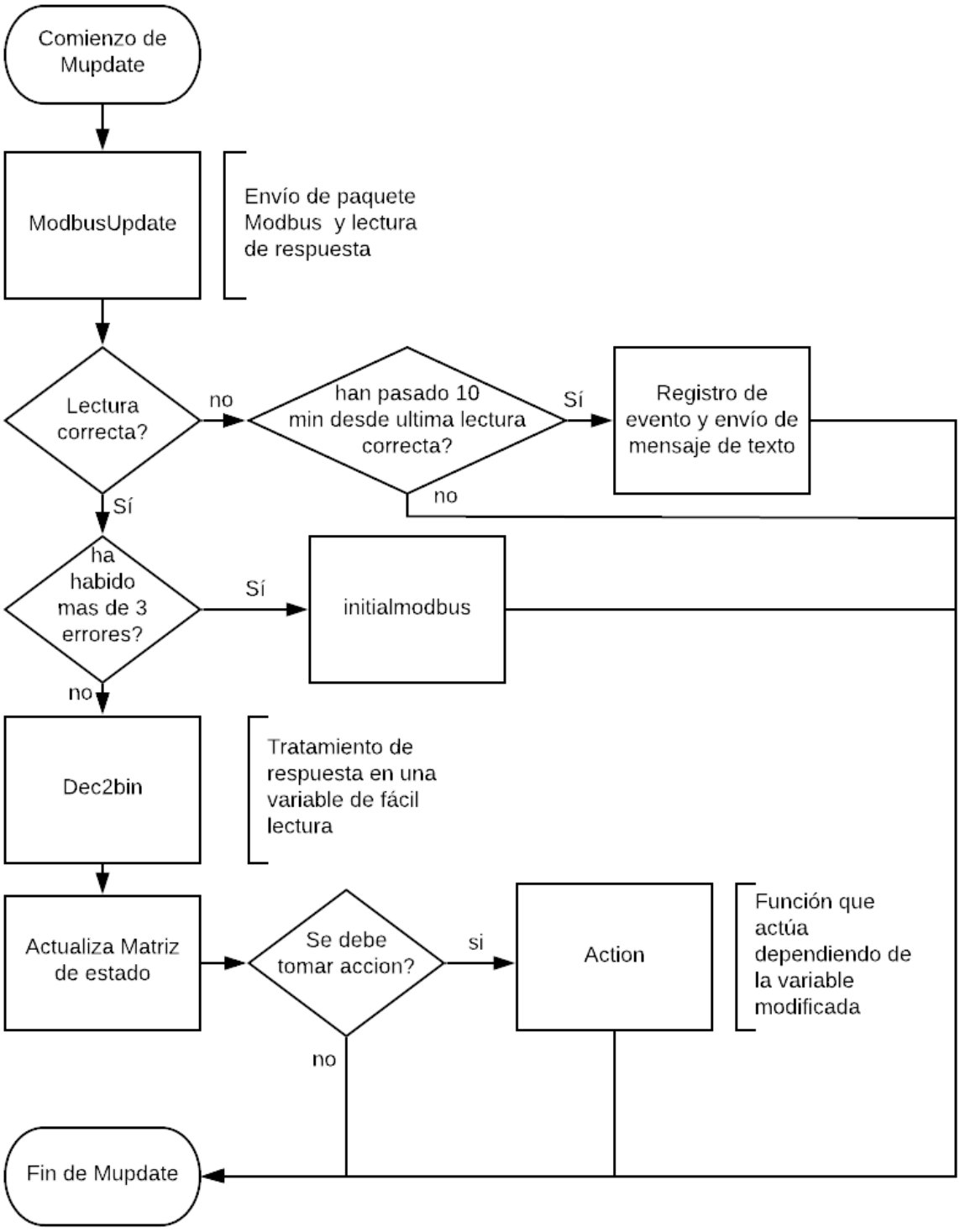


Fig. 5.5.7 Diagrama de Mupdate

Power

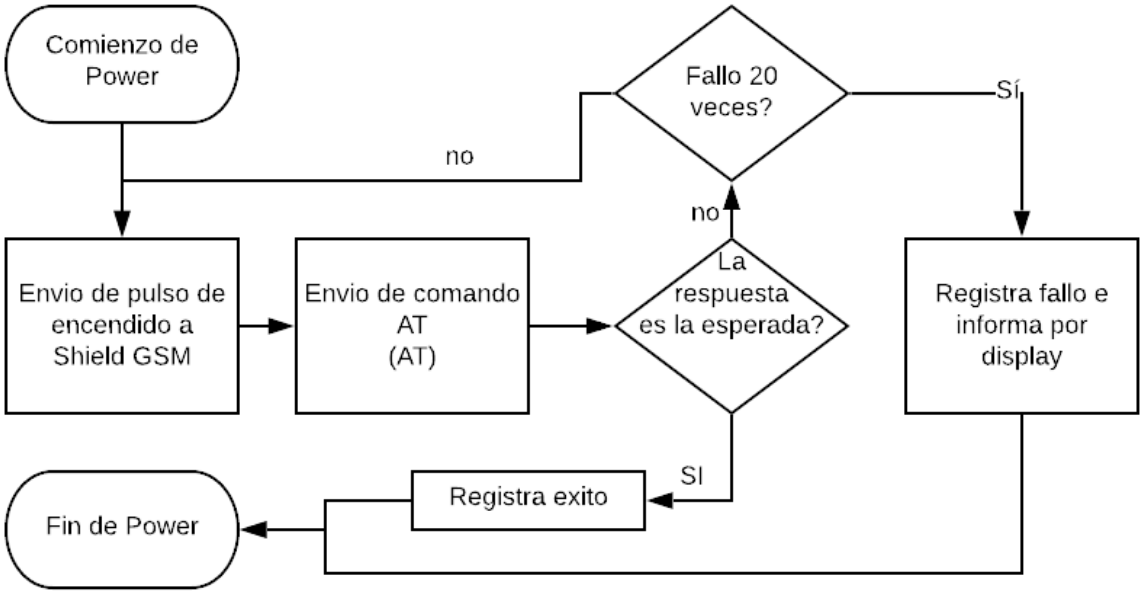


Fig. 5.5.8 Diagrama de Power

Su tarea es encender el shield GMS, verificar la conexión con la red al enviar un comando AT y registrar en la memoria SD el resultado. El comando a enviar es “AT”, en caso de mantener una conexión exitosa devolverá “OK” como respuesta. En caso de fallar luego de veinte intentos (1 segundo por intento) continuará sin la capacidad de monitoreo hasta que se estabilice la señal.

SendAT

Cumple la tarea de controlar el Shield GSM mediante el uso de los comandos AT. Al recibir un mensaje verifica que el mismo posea el código definido por el usuario (Código de administrador), en caso de no tenerlo ignora el mensaje. Las funciones a tener en cuenta son las generadas por el usuario en el archivo correspondiente de la memoria SD y aquella que permite el ingreso manual “-Man:”. (Ver capítulo 6). Finalmente la función devuelve un valor indicando si la respuesta fue la esperada o hubo algún inconveniente.

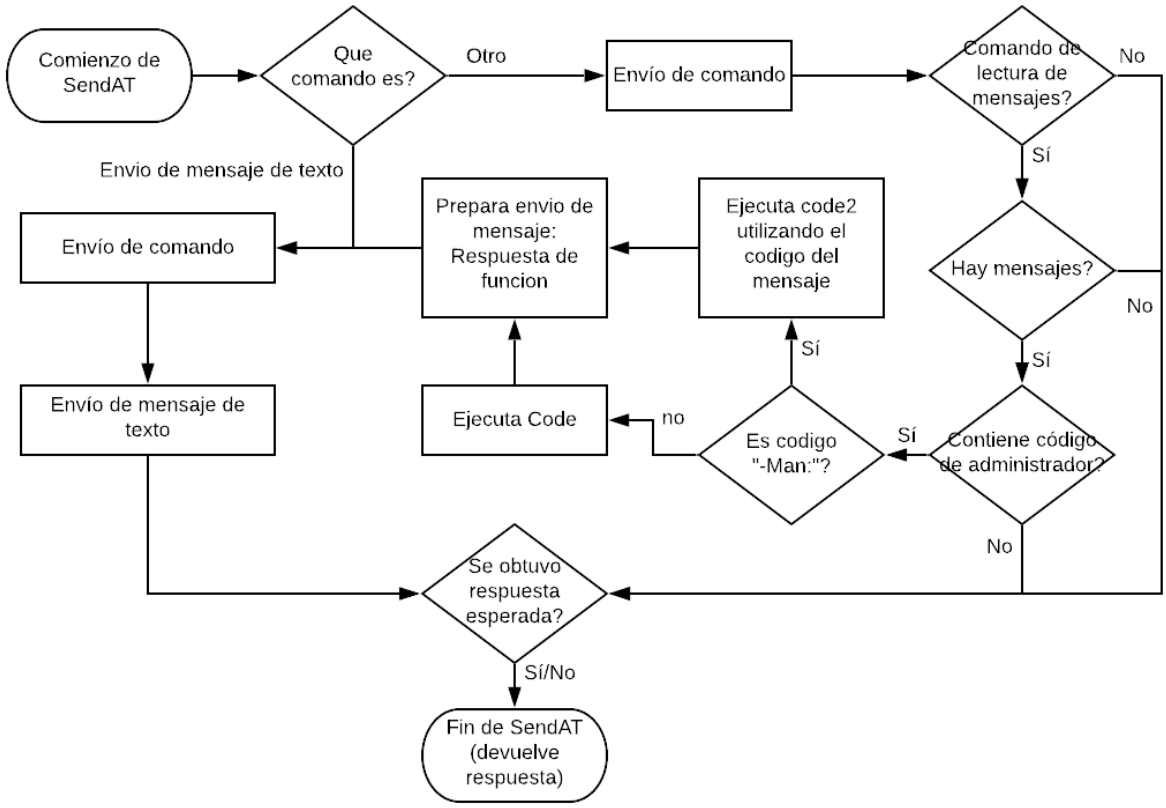


Fig. 5.5.9 Diagrama de SendAt

Code

Su trabajo es buscar la función solicitada dentro de las definidas por el usuario. En caso de encontrarla ejecuta “Code2”, de no encontrarla debe informar que no se ha encontrado.

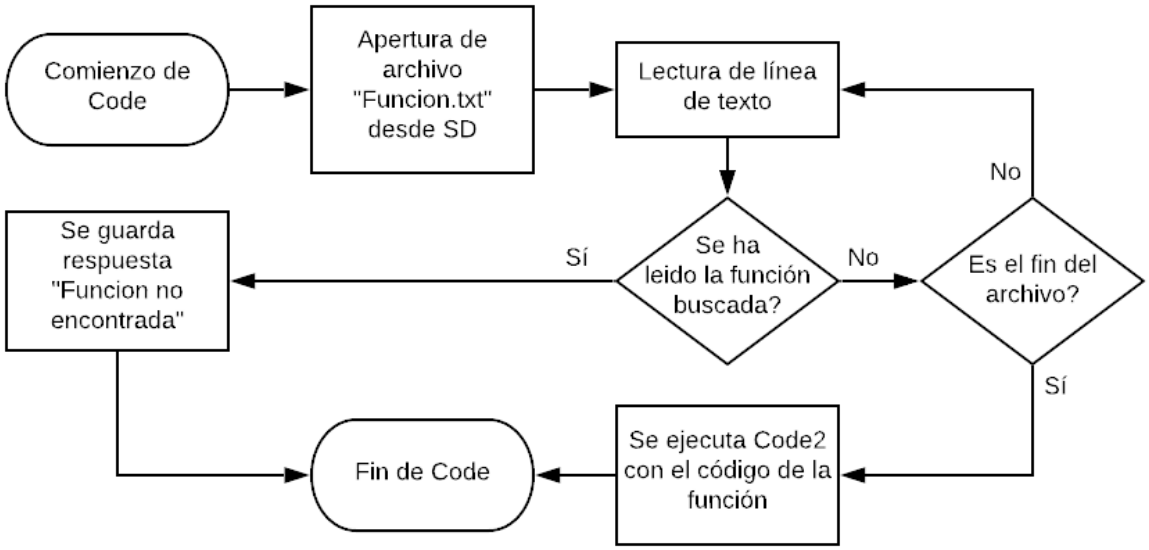


Fig. 5.5.10 Diagrama de Code

Code2

Función cuyo objetivo es buscar las tareas dentro de la función asignada y ejecutarlas. Luego de cumplir con la tarea registrar para luego enviar mensaje de texto con la respuesta. (Ver capítulo 7 para información sobre Tareas y Funciones)

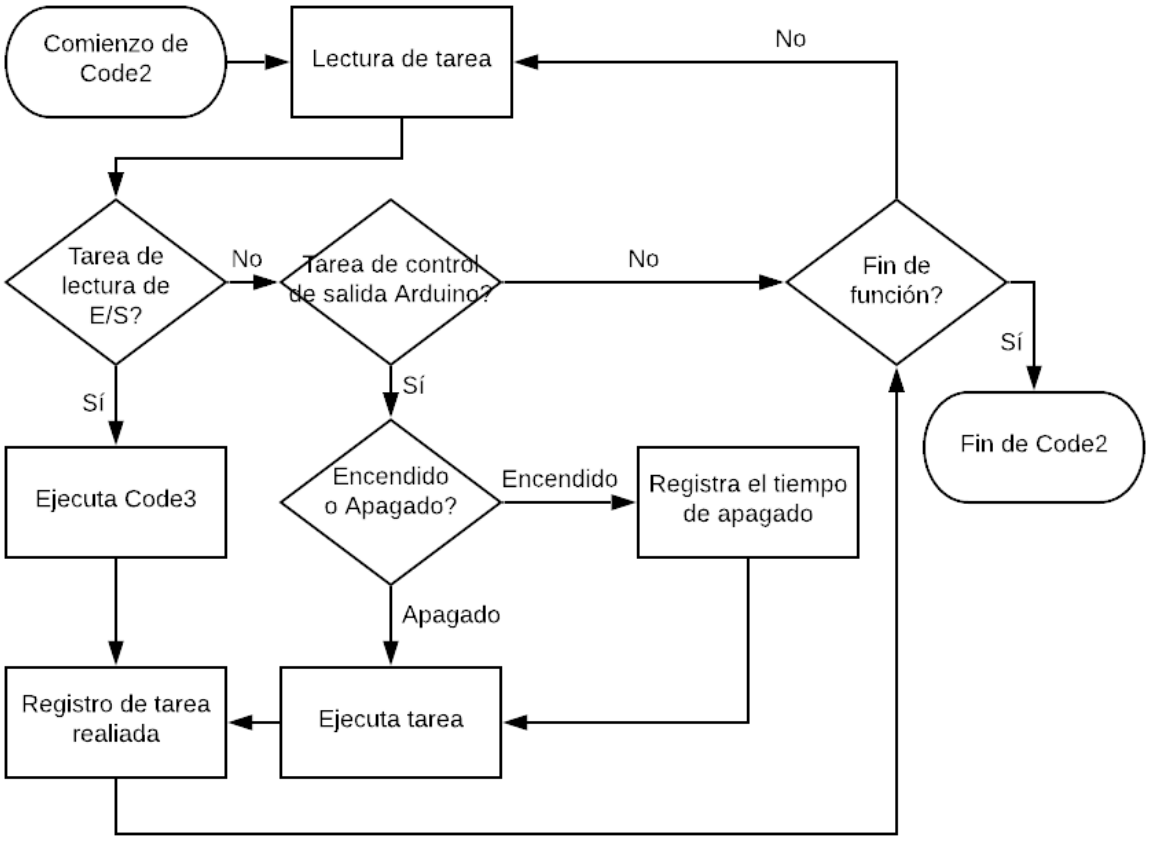


Fig. 5.5.11 Diagrama de Code2

Code3

Busca el estado actual de la entrada o salida requerida y registra lo obtenido para envío de respuesta.

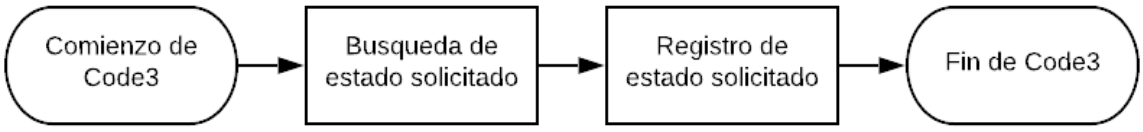


Fig. 5.5.12 Diagrama de Code3

Action

Función la cual toma acciones según los cambios presentados en las entradas del PLC o de Arduino. Las instrucciones a seguir son dictadas según lo configurado por el usuario en el archivo correspondiente dentro de la memoria SD. (Ver Capítulo 6)

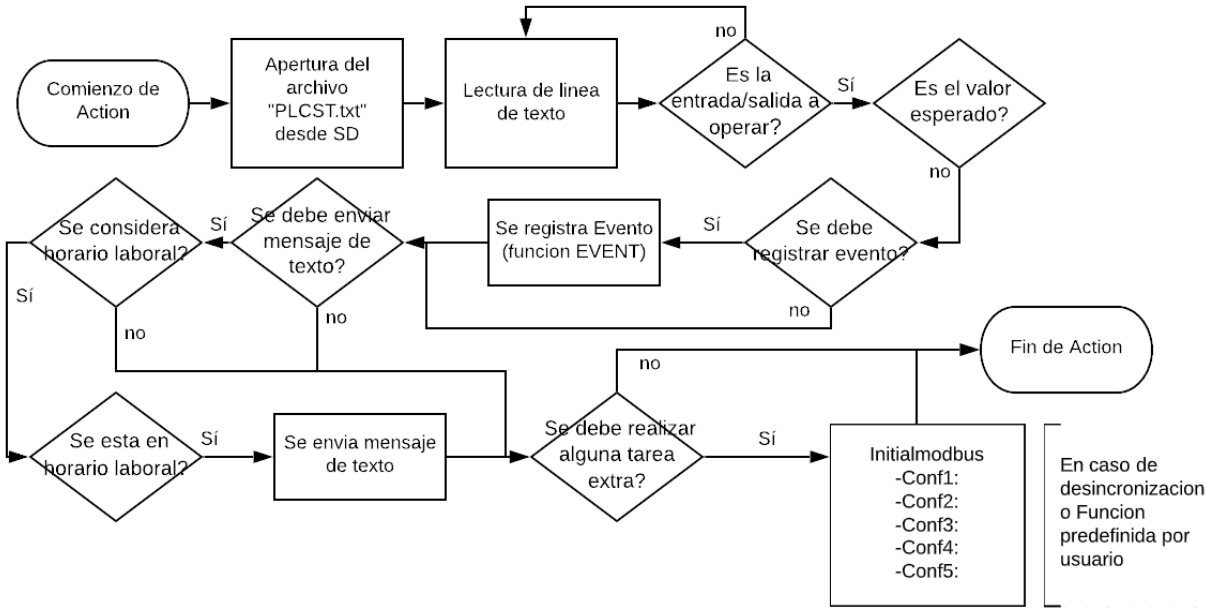


Fig. 5.5.13 Diagrama de Action

Monitorinit

El deber de la función es cargar la matriz de monitoreo, la misma matriz es utilizada en “Mupdate” y define si ante un cambio de alguna entrada (PLC o Arduino) se debe tomar acción o ignorarla.

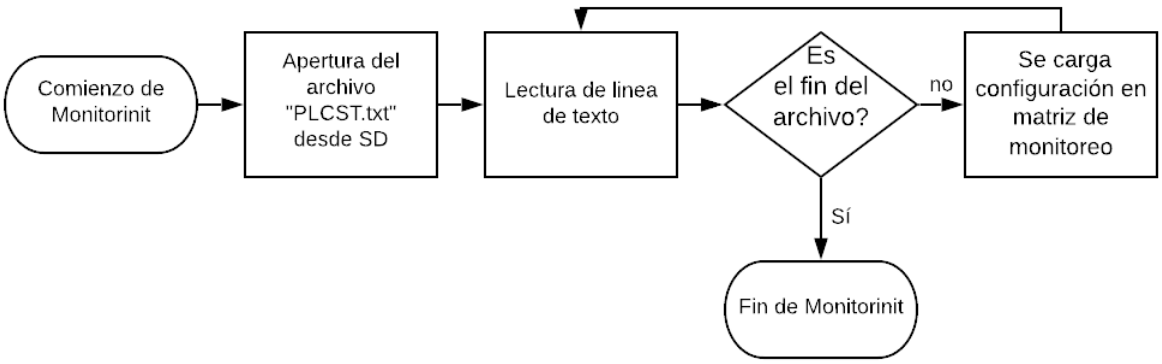


Fig. 5.5.14 Diagrama de Monitorinit

SetupfromSD

Carga la clave de administrador y el número de teléfono designado desde la memoria SD

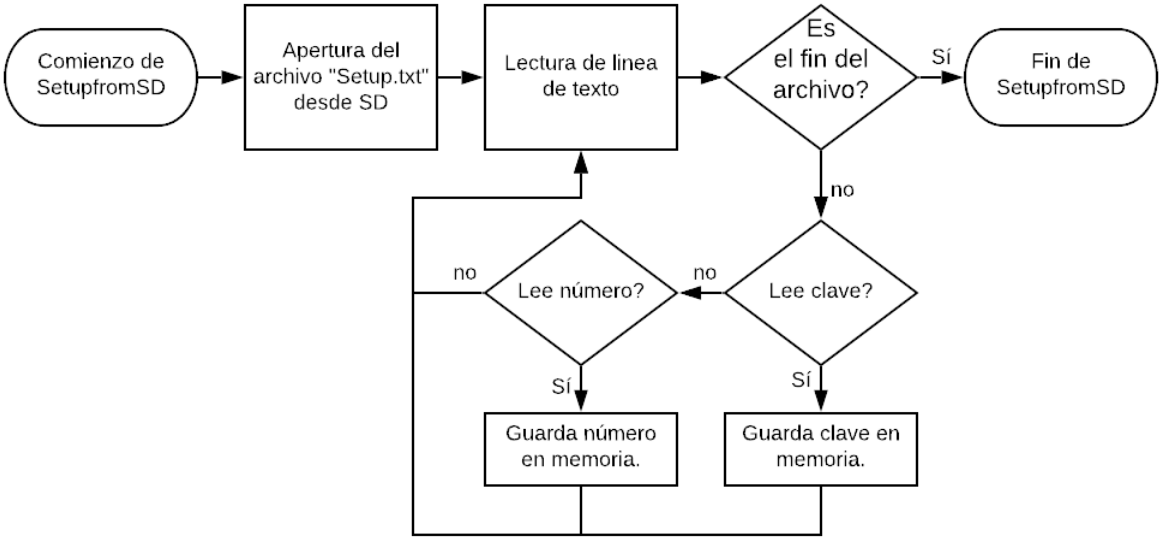


Fig. 5.5.15 Diagrama de SetupfromSD

Event

Cumple la tarea de registrar eventos en el archivo “Event.txt” y actualizar las variables de eventos.

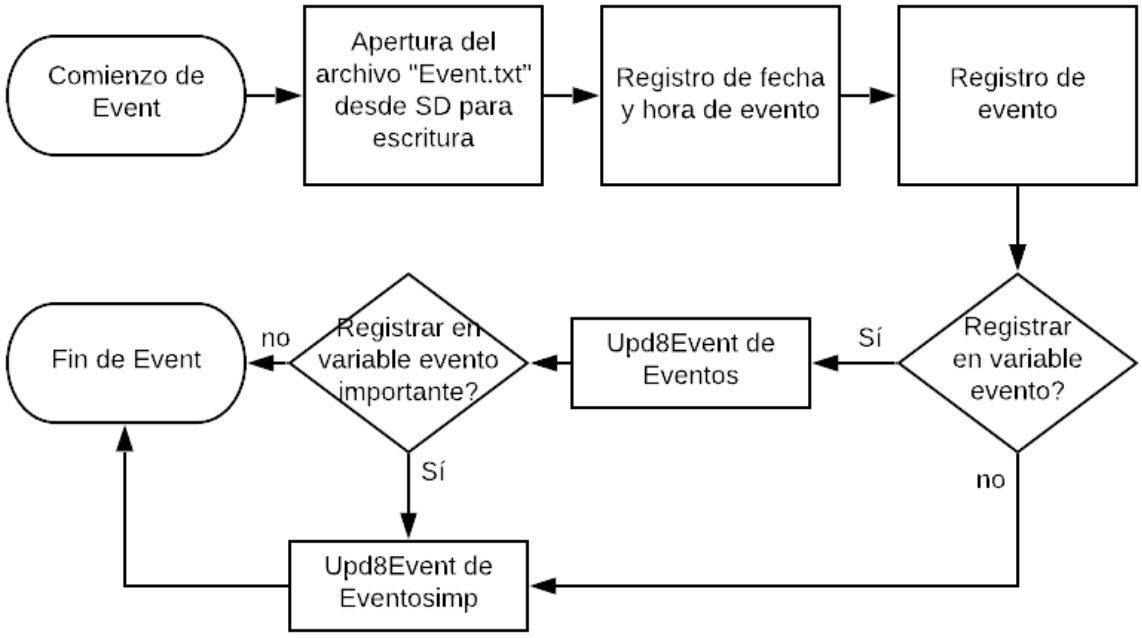


Fig. 5.5.16 Diagrama de Event

Upd8Event

Actualiza las listas de eventos y eventos importantes para que puedan ser visualizadas en el display cuando son requeridas.

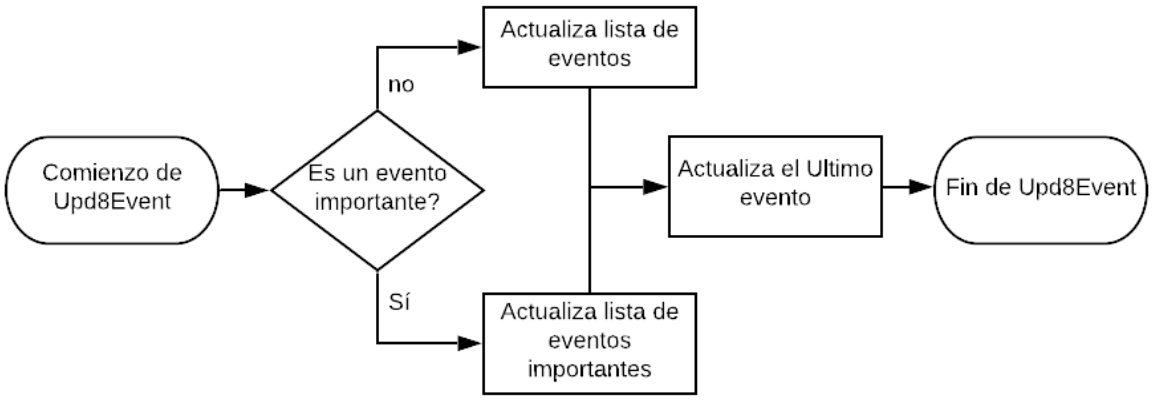


Fig. 5.5.17 Diagrama de Upd8Event

BOTON

Verifica el botón registrado por las interrupciones y actúa en consecuencia. Puede cambiar la posición del cursor en el display si se presiona alguno de los botones direccionales (Arriba / Abajo), o bien cambiar el menú al pulsar el botón de función.

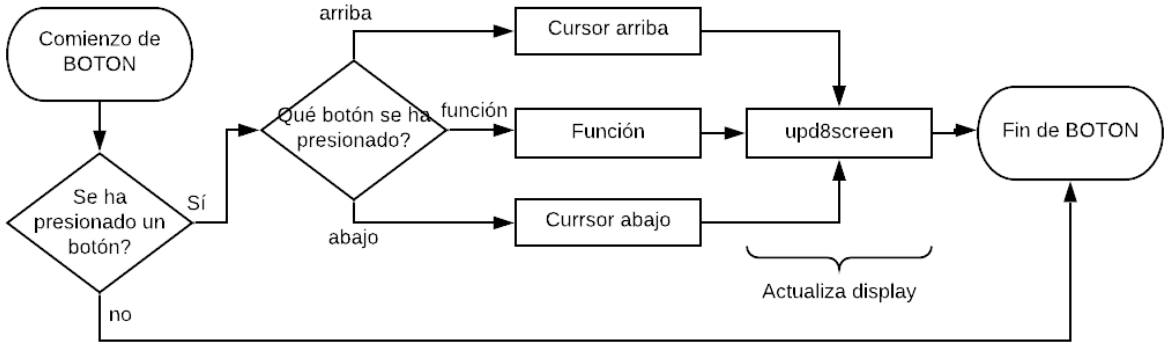


Fig. 5.5.18 Diagrama de BOTON

TIMER

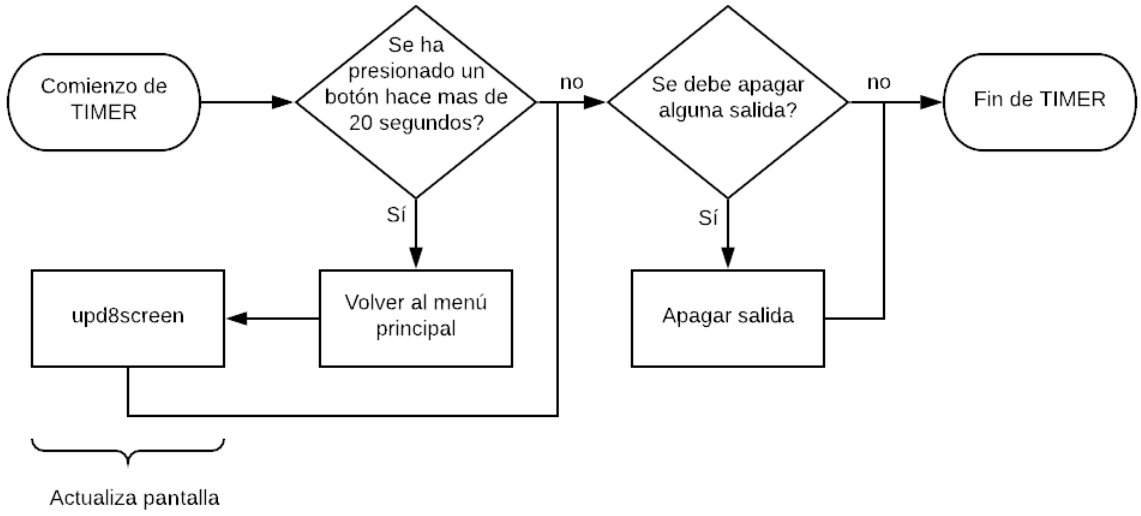


Fig. 5.5.19 Diagrama de TIMER

Cumple tareas relacionadas con temporizadores:

- Verificar los temporizadores de las salidas de Arduino (ver ONF), en caso de que se haya cumplido el tiempo apaga la salida.

- Volver al menú principal en caso de no haberse presionado ningún botón en 20 segundos.

Interrupciones (inter2, inter3 e Inter18)

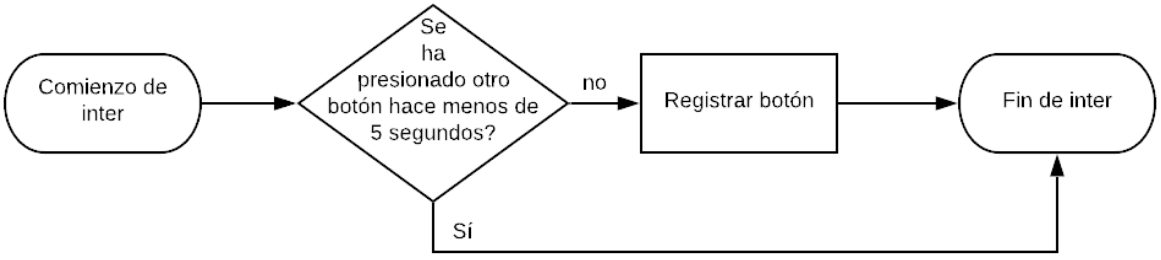


Fig. 5.5.20 Diagrama de interrupciones

Las mismas están definidas para ejecutarse cuando se presionan 3 botones (Arriba, Abajo y Función). Cuando se presiona un botón Arduino ve varias activaciones de la entrada, pues la señal recibida no es un escalón perfecto. Este fenómeno se conoce como efecto rebote y se da en la conmutación de todo tipo de contacto mecánico. Para poder solucionar el problema, luego de registrar una activación no registra otra hasta que pasen al menos 500 milisegundos. El código es análogo para cara una de las entradas definidas.

5.6 Conexiones de Arduino

Las conexiones con otros componentes desde Arduino se realizan siguiendo el siguiente esquema

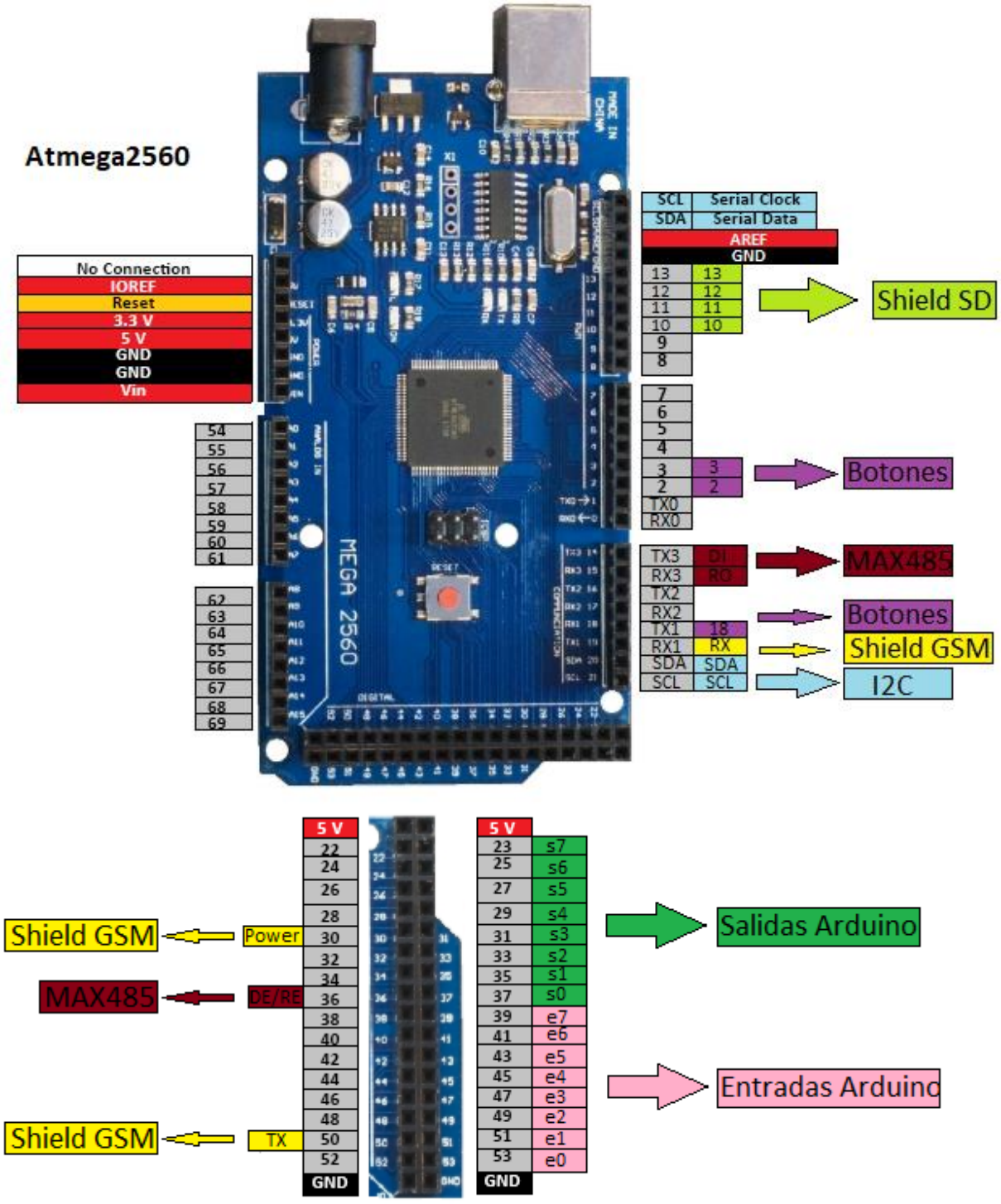


Fig. 5.6.1 Diagrama de Conexiones

NOTA: En caso de no aclararse, todos los componentes son conectados en caso de ser necesario a 5v y GND desde el punto de alimentación. Debido a la sensibilidad de los pines en la placa Arduino, puede ocurrir un cambio de valor generado por corrientes parasitarias. Todos los pines entre el 22 y 53 que sean utilizados son conectados a una resistencia de 1k ohm conectada a gnd para evitar una falsa lectura. En adición se utiliza cable trenzado para reducir nivel de ruido EMI (Electromagnetic Interference) inducido.

Resumen de conexiones.

La conexiones con el Shield SD, se realiza directamente montando el mismo sobre la placa Arduino, por lo cual se conecta en adicion 5v y GND.

Conexiones	
Arduino	Shield SD
13	13
12	12
11	11
10	10
5v	5v
gnd	gnd

Tabla 5.6.2 Arduino/Shield SD

Los botones para el control del display utilizan los pines de Arduino que corresponden a las interrupciones en este modelo. El extremo restante de cada botón es conectado a 5v.

Conexiones	
Arduino	Botones
3	Arriba
2	Función
18	Abajo

Tabla 5.6.3 Arduino/Botones

En la conexión realizada al Max485 el pin 36, que funciona como “habilitador” es conectado en RE y DE simultáneamente. Las salidas correspondientes a RS485(A y B) son conectadas a los pines correspondientes del DB9 de la conexión al PLC.

Conexiones	
Arduino	Max485
TX3	DI
RX3	RO
36	DE
36	RE

Tabla 5.6.4 Arduino/Max485

El shield GSM posee un pin que permite encenderse, en el shield es el pin 9 y se lo conecta al pin 30 de Arduino. Esto no es usualmente necesario, pero considerando que el encendido manual es imposible, se vuelve una necesidad.

Conexiones	
Arduino	Shield GSM
19	RX
50	TX
30	Power (9)

Tabla 5.6.5 Arduino/ShieldGSM

El I2C es conectado directamente al display.

Conexiones	
Arduino	I2C
SDA	SDA
SCL	SCL

Tabla 5.6.6 Arduino/I2C

Las salidas son conectadas a un módulo de 8 relees optoacoplados. Las entradas son conectadas a Optoacopladores 4N35.

“e0” es conectada a un interruptor, el mismo cumple la función de indicar presencia laboral.

Conexiones		
Arduino	Entradas	Salidas
53	e0	s0
51	e1	s1
49	e2	s2
47	e3	s3
45	e4	s4
43	e5	s5
41	e6	s6
39	e7	s7

5.6.7 Arduino/Entradas/Salidas

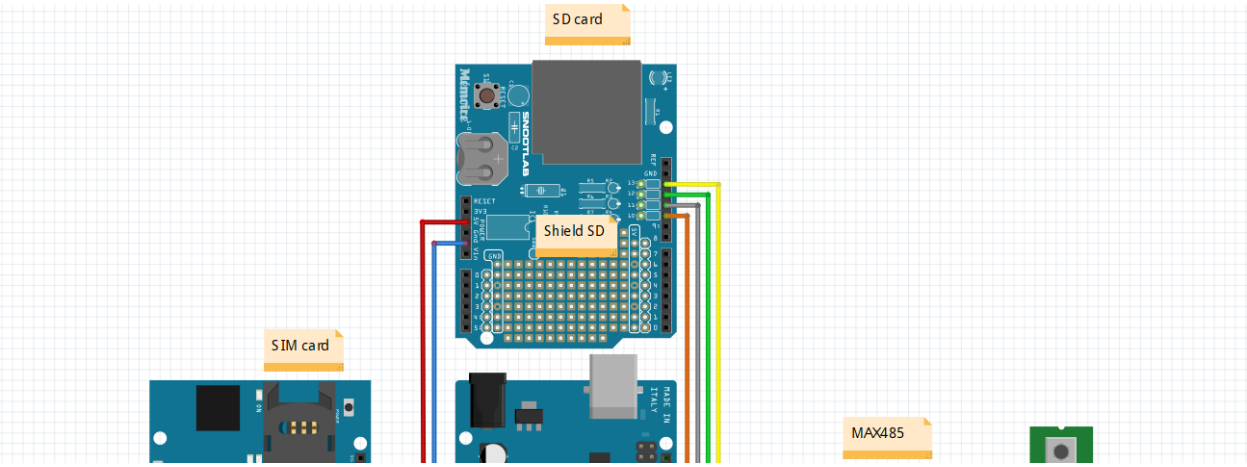


Fig. 5.6.8 Conexiones Parte 1

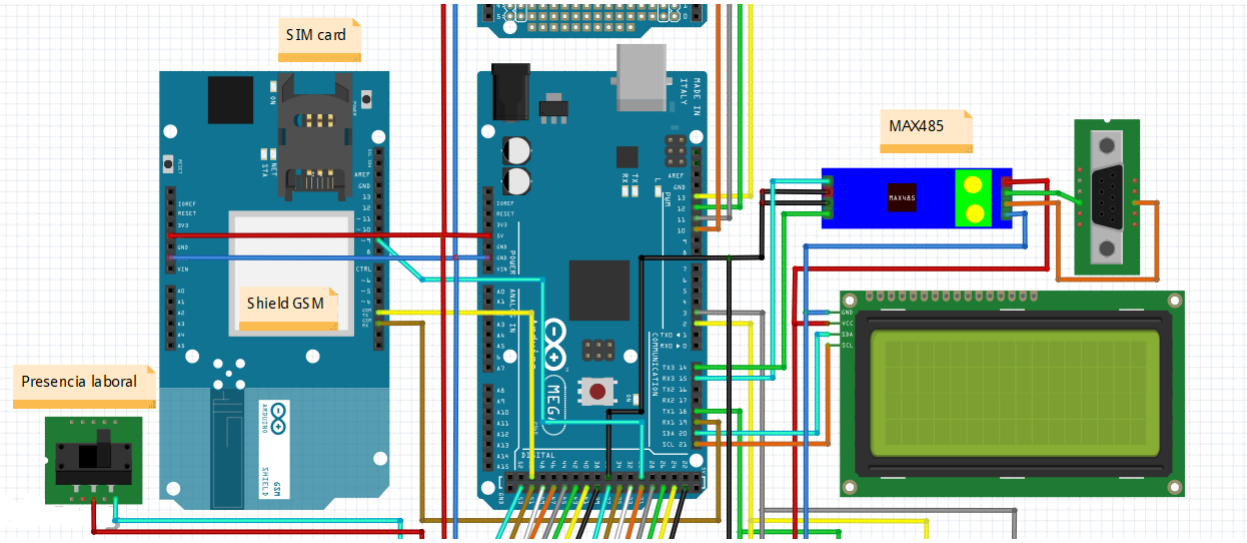


Fig. 5.6.9 Conexiones Parte 2

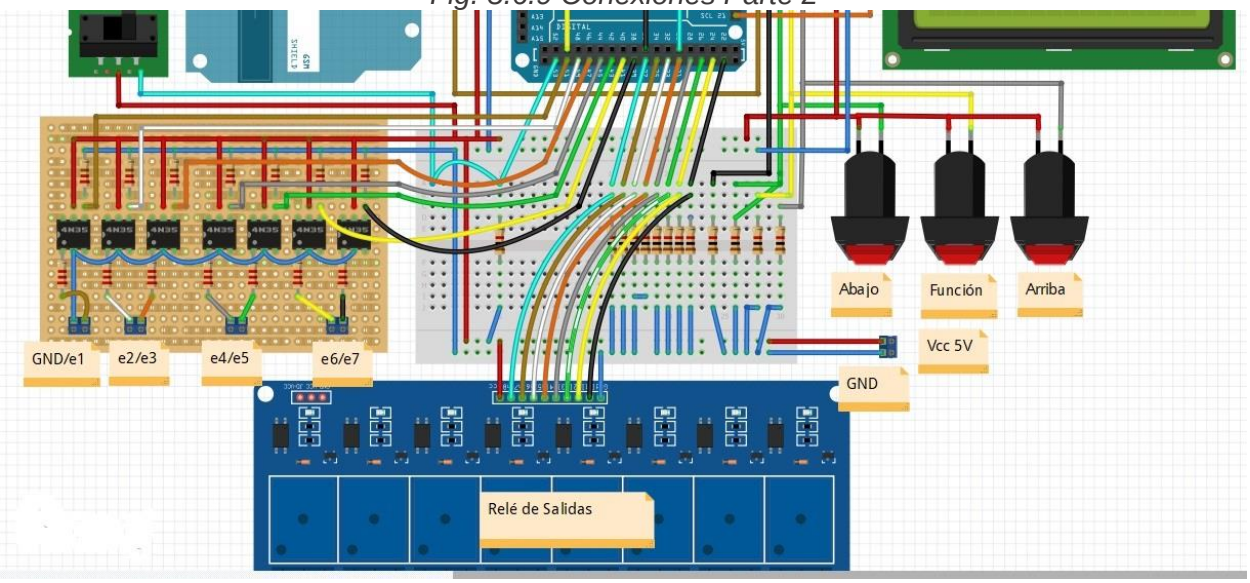


Fig. 5.6.10 Conexiones Parte 3

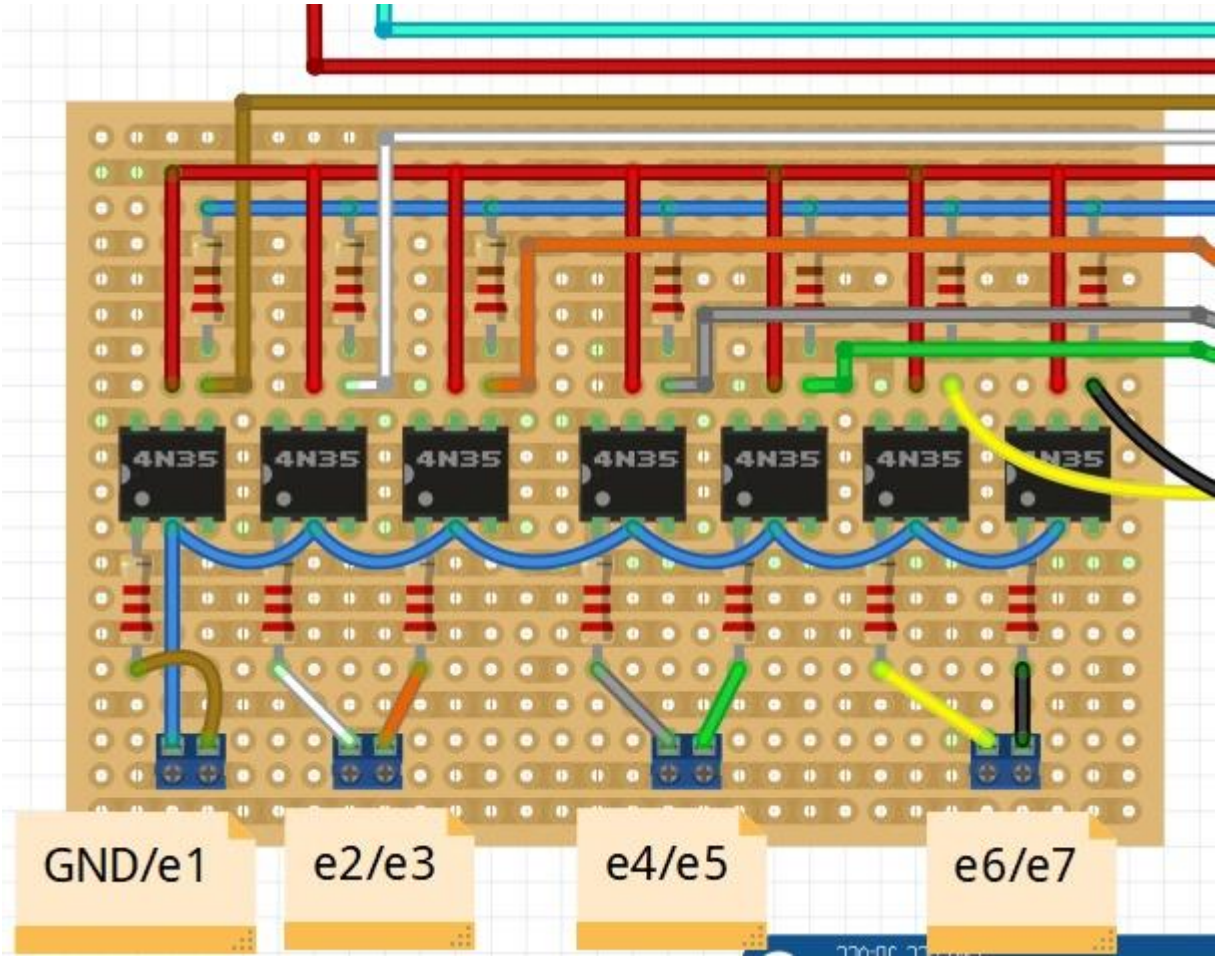


Fig.5.6.11 Detalle circuito de entrada

CAPÍTULO N°6:
MANUAL DE USUARIO

6.1 Objetivos del capítulo

En el capítulo se explicará la configuración y modo de uso del dispositivo de telemetría diseñado en el proyecto. Permitiendo un mayor entendimiento sobre el alcance del mismo y dando al usuario la capacidad de aplicar el aparato en función de sus necesidades.

6.2 Configuración

Existen dos métodos para la configuración del dispositivo, el primero de ellos es por medio del programa de configuración dentro de la memoria SD y el segundo es realizar la modificación manual de los archivos.

6.2.1 Configuración manual

Para poder usar correctamente el dispositivo de monitoreo remoto es necesario configurarlo en función de las necesidades. Para ello se puede acceder a diferentes archivos dentro de la memoria SD, cada uno de ellos cumple diferentes funciones. A continuación se detalla cada uno de ellos:
Nota: la secuencia formada por "*" indica el fin del archivo y asegura una lectura correcta.

Setup.txt: En el mismo se permite configurar la clave de usuario y el número de teléfono designado para el envío de mensajes.

Donde dice "Number" se debe colocar el número de teléfono respetando el prefijo internacional y cogido de área, 54 y 223 en este caso respectivamente (Argentina, Mar del plata).

En "Codigo" se debe colocar el código de administrador. Este código debe estar en cada mensaje enviado al dispositivo, de lo contrario será ignorado.

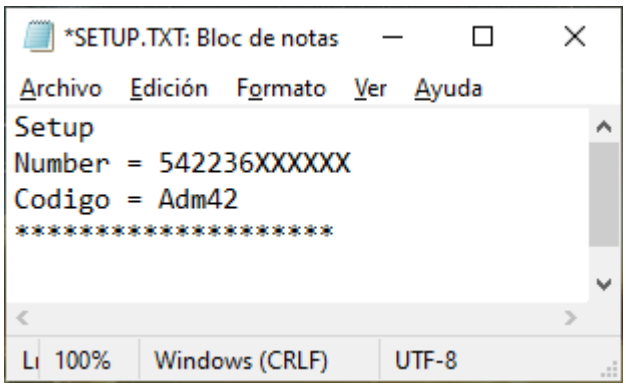


Fig. 6.2.1.1 Setup.txt

Names.txt: En el archivo se pueden colocar los nombres de las Entradas/Salidas para ser mostrados en la pantalla. Se permiten hasta 15 caracteres por nombre, los caracteres excedentes serán ignorados.

Se pueden distinguir cuatro tipos de Entradas/Salidas:

- IX.X: Entradas del PLC
- QX.X: Salidas del PLC
- E0.X: Entradas de Arduino
- S0.X: Salidas de Arduino

EJ1: "I0.1:Horno 1", cuando se acceda al menú de E/S de la pantalla se podrá ver sin problemas el nombre "Horno 1" asignado a la entrada I0.1

EJ2: "Q0.4:Alarma de presencia en el taller", solo se podrá visualizar "Alarma de prese" en la pantalla cuando uno verifique la salida Q0.4.

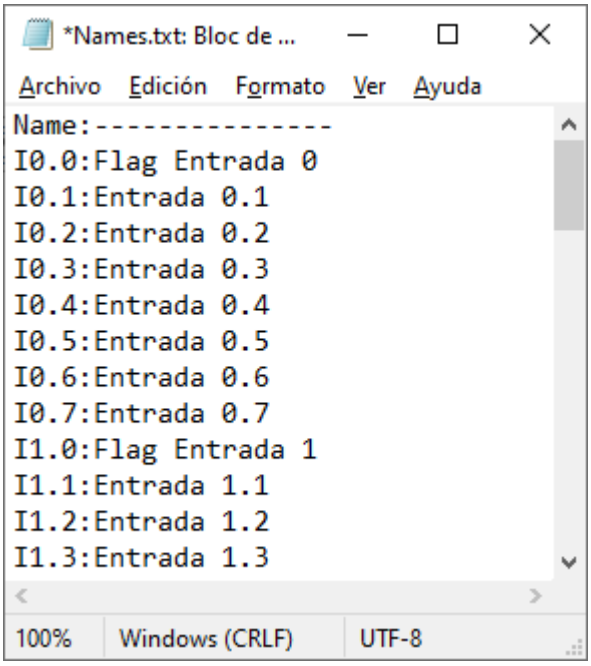


Fig 6.2.1.2 Names.txt

PLCST.txt: El archivo cumple la función de configurar la respuesta ante el cambio en las siguientes 3 opciones:

- IX.X: Entradas PLC
- QX.X: Salidas PLC
- E0.X:Entradas Arduino

La línea de código a ingresar posee el siguiente formato:
YX.X:A,B,C,D,E,F,Texto;

Es necesario tener en cuenta que luego de indicar la Entrada/Salida se debe colocar ":", entre otros parámetros se debe utilizar "," y para finalizar ";".
Donde:

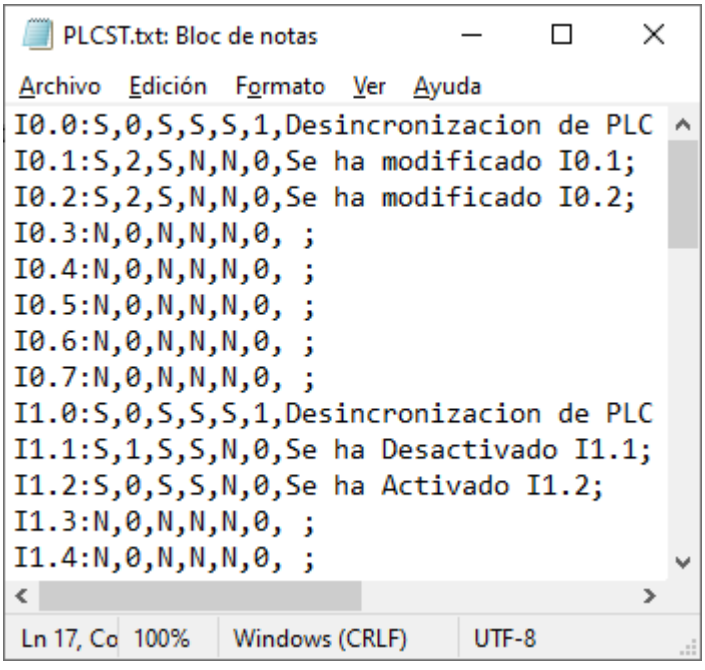


Fig.6.2.1.3 PLCST.txt

Parámetro	Función	Valor esperado de parámetro
YX.X	Dirección de Entrada/Salida a configurar	(Entradas PLC) **I0.0, I0.1, … ,I0.7** **I1.0, I1.1, … ,I1.7** **I2.0, I2.1, … ,I2.7** (Salidas PLC) **Q0.0, Q0.1, … ,Q0.7** **Q1.0, Q1.1, … ,Q1.7** (Entradas Arduino) **E0.0, E0.1, … ,E0.7** (Salidas Arduino) **S0.0, S0.1, … ,S0.7**
A	¿Se debe monitorear?	**S**: Si, en caso de registrarse un cambio en la entrada/salida, actuará según lo configurado. **N**: No, en caso de registrarse un cambio no actuará.
B	¿Valor esperado?	**0**: Normalmente desactivado, en caso de activarse actuará. **1**: Normalmente activado, en caso de desactivarse actuará. **2**: Normalmente estable, en caso de cambio actuará.
C	¿Registrar evento en memoria SD?	**S**: Si, se registrará evento en memoria SD. **N**: No, no se registrará evento en memoria SD. Nota: Los eventos son registrados en el archivo “Event.txt” dentro de la memoria SD.
D	¿Informar evento por mensaje de texto?	**S**: Si, se enviará mensaje de texto. **N**: No, no se enviará mensaje de texto.
E	¿Considerar horario laboral?	**S**: Si, en caso de evento se enviará mensaje de texto solo fuera del horario laboral. **N**: No, se enviará mensaje sin importar el horario laboral.
F	Función extra a realizar	**0**: No realizar ninguna función extra. **1**: Ejecutar “-Conf1”. **2**: Ejecutar “-Conf2”. **3**: Ejecutar “-Conf3”. **4**: Ejecutar “-Conf4”. **5**: Ejecutar “-Conf5”. **6**: Sincronizar, intenta obtener sincronía con el PLC en caso de pérdida de paquetes. se recomienda que las líneas pertenecientes a Entradas/Salidas de control (I0.0, I1.0, I2.0, Q0.0 y Q1.0) mantengan la función 6 dentro las acciones a realizar.

		Notas: "-Conf1:" a "-Conf5:" son funciones definidas por el usuario en el archivo "Funcion.txt". "Sincronizar"
Texto	Texto a enviar por mensaje de texto y/o registrar en memoria SD.	*Notas: Se permiten hasta 60 caracteres, el texto extra será ignorado.*

Tabla 6.2.1.4 PLCST.txt

EJ1: "I2.1:S,2,S,N,N,0,Se ha modificado el valor de I2.1;" Significa entonces:

Parámetro	Valor dado	Significado
YX.X	I2.1	Entrada PLC I2.1
A	S	Debe ser monitoreada
B	2	Actuar al cambiar de estado
C	S	Registrar evento en memoria SD
D	N	No enviar mensaje de texto
E	N	No considerar horario laboral
F	0	No ejecutar acción extra
Texto	Se ha modificado el valor de I2.1	Texto a registrar en Eventos

Tabla 6.2.1.5 PLCST.txt Ejemplo 1

EJ2: "E0.1:S,0,S,S,S,0,Activacion de entrada Arduino 1;" Significa entonces:

Parámetro	Valor dado	Significado
YX.X	E0.1	Entrada Arduino E0.1
A	S	Debe ser monitoreada
B	0	Actuar al activar
C	S	Registrar evento en memoria SD
D	S	Enviar mensaje de texto
E	S	Considerar horario laboral
F	0	No ejecutar acción extra
Texto	Activación de entrada Arduino 1	Texto a registrar en Eventos

Tabla 6.2.1.6 PLCST.txt Ejemplo 2

EJ3 "E0.4:S,1,S,N,N,2, Se ejecuta –Conf2:;" Significa entonces:

Parámetro	Valor dado	Significado
YX.X	E0.4	Entrada Arduino E0.4
A	S	Debe ser monitoreada
B	1	Actuar al desactivar
C	S	Registrar evento en memoria SD
D	N	No enviar mensaje de texto
E	N	No considerar horario laboral
F	2	Ejecutar acción extra "-Conf2"
Texto	**Se ejecuta –Conf2:"**	Texto a registrar en Eventos

Tabla 6.2.1.7 PLCST.txt Ejemplo 3

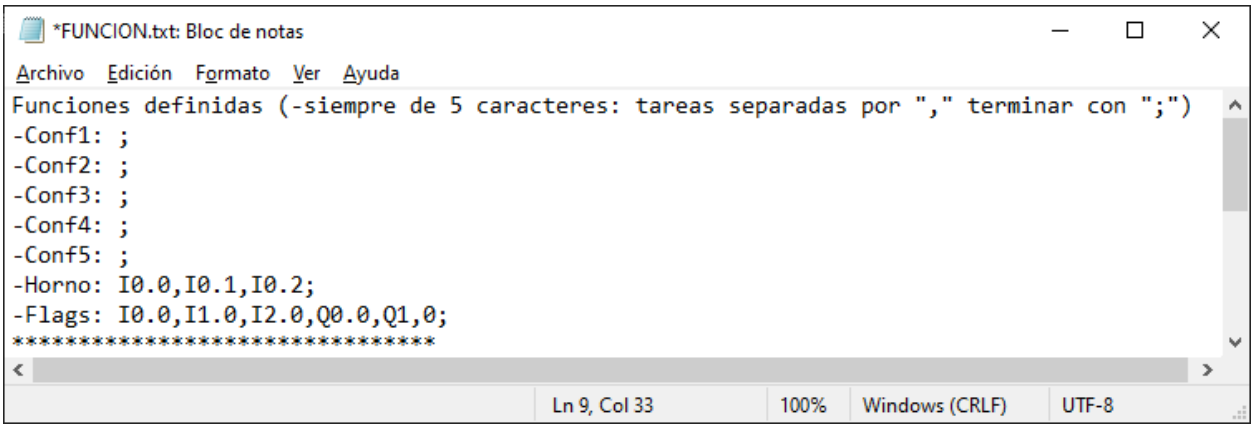


Fig. 6.2.1.8 Funcion.txt

Funcion.txt: En el archivo se registran las funciones personalizadas por el usuario. De ese modo, se podrá realizar una serie de tareas con el ingreso de unos pocos caracteres.
Cada función debe cumplir con las siguientes normas:

- Comenzar con "-"
- Tener un nombre de exactamente cinco caracteres EJ: Horno, Luces, Quema, etc.
- Un ":" para determinar el comienzo de las tareas
- Una o más tareas, en caso de haber más de una deben estar separadas por ","
- Finalizar la secuencia con ";"

Tarea	Función	Parámetros	Valor esperado de parámetro	Ejemplo
Lectura	indicando la dirección de la E/S se obtendrá la información de su Estado y nombre asignado por el usuario en la configuración	**YX.X**	(Entradas PLC) **I0.0, I0.1, … ,I0.7** **I1.0, I1.1, … ,I1.7** **I2.0, I2.1, … ,I2.7** (Salidas PLC) **Q0.0, Q0.1, … ,Q0.7** **Q1.0, Q1.1, … ,Q1.7** (Entradas Arduino) **E0.0, E0.1, … ,E0.7** (Salidas Arduino) **S0.0, S0.1, … ,S0.7**	**I0.2**: Devuelve el estado de entrada del PLC I0.2 **Q1.3**: Devuelve el estado de la salida del PLC Q1.3 **E0.7**: Devuelve el estado de la entrada de Arduino E0.7 **S0.1**: Devuelve el estado de la salida de Arduino S0.1

Apagado	Desactiva la salida de Arduino, en caso de ya encontrarse en tal estado no ocurre nada.	**S-X**	**X**: Número de salida Arduino (0 - 7).	**S-2**: Apaga la salida S0.2 **S-7**: Apaga la salida S0.7
Encendido	Activa la salida de Arduino por un tiempo determinado, en caso de ya encontrarse en tal estado se actualiza el temporizador.	**S+X.YYYY**	**X**: Número de salida Arduino (0 - 7). **YYYY**: Tiempo en minutos con máximo de 1500 minutos, 0 es por tiempo indeterminado. (En caso de ingresar un valor superior, se considera 1500)	**S+2.60**: Activa la salida S0.2 por 60 minutos. **S+3.0**: Activa la salida S0.3 por tiempo indefinido.

Tabla 6.2.1.9 Tareas

Combinación: Una función puede realizar varias tareas, para ello simplemente se las debe separar utilizando "," entre ellas.

EJ: I0.1,I0.2,E0.5,S+2.7,S-1 - Se obtiene información del estado de **I0.1**, **I0.2**, **E0.5**, se enciende **S0.2** por **7** minutos y se apaga **S0.1**

Ejemplos de funciones:

-Horno:I0.1,I0.2,I0.3;
-Luces:S+1.10;
-Reset:S-1,S-2,S-3,S+4.2;

Las funciones denominadas "-ConfX:" son funciones a definir por el usuario que pueden ser ejecutadas en respuesta a un cambio de alguna entrada/salida. (Ver PLCST.txt)

6.2.2 Programa de configuración

Se ha creado un programa utilizando el software LabVIEW con la finalidad de facilitar al usuario la tarea de programar el dispositivo. El mismo se coloca dentro de la memoria SD junto a los archivos de configuración.

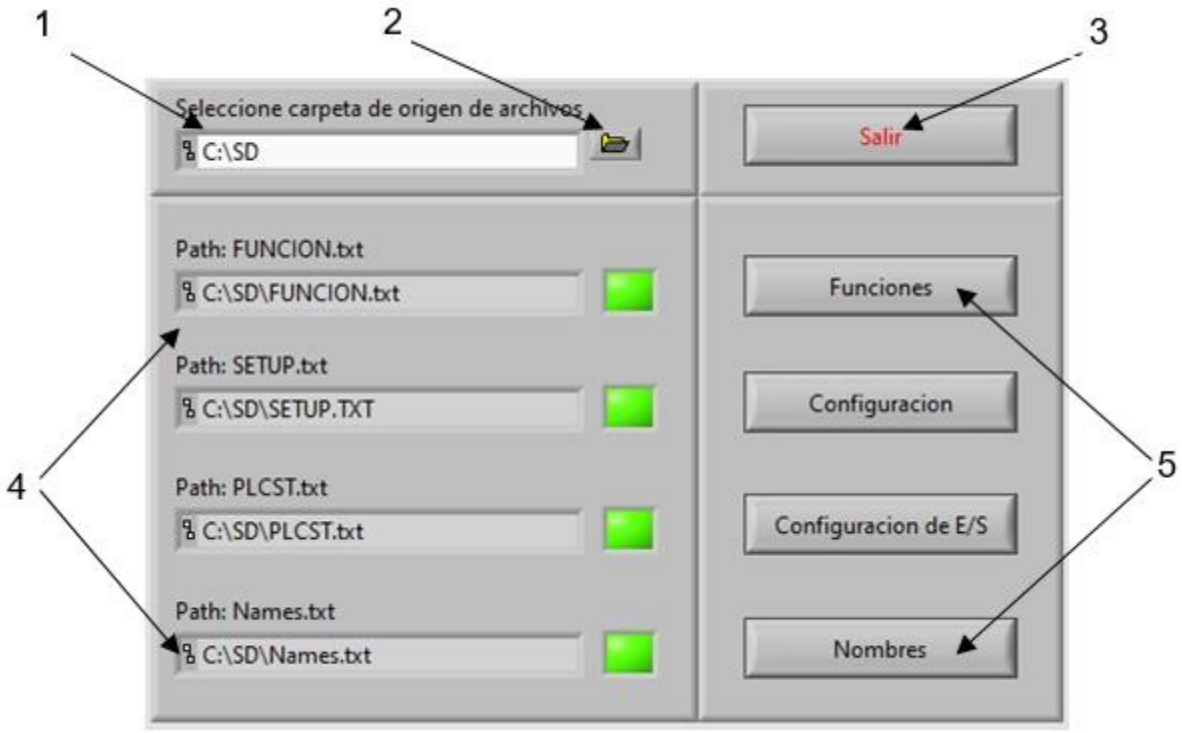


Fig.6.2.2.1 Ventana principal

La pantalla principal permite el acceso a diferentes ventanas para realizar la configuración.

1. Dirección de archivos de texto de configuración.
2. Botón de acceso a búsqueda de dirección.
3. Botón de cierre del programa.
4. Dirección de los archivos de configuración. Los LEDs ubicados a la derecha indicaran si los archivos son correctos iluminándose en verde, en caso contrario se tornaran rojos.
5. Botones de acceso la configuración de los archivos de texto.

Funciones

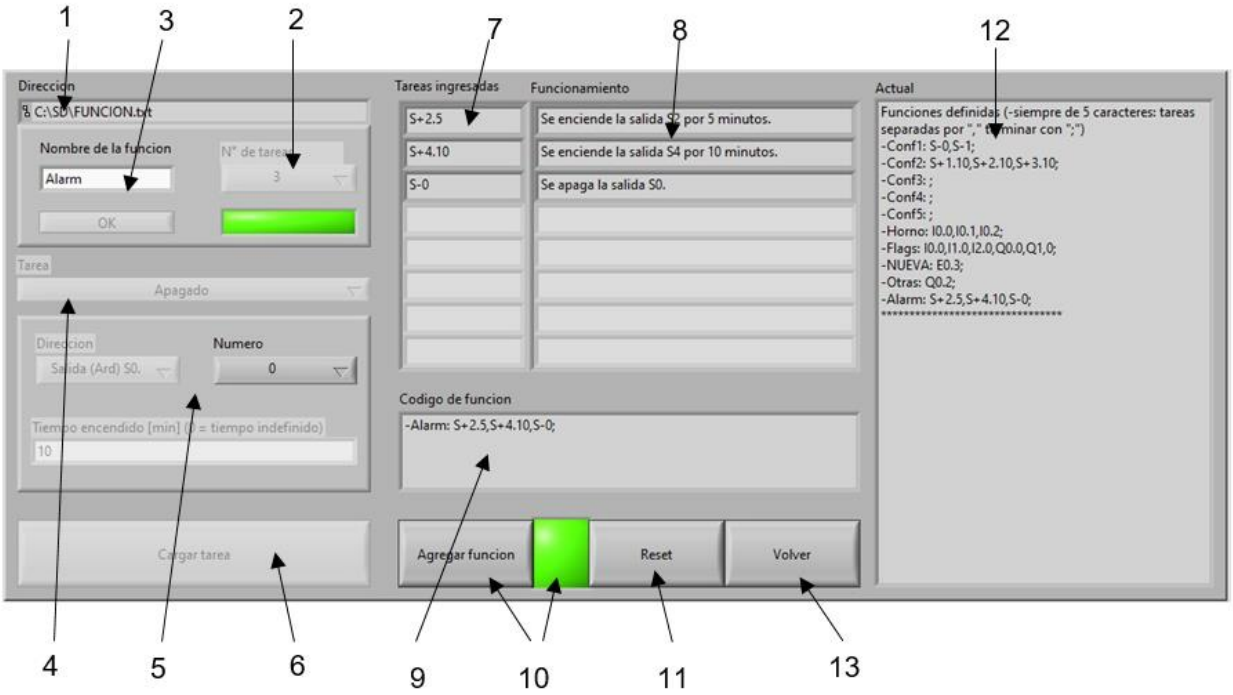


Fig. 6.2.2.2 Ventana funciones

Ventana que permite la visualización de las funciones existentes y el agregado de nuevas funciones.

1. Dirección del archivo “FUNCION.txt”.
2. Número de tareas a definir dentro de la función, se pueden asignar hasta 8 tareas.
3. Nombre de la función a definir. Se debe introducir exactamente 5 caracteres, en caso de no introducirse esa cantidad al presionar el botón “OK” el LED se tornara rojo indicando error. En caso de introducir el nombre de una función ya definida esta se sobrescribirá.
4. Selector de tarea, esta puede ser “Lectura”, “Encendido” o “Apagado”
5. Opciones de tarea. Se puede definir la dirección solo para “Lectura” (para las otras opciones se asigna automáticamente S0.). El número varía de 0 a 7. El “Tiempo de encendido” solo se habilitara en caso de seleccionar “Encendido”.
6. Botón de carga de tarea.
7. Se agregaran los códigos de las tareas cargadas.
8. Se mostrara una explicación coloquial de cada tarea ingresada.
9. Permite visualizar el código de la función a que se esta ingresando.
10. Botón de carga de función. Si se ha cargado exitosamente el LED se iluminara en verde, en caso contrario se tornara rojo.
11. Botón de reset, elimina lo ingresado y se prepara para poder ingresar una nueva función.
12. Visualización del archivo “FUNCION.txt”.
13. El botón volver cierra la ventana permitiendo volver a la ventana principal.

Configuración

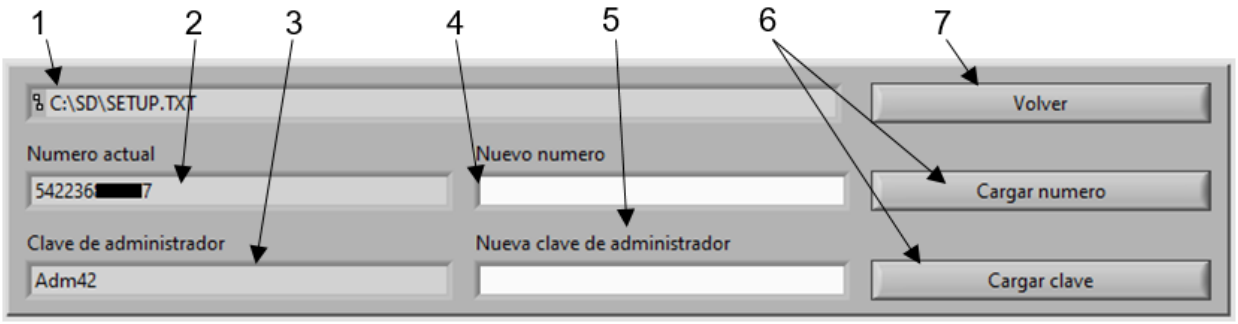


Fig.6.2.2.3 Ventana configuración

Ventana con la finalidad de visualizar y configurar el número de teléfono al cual se enviarán los mensajes de texto y la clave de administrador, la cual debe estar en cada mensaje para que sea considerado por el dispositivo.

1. Dirección del archivo “SETUP.TXT”.
2. Número de teléfono actual.
3. Clave de administrador actual.
4. Nuevo número a configurar.
5. Nueva clave a configurar.
6. Botones de actualización de número y clave.
7. El botón volver cierra la ventana y vuelve a la ventana principal.

Configuración E/S

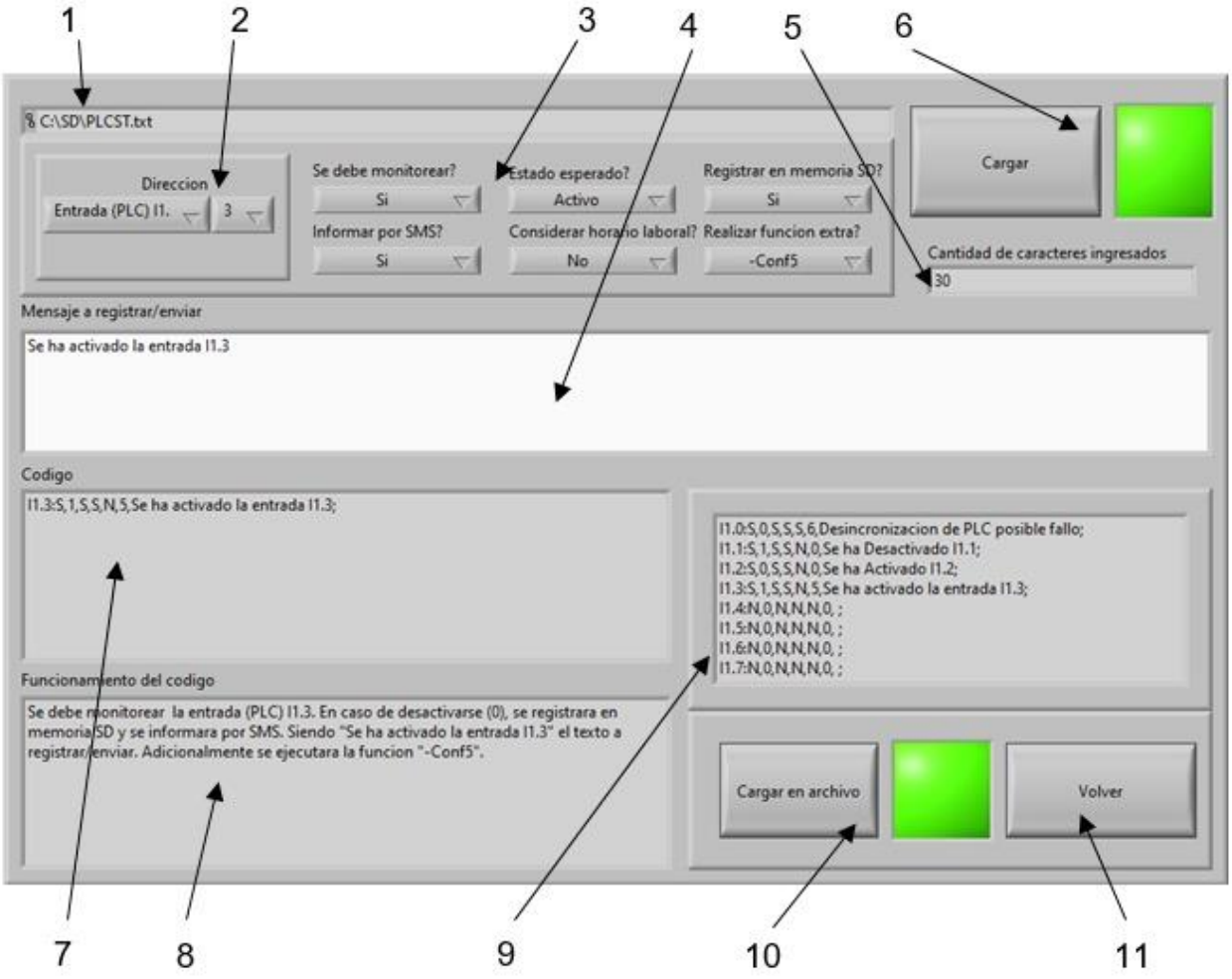


Fig. 6.2.2.4 Ventana configuración E/S

Se permite la configuración de como actuara el dispositivo ante la variación del valor de alguna entrada o salida (PLC o Arduino).

1. Dirección del archivo “PLCST.txt”.
2. Dirección de la entrada o salida a configurar.
3. Opciones de configuración de la entrada o salida, definiendo bajo que situación deberá actuar el dispositivo y como debe actuar ante tal evento.
4. Mensaje a registrar en la memoria SD y/o enviar por SMS.
5. Número de caracteres ingresados en “4.”, el máximo permitido es 60.
6. Botón de carga, en caso de algún error, el LED se tornara rojo.
7. Código de la función definida por el usuario.
8. Explicación coloquial del código 7.
9. Visualización de la configuración de entradas/salidas. Las mismas se visualizan de manera agrupada según la dirección seleccionada (EJ. Si selecciona en dirección “I2.”, se mostraran desde I2.0 hasta I2.7).
10. Botón de carga de código ingresado. El LED se iluminara en verde si se ha realizado la tarea con éxito y rojo en caso de fracaso.

11. El botón volver cierra la ventana y retorna a la ventana principal.

Nombres

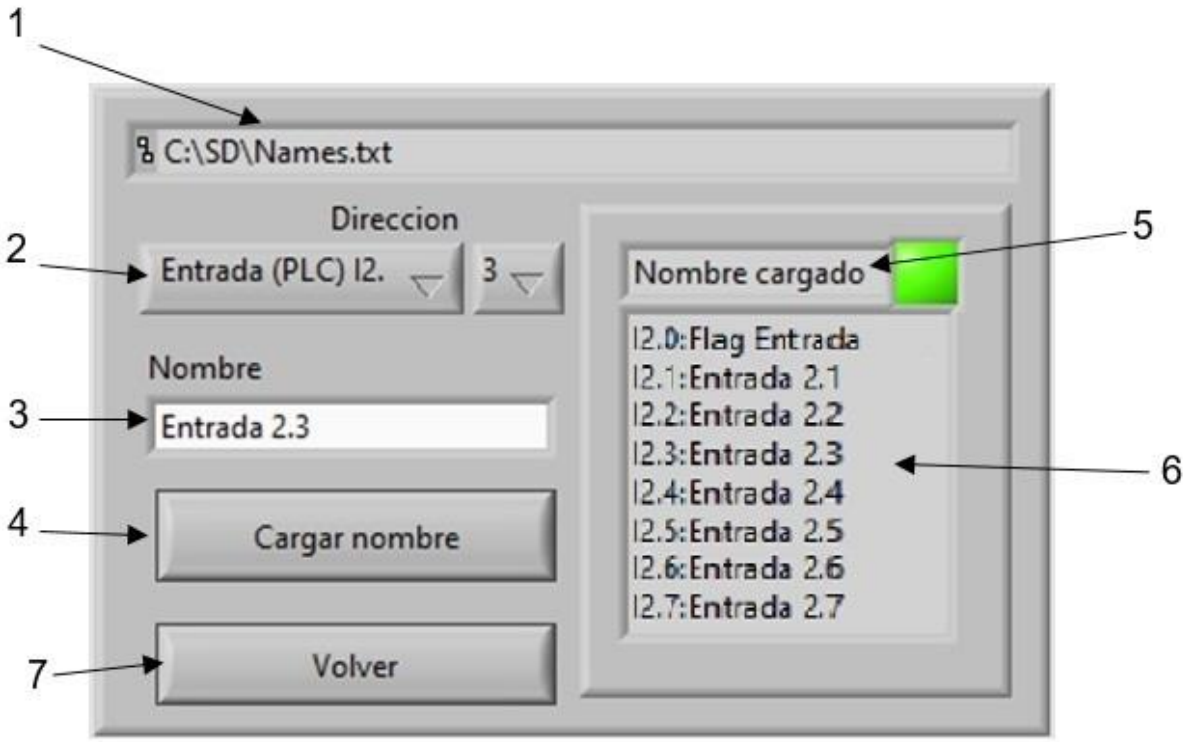


Fig. 6.2.2.5 Ventana nombres

Ventana de configuración de nombres correspondientes a las entradas y salidas a visualizar en el display LCD.

1. Dirección del archivo “Names.txt”
2. Dirección de la entrada o salida a configurar
3. Nombre a asignar. Solo se permite un máximo de 15 caracteres.
4. Botón de carga de nombre.
5. Indica el resultado de la acción causada por el botón “Cargar nombre”. En caso de éxito se indicara “Nombre cargado” y el LED se mantendrá en verde, si lo ingresado no es válido se indicara “Nombre muy largo” o “Nombre vacío” y el LED se volverá rojo.
6. Se indica el nombre actual de las entradas o salidas. Las mismas se visualizan de manera agrupada según la dirección seleccionada (EJ. Si selecciona en dirección “I2.”, se mostraran desde I2.0 hasta I2.7).

6.3 Modo de uso

6.3.1 Eventos

Los eventos pueden ser leídos desde el archivo “EVENT.txt” ubicado en la memoria SD, en el mismo se indicará fecha, hora, texto asignado y el valor que se adquiere al disparar el evento.

EJ: 29/08/2019 - 17:23:55: Se ha modificado el valor de I2.1 (1).

El (1), indica que se ha producido cuando la entrada I2.1 se ha activado. Esto permite que al requerir registro de evento ante cualquier cambio no se generen confusiones durante la verificación.

6.3.2 Pantalla

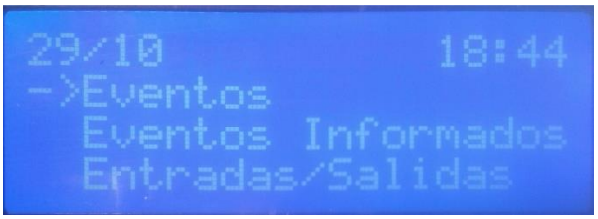


Fig. 6.3.2.1 Pantalla principal

Para controlar la pantalla se utilizan tres botones. Los botones de Arriba y Abajo permiten cambiar la selección del menú en el cual se esté. Mientras que el botón función permite seleccionar la opción deseada. Para volver se debe seleccionar la opción “Volver”, o simplemente presionar el botón función en los menús donde se encuentre tal opción.

Si no se presiona ningún botón durante 20s se pondrá en modo espera en el cual mostrará el último evento ingresado en la memoria SD, en caso de presionar algún botón volverá al menú principal.

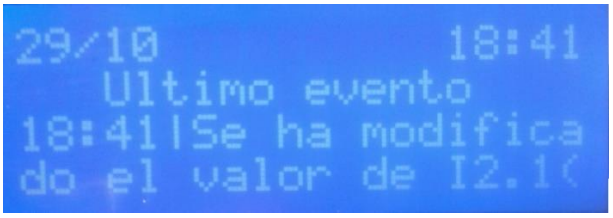


Fig. 6.3.2.2 Pantalla en modo espera

6.3.3 Menús

Dentro del menú principal se pueden distinguir tres opciones:

- **Eventos:** Detallan los últimos 6 eventos que han sido registrados en la memoria SD, en caso de presionar “Función” sobre alguno de ellos se accederá al detalle del mismo. Se debe seleccionar “Volver” para regresar al menú principal.

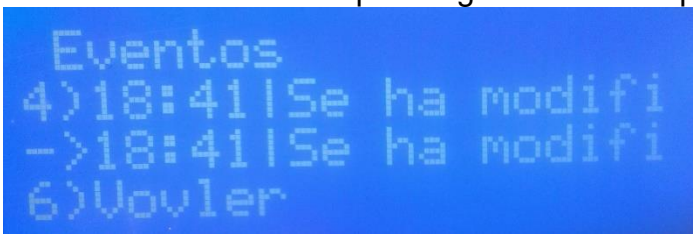


Fig. 6.3.3.1 Menú eventos

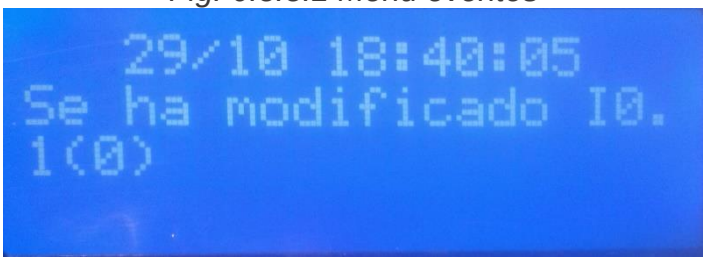


Fig. 6.3.3.2 Detalle eventos

- **Eventos Importantes:** Análogo a Eventos, pero solo aparecerán aquellos eventos los cuales se registra que deben enviar mensajes SMS (independientemente si se envía o no por causa del horario laboral).
- **Entradas/Salidas:** Se accede al submenú, en el cual se pueden seleccionar paquetes de entradas/salidas (I0.X, I1.X, I2.X, Q0.X, Q1.X, E0.X, S0.X). Cada opción permite visualizar el estado de cada uno de sus ocho componentes, indicando su dirección y nombre. En la parte superior derecha de la pantalla se podrá ver una seguidilla de ocho caracteres, los cuales indican el estado de las Entradas/Salidas pertenecientes al paquete.

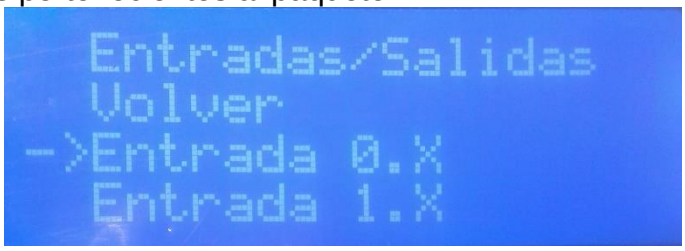


Fig. 6.3.3.3 Menú Entradas/Salidas

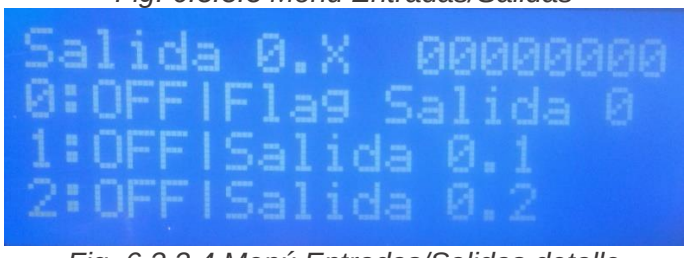


Fig. 6.3.3.4 Menú Entradas/Salidas detalle

Nótese: el primer valor corresponde a la entrada/salida YX.0, mientras que el último al YX.7

6.4 Mensajes de texto

Al enviar un mensaje de texto al dispositivo de monitoreo, se debe enviar en adición a la función a ejecutar, el código de administrador. En caso de no estar adjunto el dispositivo ignorara el mensaje. En caso de no ser una función definida, se obtendrá de respuesta el mensaje “Función no encontrada”. Se debe tener en cuenta que Arduino reconoce Mayúsculas, por lo cual el código debe ser escrito respetando mayúsculas.

Existen dos posibilidades para obtener una respuesta del dispositivo de monitoreo, utilizando las funciones definidas o ingresando el código manualmente “-MAN:”

Funciones definidas por el usuario

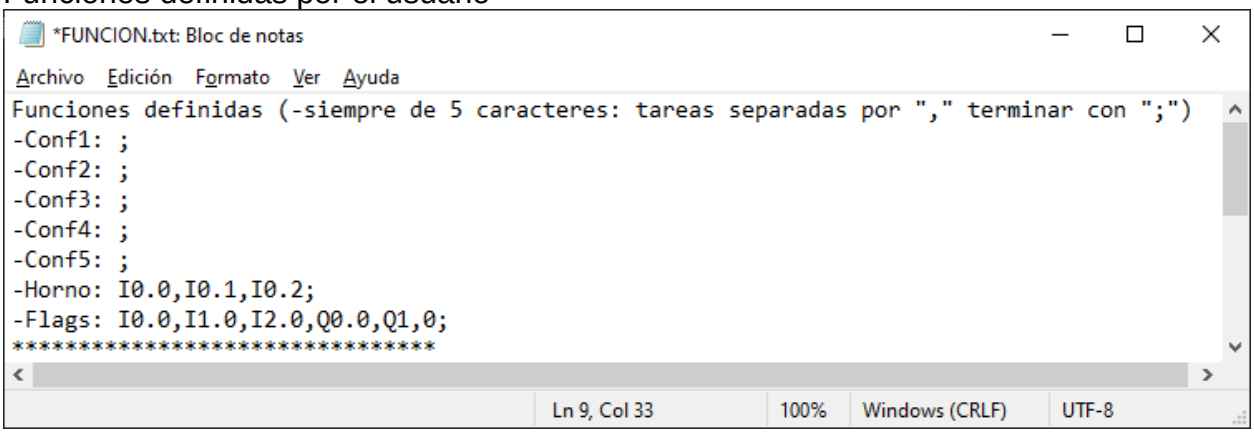


Fig. 6.4.1 Funcion.txt

Como se ha visto previamente en este capítulo 4, para poder solicitar información del estado de las Entradas/Salidas es necesario crear funciones en el archivo “FUNCION.txt”. Considerando que se han programado las funciones de la imagen, se podría enviar el siguiente mensaje:
>Envío: “-Adm42 –Horno:” (siendo Adm42 es el código de administrador)

>Respuesta: “I0.0:OFF|Flag Entrada 0 I0.1: ON|Entrada 0.1 I0.2: ON|Entrada 0.2”

Lo cual indicará que I0.0 se encuentra desactivada mientras que I0.1 e I0.2 se encuentran activas. Para el ejemplo se utilizaron los archivos mostrados como ejemplos de configuración.

Ingreso manual

Para obtener una respuesta fuera de las configuradas en el archivo “Funcion.txt” se puede utilizar le método Manual. Para ello se debe escribir en el mensaje la clave del administrador y el código “-MAN:” seguido por las tareas a realizar (separadas por “,” en caso de ser más de una) y terminar con “;”. De igual manera que se definiría una función en el archivo “Funcion.txt”.

EJ: -Adm42 –Man:I0.0,I0.1,I0.2; - Se obtendrá el mismo efecto que la función “-Horno” definida previamente. (Siendo Adm42 el código de administrador)

CAPÍTULO N°7:
ANALISIS DE COSTOS

7.1 Objetivo del capítulo

Se realizará una comparación de costos de las alternativas mencionadas en el capítulo 2 con el costo del prototipo creado.

7.2 Costo del prototipo

En la siguiente tabla se presenta el costo de los productos utilizados.

Componente	Costo	Cantidad	Total
Arduino MEGA 2560	1050.00	1	1050.00
Shield GSM/GPRS	2155.00	1	2155.00
Shield SD	360.00	1	360.00
Memoria SD 16 GB	310.00	1	310.00
MAX 485	73.00	1	73.00
Display LCD 20x04	602.00	1	602.00
Módulo I2C	N/A	1	N/A
Módulo de 8 Relés Optoacoplados	761.83	1	761.83
Sistema de alimentación ininterrumpida (UPS)	2499.00	1	2499.00
Paca experimental 100x50	150.81	1	150.81
Placa experimental 50x50	82.62	1	82.62
Tirapostes	75.68	1	75.68
Tirapostes largos	108.08	1	108.08
Interruptor	55.00	2	110.00
Pulsador	52.00	3	156.00
Cables Arduino	241.90	1	241.90
4N35 Optoacoplado	29.38	7	205.66
Gabinete 40cm x 21cm x 30cm	3205.50	1	3205.50
Resistencia (1k)	0.75	18	13.50
Resistencia (22k)	0.75	7	5.25
Total	12165.83		

Tabla 7.2.1 Costo de componentes

Dando como resultado un costo equivalente a 12165.83$ (noviembre de 2019), dicho costo incluye únicamente materiales, se debe considerar en adición el costo de mano de obra. La programación y pruebas del dispositivo han tardado 3 meses y el ensamblado 5 días. Por su puesto, el costo obtenido será difícil de comparar con un producto del mercado dado que para poder obtener una nueva unidad no es necesario crear un nuevo programa, sino que simplemente debe cargarse, reduciendo en gran medida el costo de la nueva unidad.

7.3 Comparación de costos

En la siguiente tabla se detallan los costos de los dispositivos mencionados en el capítulo 2.

Método	Costo	Diferencia	Relación (%)
MD720-3	102326.4	90160.57	841.0967439
CP243-1 IT	26000	13834.17	213.7133266
X-Messenger	35904	23738.17	295.1216645
Prototipo	12165.83	0	100

Tabla 7.3.1 comparación de costos

Considerando lo expuesto en la anterior tabla se puede observar que el costo del método alternativo más económico es 213.71% el valor del prototipo. En otras palabras el prototipo es 46.79% más económico que el uso del CP242-1 IT. Adicionalmente este método únicamente permite el uso de la tecnología Ethernet. Considerando el siguiente de las alternativas, el X-Messenger posee funcionalidades similares al prototipo con un costo de casi el triple del prototipo.

7.4 Conclusiones

Teniendo en cuenta los costos y las funcionalidades de las alternativas expresadas en el capítulo 2, se puede concluir que el prototipo resulta tener un costo muy inferior a la alternativa con funcionalidades similares (X-Messenger) y un costo reducido a la alternativa que podría solucionar parcialmente los objetivos planteados.

CAPÍTULO N°8: CONCLUSION

8.1 Objetivo del capítulo

Informar el resultado del proyecto presentado e indicar pasos futuros que podría tomar el mismo.

8.2 Conclusión final

En el presente informe se presentó el diseño y montado de un dispositivo de telemetría. El cual cumplirá las tareas de monitoreo del horno instalado en el Cementerio Parque, facilitando información sobre su estado, tanto dentro como fuera de los horarios laborales.

El mismo presentó múltiples desafíos a la hora de encontrar una solución a los problemas presentados sin incumplir con las condiciones establecidas y con los limitantes del lugar. Dentro de los principales obstáculos a superar se encontraron:

- Comunicación Arduino/PLC
- Comunicación Arduino/Usuario
- Configuración del dispositivo

Arduino como se menciona en el capítulo 5, es una herramienta que permite ser adaptada según las necesidades. El hecho de que sea libre tanto en Software como en Hardware permite la existencia de una amplia comunidad que trabaja y comparte información. Dentro de la misma, se encuentran las llamadas “librerías” las cuales están conformadas principalmente por un conjunto de funciones que permiten simplificar el código o bien facilitar el uso de Shields o módulos determinados.

El primero de los desafíos representó la búsqueda y adaptación de una librería la cual permitiera un intercambio de información entre Arduino y el PLC. El camino a utilizar fue Modbus, un protocolo ampliamente utilizado a nivel industrial.

Por el lado de Arduino se encontraron múltiples librerías, dentro de las cuales utilizaban puerto el Serie o el Ethernet. Dentro de ambas posibilidades, aquellas que utilizaban el puerto Ethernet tendían a ser más funcionales a lo buscado. Mientras que aquellas que utilizaban el puerto Serie tendían a no estar completas o no funcionar correctamente.

Desde el punto de vista del PLC, para el uso de Modbus utilizando el puerto Ethernet era necesario un módulo de expansión, el cual resulto imposible de adquirir. El uso de Modbus por el puerto Serie represento una gran dificultad, las librerías Modbus no son compatibles con el CPU 216-2, por lo cual se tuvo que buscar la manera de configurar el “freeport” para funcionar bajo el protocolo Modbus sin necesidad de las librerías. Esto fue posible gracias a la modificación de uno de los ejemplos suministrados por Siemens en el cual utilizan Modbus de ese modo. La dificultad presente en tal tarea fue buscar la forma de configurarlo para trabajar con los paramentos necesarios (velocidad, paridad, etc.).

Nuevamente desde Arduino, sabiendo que la única posibilidad del uso de Modbus implicaría el uso del puerto Serie, fue necesaria la búsqueda de una librería funcional. Lamentablemente la gran mayoría de librerías o bien son incompletas (carentes de funciones necesarias de Modbus) o simplemente no funcionan correctamente. La utilizada fue “Simple-Modbus-Master”, la librería más completa encontrada para el uso

en el proyecto. Para poder utilizarla fue necesario investigar su funcionamiento y así realizar el código en función a lo obtenido. Finalmente luego de realizar algunas modificaciones a la misma, fue obtenida de manera exitosa una comunicación.

Fue necesaria la creación de un sistema que permita al usuario configurar el funcionamiento del dispositivo acorde a las necesidades. Para reducir la dificultad de comunicación entre el usuario y Arduino se ideo un sistema que le otorga al usuario la posibilidad de crear funciones las cuales cumplen con tareas programables. Estas funciones podrán ser ejecutadas por el dispositivo al enviarle un SMS con el código de usuario y la función. Estas configuraciones son guardadas en una memoria SD.

El último de los desafíos fue originado en base a la futura idea de una modificación de las instalaciones, agregando sensores adicionales (Sensor de gases, sensor de movimiento). Originalmente el programa se planeaba realizarlo dentro del código. Esto generaría un problema a la hora de expandir la instalación, dado que para que esta sea reconocida por el programa era necesario modificarlo.

Esto genero la necesidad de darle flexibilidad al dispositivo. Para lograrlo, se utilizó la memoria SD con archivos configurables por el usuario en función a sus necesidades. Permitiendo así que en caso de una modificación del programa cargado en el PLC para que el sistema mantenga su funcionamiento correcto solo debe configurarse la memoria SD, evitando de ese modo que se deba modificar el código instalado en la placa Arduino.

Todo esto ha llevado a la creación de un dispositivo de bajo costo el cual no solo cumple con los objetivos originalmente dados, sino que tiene la posibilidad de adaptación para funcionar en múltiples situaciones. Teniendo la capacidad de adaptabilidad a los requisitos de los usuarios, capacidad de monitoreo de otro dispositivo vía Modbus, sistema configurable de alarma y capacidad de control de sus propias salidas (en respuesta a un mensaje de texto, ante una entrada de Arduino o cambio en entradas o salidas del PLC).

8.3 Posibles mejoras al proyecto

El proyecto ha cumplido los objetivos iniciales y considerando su naturaleza como proyecto final de carrera de grado se considera finalizado. Sin embargo, el mismo puede ser optimizado, a continuación una lista con posibilidades.

- Optimización del código en Arduino
- Configuración de Modbus desde memoria SD
- Capacidad de monitoreo de más de un dispositivo vía Modbus
- Capacidad de control de dispositivos conectados
- Uso de otras tecnologías de comunicación (GPRS, Zigbee, etc.)
- Creación de APP para Smartphone permitiendo una fácil comunicación
- Configurar el uso de más Entradas/Salidas
- Envío de mensajes de texto a más de un número telefónico según evento
- Consideración de mensajes únicamente si provienen de números designados

GLOSARIO DE SIGLAS

A continuación se presenta una lista de las siglas utilizadas en el presente proyecto, las mismas se encuentran ordenadas de forma alfabética.

ASCII: Código estándar estadounidense para el intercambio de información (*American Standard Code for Information Interchange)*
AT: Comandos de atención (ATtention commands)
CEPT: Conferencia Europea de Administraciones de Correos y Telecomunicaciones (en francés, Conférence européenne des administrations des postes et des télécommunications)
EMI: Interferencia electromagnética (Electromagnetic Interference)
ETSI: Instituto europeo de normas de telecomunicaciones (European Telecommunications Standards Institute)
GPRS: Servicio general de paquetes vía radio (General Packet Radio Service)
GSM: Sistema global para las comunicaciones Móviles (Global System for Mobile communications)
HTML: Lenguaje de Marcas de Hipertexto (HyperText Markup Language)
I/O: Entradas/Salidas, E/S (Input/Output)
I2C: Circuito inter-integrado (Inter-Integrated Circuit)
ICSP: Programador de chip serial integrado (In Chip Serial Programmer)
IDE: Entorno de desarrollo integrado (Integrated Development Enviroment)
IP: Protocolo de internet (Internet Protocol)
LCD: pantalla de cristal líquido (liquid crystal displays)
LED: Diodo emisor de luz (*Light-emitting Diode*)
MPI: Interfaz multipunto (Multi-Point Interface)
OSI: Interconexión de Sistemas Abiertos (Open Systems Interconection)
PLC: Controlador de lógica programable (Programmable Logic Controller)
PLC: Controlador Lógico Programable (Programmable Logic Controller)
PPI: Interfaz punto a punto (Point to Point Interface)
PWM: Modulación por ancho de pulsos (Pulse Width Modulation)
RAM: Memoria de acceso aleatorio (*Random Access Memory*)
ROM: Memoria de solo lectura (*Read Only Memory*)
RTC: Reloj de tiempo real (Real Time Clock)
RTU: unidad terminal remota (Remote Terminal Unit)
SIM: Módulo de identificación de abonado (Subscriber Identity Module)
SMS: Servicio de mensajes corto (Short Message Service)
SPI: Protocolo de comunicación serial (Serial Peripherical Interface)
TCP: Protocolo de control de transmisión (Transmission Control Protocol)
TTL: lógica transistor a transistor (transistor-transistor logic)
UART: Transmisor-Receptor Asíncrono Universal (Universal Asynchronous Receiver-Transmitter)
UDP: Protocolo de datagrama de usuario (User Datagram Protocol)’
USB: Bus serial universal (Universal Serial Bus)
VPN: Red privada virtual (Virtual Private Network)

BIBLIOGRAFIA

[1] Arduino, (2019) http://www.arduino.cc

[2] **Domínguez, Alberto Castro**, (2000). *Sistema de control de temperatura a través de Arduino y la tecnología GPRS/GMS.* **España: Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación**

[3] **Hernández Oré Ricardo Japhet, Ludeña Gutierrez Abel Gustavo**, (2015). *Telelocalización y captura de imágenes para mejorar una base de datos criminalística del robo de vehículos en la ciudad de lima.* **Perú: Universidad Ricardo Palma.**

[4] **Álvaro Pachón de la Cruz**, (2004). *Evolución de los sistemas móviles celulares GSM*. **Colombia: Universidad ICESI.**

[5] **Hayes modem AT command set**, (2019). https://www.lammertbies.nl/comm/info/hayes-at-commands

[6] **SIMCom**, (2015). *Sim900 AT Commands Manual.*

[7] **Aitziber Marín Iturrarte**, (2012). *Control de PLCs Siemens S7-1200 mediante el protocolo Modbus a través del programa LABVIEW para realización de prácticas de comunicación industrial.* **España: Universidad Pública de Navarra.**

[8] **Rabadán Barastegui Juan José**, (2017). *Diseño y desarrollo de una red Modbus RTU basada en Arduino*. **España: Universidad de Sevilla**.

[9] **MODICON, inc.** (Junio 1996). *Modicon Modbus Protocol Reference Guide.*

[10] **SIEMENS.** (2008). *Manual del sistema de automatización S7-200*

[11] **Modbus,** (2019). http://modbus.org/

ANEXO I: CODIGO FUENTE

Código fuente del programa realizado en Arduino IDE 1.6.11

```
        // **Librerias**
#include <SoftwareSerial.h> //Libreria de GPRS
#include <Wire.h> // Necesaria por RTClib.h
#include <LiquidCrystal_I2C.h> //Libreria de pantalla
#include <SPI.h> //Libreria que permite comunicacion con bus SPI (Serial
Peripheral Interface)
#include <SD.h>  //Libreria para uso de memoria SD*/
#include <SimpleModbusMaster.h> //Libreria de maestromodbus
#include <Arduino.h>
#include "RTClib.h" //Reloj

// *** Variables y definiciones ***

File Archivo; //Declara archivo

RTC_Millis rtc;

SoftwareSerial GPRS (50, 19); // Define pines de TX y RX para el shiedl GPRS
-- Arduino UNO: (TX = 3 RX = 2)|Mega: (TX = 50 ; RX = 19)
#define Powerpin 30 // define al pin 30 como Powerpin del shield GPRS

// *** Declaraciones Pantalla ***

#define I2C_ADDR    0x27 // <<----- Direccion, se puede encontrar utilizando
el programa I2C Scanner adjunto en anexos
#define BACKLIGHT_PIN     3
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
#define D6_pin  6
#define D7_pin  7
LiquidCrystal_I2C  lcd(I2C_ADDR, En_pin, Rw_pin, Rs_pin, D4_pin, D5_pin,
D6_pin, D7_pin); //Declara "lcd" como pantalla

// *** Fin declaraciones Pantalla ***

// *** Comienzo de definiciones para MODBUS ***

#define baud 19200
#define timeout 1000
#define polling 100 // the scan rate
#define retry_count 10
#define TxEnablePin 36 // Enable pin

// Numero Total de paquetes
```

```
#define TOTAL_NO_OF_REGISTERS 1

enum
{
  PACKET1,
  PACKET2,
  PACKET3,
  PACKET4,
  PACKET5,
  TOTAL_NO_OF_PACKETS // Actualiza automaticamente el numero de paquetes.
};
// Creacion de Array de paquetes y de Master register
Packet packets[TOTAL_NO_OF_PACKETS];
unsigned int regs[TOTAL_NO_OF_REGISTERS];

// *** Fin definiciones para Modbus ***

// *** variables globales ***

int i,n; // Integers para contadores varios
String Number, Codigo,LT;
char Message[201];
byte Messageind,mupok;
char texto[61];
byte bin [8], di, dn, nivel, index, func, boton,mn; // Variable para lectura
de binario y indicadores de menu utilizados en display
byte
Estado[7][8]={{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,
0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0}};
/* Estado, matriz que guarda los estados del PCL, donde:
 *  [0][x] = I0.X
 *  [1][x] = I1.X
 *  [2][x] = I2.X
 *  [3][x] = Q0.X
 *  [4][x] = Q1.X
 *  [5][x] = E0.X
 *  [6][x] = S0.X
*/
byte
Monitor[6][8]={{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0
,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0}}; // Controlar cambio ?
byte IN[8]={53,51,49,47,45,43,41,39}; // Pines de entrada arduino
byte OUT[8]={37,35,33,31,29,27,25,23}; // Pines de salida arduino
char Eventos [450];
char Eventosimp [450];
char UltimoEvento [75];
unsigned long tprev,tact,TI,Mupmilli;
unsigned long TS [8] = {0,0,0,0,0,0,0,0}; //Timers de salidas arduino
char Input0 [8][16];
char Input1 [8][16];
char Input2 [8][16];
char Input3 [8][16]; //Ard
char Output0 [8][16];
char Output1 [8][16];
char Output2 [8][16]; //Ard
byte g = 0; //-----------------------------------------------------------------
-----------------------------------BORRAR LUEGO-----------------------------
```

```
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-----------------------------------------------
unsigned long tdebug = 0; //-------------------------------------------------
-------------------------------------BORRAR LUEGO-----------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-----------------------------------------------
// *** Comienzo del setup ***

void setup()
{
  pinMode(Powerpin, OUTPUT); // Asignacion del pin como una salida
  for (i=0;i<8;i++){
    pinMode(IN[i], INPUT); // Entradas de arduino
    pinMode(OUT[i], OUTPUT); // Salidas de arduino
    digitalWrite(OUT[i],LOW);
  }
  // *** Inicializacion de Pantalla ***

  lcd.setBacklightPin(BACKLIGHT_PIN, POSITIVE);
  lcd.setBacklight(HIGH);
  lcd.begin (20, 4); // Donde (X,Y) X: numero de caracteres por linea de la
pantalla Y: Numero de lineas de la pantalla
  lcd.setCursor (0, 1); //Posiciona el cursor en posicion 0,1 (segunda linea)
  lcd.print ("     Iniciando      ");


  // *** Configuracion inicial de MODBAS ***
  //  Armado de paquetes

  modbus_construct(&packets[PACKET1], 1, READ_COIL_STATUS, 0, 8, 0);
//Lectura de Q0.0 a Q0.7
  modbus_construct(&packets[PACKET2], 1, READ_COIL_STATUS, 8, 8, 0);
//Lectura de Q1.0 a Q1.7
  modbus_construct(&packets[PACKET3], 1, READ_INPUT_STATUS, 0, 8, 0);
//Lectura de I0.0 a I0.7
  modbus_construct(&packets[PACKET4], 1, READ_INPUT_STATUS, 8, 8, 0);
//Lectura de I1.0 a I1.7
  modbus_construct(&packets[PACKET5], 1, READ_INPUT_STATUS, 16, 8, 0);
//Lectura de I2.0 a I2.7

  // Inicializacion de MODBUS

  modbus_configure(&Serial3, baud, SERIAL_8O1, timeout, polling, retry_count,
TxEnablePin, packets, TOTAL_NO_OF_PACKETS, regs);

  // *** Inicializacion de Serial ***

  Serial.begin(9600); // Inicio Serial

  // *** Inicializacion de Shield SD ***

  LCDclear(2); //Posiciona el cursor en posicion 0,1 (segunda linea)
  lcd.print (" Lectura tarjeta SD ");

  Serial.print("Iniciando tarjeta SD...");
```

```
  pinMode(SS, OUTPUT);   // SS? ver apunte de libreria
  if (!SD.begin(10, 11, 12, 13)) { //Se conecta a la memoria SD utilizando
los pines 10, 11, 12 y 13. (Recomendados para MEGA)
    Serial.println("Inicio fallido!"); //informa el fallo
    lcd.print ("   SD no conectada  ");
    delay(5000);
    LCDclear (3); // Limpia la 4ta linea
    return;
  }
  // *** Inicializacion de RLT ***

  rtc.begin(DateTime(F(__DATE__), F(__TIME__)));

  // *** Inicializacion de GPRS ***

  Serial.print("iniciando Gprs");
  LCDclear (2); //Limpia y posiciona el cursor en posicion 0,2 (tercer linea)
  lcd.print ("     Modulo GMS     ");
  GPRS.begin(19200); // Duda si va antes o despues de POWER();
  Power(); // Enciende el shield GPRS----------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
------------------------------------------------------------

  // Inicializacion de variables
  i = 0;
  n = 0;
  mn = 0;
  regs[0] = 256; // regs[0] maximo por lectura = 255;
  dn = 0; // variable de display, valor de inicio
  di = 1; // variable de display, valor de inicio
  nivel = 0;
  index = 0;
  func = 0;
  memset(Eventos,' ',449);
  memset(Eventosimp,' ',449);
  UltimoEvento [0] = '\0';
  strcpy(UltimoEvento,"              Todo listo");
  tprev = 0;
  tact = 0;
  TI = 0;
  // *** Lectura de datos y carga de variables desde SD ***
  Serial.println("Lecturas de datos y carga de datos desde SD");
  lcd.setCursor (0, 1); //Posiciona el cursor en posicion 0,1 (segunda linea)
  lcd.print ("   Carga de datos   ");
  lcd.setCursor (0, 2);
  lcd.print ("     iniciales      ");
  Nombresinit(); //Carga nombres
  Monitorinit(); //Cargado de matriz monitor desde PLCST.txt
  SetupFromSD(); //Cargado de variables desde SETUP.TXT
  LCDclear(4);
  LCDclear(1); //Posiciona el cursor en posicion 0,1 (segunda linea)
  lcd.print ("  Recibiendo datos  ");
  LCDclear(2);
  lcd.print ("     desde PLC      ");

   // *** Primer lectura de MODBAS ***
```

```
  Serial.print("iniciando carga de modbus");
  initialmodbus (); // Lectura preliminar y ajuste para lectura de I0.0 en
proximo Update
  Serial.println("inicio exitoso");
  Serial.println("Carga de matriz de estado");
  n=0;
  do{ // Carga de matriz de estado
    regs [0] = 256;
    modbus_update();
    delay(500);
    if (regs[0] != 256) {
      Dec2Bin(regs[0]);
      for (i=0;i<8;i++){
        Estado[n][i]=bin[i];
      }
      n++;
    }
  } while (n<5);
  n = 0;
  for (i=0;i<8;i++){
    if (digitalRead(IN[i]) == HIGH){
      Estado[5][i]=1;
    } else {
      Estado[5][i]=0;
    }
  }
  for (i=0;i<8;i++){
    if (digitalRead(OUT[i]) == HIGH){
      Estado[6][i]=1;
    } else {
      Estado[6][i]=0;
    }
  }
  i = 0;
  Serial.println("Valores de ESTADO");
  do {
    Serial.print(Estado[n][i]);
    Serial.print(" ");
    i++;
    if (i==8){
      Serial.print (" | ");
      Serial.println(n);
      n++;
      i = 0;
    }
  }
  while (n<5);
  for (i=0;i<8;i++){
    Serial.print (Estado[5][i]);
    Serial.print (" ");
  }
  Serial.print (" | ");
  Serial.println ("5");
  n=0;
  Serial.println("TODO LISTO");
  LCDclear(4);
  lcd.setCursor (0, 1); //Posiciona el cursor en posicion 0,1 (segunda linea)
```

```
  lcd.print ("     Todo Listo     ");

// *** Asignacion de Interrupciones ***

  attachInterrupt(digitalPinToInterrupt(2), inter2, RISING);
  attachInterrupt(digitalPinToInterrupt(3), inter3, RISING);
  attachInterrupt(digitalPinToInterrupt(18), inter18, RISING);
}

// *** Fin del Setup ***

// *** Compienzo de programa principal ***

void loop()
{
 Mupdate(); // Actualiza lectura
 delay(100);
 if (boton != 5) {
   BOTON();
 }
 TIMER(); // Funcion que se activa por tiempo
 sendAT("AT+CMGR=1","+CMGR:",20000); //Busca mensaje de texto
}

// *** Fin del programa principal ***

// *** Declaracion de funciones ***

// Funciones MODBAS

void Dec2Bin (int val) { // Transforma un numero decimal en binario
  for (i=0;i<8;i++) { // inicializa String
      bin[i] = 2; // Inicializa el string en caso de visualisar "2"
imoplicaria error
  }
  for (i = 0; i<8; i++) { // Calcula los valores 1 = on 0 = off
    if (val % 2 == 0) { //si es par
      bin[i] = 0;
      val = val/2;
    } else { //es impar
      bin[i] = 1;
      val = (val-1)/2;
    }
  }
  i = 0;
}


void initialmodbus () { // Comienza lectura por MODBUS y asegura que proxima
lectura sea el I0.0 hasta I0.7
  i = 0;
  n = 0;
  do {
    regs[0] = 256; // Ajusta valor 256
    modbus_update();
    Serial.print("UPDATE :");
    Serial.print(regs[0]);
```

```
    Serial.print(" ");
    delay(500);
    if (regs[0] != 256) { // En caso de lectura correcta regs[0] valdra como
maximo 2^8 (255)
      Dec2Bin(regs[0]);
      Serial.print(" valor leido : ");
      do{
        Serial.print(bin[i]);
        Serial.print(" ");
        i++;
      }
      while (i<8);
      i = 0;
      if (bin[0] == 1) { // Si bin[0] = 1, entonces estaria leyendo Q0.0 o
Q1.0 (paquete 1 y 2)
        Serial.println("N++");
        n++;
      }else{
        Serial.println("NOPE");
        n = 0;
      }
    }
  }
  while (n<2); // Si n=2, se habra leido existosamente Q0.0 y luego Q1.0, el
proximo paquete sera el 3 (I0.0 a I0.7)
  mn = 0;
}

void Nombresinit() { // Lee Nombres de las entradas y salidas desde Names.txt
  char c;
  int indeX = 0;
  nivel = 0;
  Archivo = SD.open("Names.txt", FILE_READ);
  char buf[25]; // No deberia tener mas de 20 caracteres por renglon (mas 2
al finalizar el renglon)
  n = 0;
  Serial.println("Leyendo Names.TXT");
  while (Archivo.available()){
    c = Archivo.read();
    if (c == '\n') { // Busca <cr>
      buf[indeX] = '\0'; // Agrega "\0" para indicar fin.
      switch (buf [3]){
        case '0': n = 0; break;
        case '1': n = 1; break;
        case '2': n = 2; break;
        case '3': n = 3; break;
        case '4': n = 4; break;
        case '5': n = 5; break;
        case '6': n = 6; break;
        case '7': n = 7; break;
      }
      indeX = 0;
      if ((buf[0] == 'Q') && (buf[1] == '0')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Output0[n][i-5] = buf[i];
```

```
          }
        }
      }
      if ((buf[0] == 'Q') && (buf[1] == '1')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Output1[n][i-5] = buf[i];
          }
        }
      }
      if ((buf[0] == 'I') && (buf[1] == '0')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Input0[n][i-5] = buf[i];
          }
        }
      }
      if ((buf[0] == 'I') && (buf[1] == '1')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Input1[n][i-5] = buf[i];
          }
        }
      }
      if ((buf[0] == 'I') && (buf[1] == '2')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Input2[n][i-5] = buf[i];
          }
        }
      }
      if ((buf[0] == 'E') && (buf[1] == '0')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Input3[n][i-5] = buf[i];
          }
        }
      }
      if ((buf[0] == 'S') && (buf[1] == '0')){
        for(i=5;((buf[i] != '\0') || (i<21));i++){
          if ((buf[i] < 127 ) && (buf[i] > 31)) { // verifica que el caracter
buf[i] se encuentre entre 31 y 127 en codigo ascii
            Output2[n][i-5] = buf[i];
          }
        }
      }
      indeX = 0;
      memset(buf,'\0',25);
    }else{
      if (indeX < 20){ // Evita que se superen el numero esperado de
caracteres
        buf[indeX] = c;
```

```
      indeX++;
    }
  }
}
for (n = 0; n<8;n++){
  Input0[n][15] = '\0';
  Input1[n][15] = '\0';
  Input2[n][15] = '\0';
  Input3[n][15] = '\0';
  Output0[n][15] = '\0';
  Output1[n][15] = '\0';
  Output2[n][15] = '\0';
}
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Input0[n][i]);
  }
  Serial.println();
}
Serial.println();
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Input1[n][i]);
  }
  Serial.println();
}
Serial.println();
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Input2[n][i]);
  }
  Serial.println();
}
Serial.println();
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Output0[n][i]);
  }
  Serial.println();
}
Serial.println();
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Output1[n][i]);
  }
  Serial.println();
}
Serial.println();
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Input3[n][i]);
  }
  Serial.println();
}
for (n = 0; n<8;n++){
  for (i = 0; i<15;i++){
    Serial.print(Output2[n][i]);
```

```
    }
    Serial.println();
  }
  Serial.println("Se ha cargado exitosamente los nombres de Entradas y
Salidas");
  Archivo.close(); // Cierra el archivo
}

void Mupdate (){ // Actualiza la matriz de estados y en caso de haber una
modificacion ejecuta funcion de accion // NOT TESTED
  i = 0;
  regs[0] = 256; // regs[0] maximo por lectura = 255;
  modbus_update();
  delay(100);
  if (regs[0] != 256) { // Obtuvo nueva lectura desde el PLC verifica si hubo
cambios
    Dec2Bin(regs[0]);
    if (mupok > 21) {
      strcpy(Message,"Conexion restablecida con el PLC");
      Event(Message,2);
      if (mupok > 22) { //si envio mensaje de texto informando fallo, informa
restablecimiento
        mupok = sendAT("AT+CMGS=", "OK", 10000); // Envia mensaje
        if (mupok == 0) {
          strcpy(texto, "No se ha podido enviar el menaje de texto");
          Event(Message,0);
        }
      }
      initialmodbus(); //sincronizacion
    }
    mupok = 0;
    Mupmilli = millis();
    for (i=0;i<8;i++) {
      if (Estado[mn][i] != bin[i]) {
        if (Monitor [mn][i] == 1) {
          Serial.println("--Se toma accion--"); //---------------DEBUG
RETIRAR SERIALSPRINTS-----------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------
          Serial.print("Estado[");
Serial.print(mn);Serial.print("][");Serial.print(i);Serial.print("]
(");Serial.print(Estado[mn][i]);Serial.print(") !=
bin[");Serial.print(i);Serial.print("]
(");Serial.print(bin[i]);Serial.println(")");
          Action(bin[i]);
          Serial.println("------------------");
        }
        Estado[mn][i] = bin[i];
        Serial.print("Nuevo valor de Estado[");
        Serial.print(mn);Serial.print("][");Serial.print(i);Serial.print("]
(");Serial.print(Estado[mn][i]);Serial.println(")"); //----------DEBUG ------
-------------------------------------------------------------------------
-----------------------------------------------------------
      }
    }
    mn++;
    if (mn == 5) { // verifica entradas arduino
```

```
      mn = 0;
    }
  } else {
    if (mupok < 21) {
      mupok++;
    } else {
      if (mupok == 21) {
        LT = Gettime();
        mupok = 22;
        strcpy(texto,"Comunicacion perdida, esperando reconexion");
        Event(texto,2);
      }
    }
   if ((millis() - Mupmilli > 600000) && (mupok == 22)){
    strcpy(Message,"No se recibe informacion desde el PLC, ultima : ");
//(creo que asi es))
      LT.toCharArray(texto, 61);
      strcat(Message,texto+'\0'); //(creo que asi es))
      Event(Message,2);
      mupok = sendAT("AT+CMGS=", "OK", 10000); // Envia mensaje
      Serial.print("SE INFORMA QUE NO SE COMUNICA");
      Serial.print(Message);
      if (mupok == 0) {
        strcpy(texto,"No se ha podido enviar el menaje de texto");
        Event (texto,0);
      }
      mupok = 23;
    }
  }
  for (i=0;i<8;i++){
    if (digitalRead(IN[i]) == HIGH){
      Estado[5][i]=1;
    } else {
      Estado[5][i]=0;
    }
  }
  for (i=0;i<8;i++){
    if (digitalRead(OUT[i]) == HIGH){
      Estado[6][i]=1;
    } else {
      Estado[6][i]=0;
    }
  }
  i = 0;
}

// Funciones de SIM

void Power () { //Enciende shield SIM900  /// Falta limpiar los lcd.print ---
-- CORREGIR VERIFICACION DE EXITO ------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------
  byte flag = 0;
  n = 0;
  Serial.println("Iniciando comunicacion remota");
  lcd.setCursor (0, 2);
  lcd.print ("Comunicacion remota");
```

```
  digitalWrite(Powerpin, HIGH); //Powerpin = pin D9 del shield
  delay (3000);
  digitalWrite(Powerpin, LOW);
  while (flag == 0) { //
    lcd.setCursor (n, 1);
    n++;
    lcd.print ("*");
    flag = sendAT("AT", "OK", 2000);
    if (flag == 0) {
      lcd.setCursor (0, 2);
      lcd.print ("   Intento fallido  ");
      lcd.print ("    Reintentando    ");
      Serial.println("Intento fallido de inicio de SIM900");
      delay (1000);
      LCDclear(4);
      if (n == 20) { //Luego de 20 intentos
         flag = 1; //Fuerza inicio
         Serial.println("No se ha podido conectar a la RED, se seguira
operando sin comunicacion remota");
         Event("No se ha podido conectar a la RED, se seguira operando sin
comunicacion remota",2);
         lcd.setCursor (0, 2);
         lcd.print ("  Intento Fallido   ");
         lcd.print ("  Operando sin RED  ");
      }
    } else {
      lcd.setCursor (0, 3);
      lcd.print ("  Intento exitoso   ");
      Serial.println("Intento exitoso de inicio de SIM900");
      delay (1000);
    }
  }
  delay (14000);
  GPRS.print("AT+CMGF=1\r"); //configura para funcionar en modo texto
  delay (1000);
  sendAT("AT+CPMS=\"SM\",\"SM\",\"SM\"","OK",1000); //Asigna el uso de la
memoria de la SIM
  if (n != 20) {
    Event("Conexion remota establecida con exito",2);
  }
  LCDclear(4);
}

byte sendAT(String AT, char* expected, unsigned int Timeout) { //se utiliza
Timeout (con "T") dado que timeout ya esta definida (verificar el
funcionamiento de esto)
  byte flag = 0;
  char ans [250];
  unsigned int prev;
  i = 0;
  char charcodigo[10];
  memset(ans,'\0',250);
  while (GPRS.available() != 0) GPRS.read(); //Limpia el buffer de entrada
  if (AT != "AT+CMGS=") { //lee mensajes de texto en caso de haber
    GPRS.println(AT+'\13'+'\10');
    prev = millis ();
    do { // lee respuesta
```

```
    if (GPRS.available() != 0){
      ans[i] = GPRS.read();
      i++;
    }
  } while ((GPRS.available() != 0) && ((millis() - prev) < Timeout));
/*Message[0] = '\0'; Messageind = 0;
  ans[0] = 'A';
  ans[1] = 'd';
  ans[2] = 'm';
  ans[3] = '4';
  ans[4] = '2';
  ans[5] = ' ';
  ans[6] = ' ';
  switch (g){
    case 0:
      ans[7] = '-';
      ans[8] = 'O';
      ans[9] = 'N';
      ans[10] = ':';
      ans[11] = 'S';
      ans[12] = '2';
      ans[13] = ',';
      ans[14] = '2';
      ans[15] = ';';
      ans[16] = '\0';
      g = 1;
    break;
    case 1:
      delay(1000);
      ans[7] = '-';
      ans[8] = 'O';
      ans[9] = 'F';
      ans[10] = 'F';
      ans[11] = ':';
      ans[12] = 'S';
      ans[13] = '1';
      ans[14] = ',';
      ans[15] = '0';
      ans[16] = ';';
      ans[17] = '\0';
      g = 2;
    break;
    case 2:
      ans[7] = '-';
      ans[8] = 'S';
      ans[9] = 'a';
      ans[10] = 'l';
      ans[11] = 'i';
      ans[12] = 'd';
      ans[13] = 'a';
      ans[14] = ':';
      ans[15] = '\0';
      g = 3;
    break;
    case 3:
      ans[7] = '-';
      ans[8] = 'H'; // cambiar luego la 0 por una H;
```

```
        ans[9] = 'o';
        ans[10] = 'r';
        ans[11] = 'n';
        ans[12] = 'o';
        ans[13] = ':';
        ans[14] = ';';
        ans[15] = '\0';
        g = 4;
      break;
      case 4:
        ans[7] = '-';
        ans[8] = 'F';
        ans[9] = 'l';
        ans[10] = 'a';
        ans[11] = 'g';
        ans[12] = 's';
        ans[13] = ':';
        ans[14] = ';';
        ans[15] = '\0';
        g = 5;
      break;
      case 5:
        ans[7] = '-';
        ans[8] = 'H';
        ans[9] = 'o';
        ans[10] = 'r';
        ans[11] = 'n';
        ans[12] = 'o';
        ans[13] = ':';
        ans[14] = ';';
        ans[15] = '\0';
        g = 6;
      case 6:
        memset(ans,'\0',15);
      break;
    }
    if (g < 6) {
      Serial.print("Codigo = "); Serial.print(Codigo); Serial.print(" g = ");
Serial.print(g); Serial.print(" ans = "); Serial.println(ans); // DEBUG -----
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------
    } */
    if (AT == "AT+CMGR=1") {
      Codigo.toCharArray(charcodigo, 10);
      if (strstr(ans, charcodigo)){
        Serial.println("Codigo encontrado");
        AT = "AT+CMGS=";
        if (strstr(ans, "-ON:")){ // funciona OK
          ONF(1,ans);
        } else {
          if (strstr (ans, "-OFF:")) { // funciona OK
            ONF(0,ans);
          } else {
            if (strstr (ans, "-Salida:")) { //parece funcionar OK
              OutputARD();
```

```
            } else {
              // Serial.println("Code salteado");
              Code(ans); //Funcion de busqueda de codigos en mensajes ---
FALTA DEFINIR CODE -- Y que defina Message para generar respuesta
              // Message[0] = 'A'; Message[1] = '\0';
            }
          }
        }
      } else {Serial.print("Codigo NO encontrado: "); Serial.println(ans);}
    }
   }
   if (AT == "AT+CMGS=") {
    Serial.print("CODIGO ENTRANTE : ");Serial.println(ans);
    Serial.println("AT = AT+CMGS");
    expected = "OK";
    AT = AT + '\"' + "+" + Number + '\"';
    Serial.print("AT : "); Serial.println(AT);
   //-------------------------------------------------------------------------
-------------------------------------------------------------------------
---------------
    GPRS.println(AT+'\13'+'\10'); //Envia comando AT
    prev = millis (); //Toma el tiempo de inicio
    i = 0;
    do { // Si hay datos en el buffer de entrada UART lee
      if (GPRS.available() != 0){ // lee respuesta
        ans[i] = GPRS.read();
        i++;
      }
    } while ((GPRS.available() != 0) && ((millis() - prev) < Timeout));
    prev = millis (); //Toma el tiempo de inicio
    delay(1000);
    GPRS.println(Message); //Variable del global mensaje a enviar
    delay(500);
    GPRS.print((char)26); //Indica fin del mensaje
    delay(5000);
    i = 0;
    do { // Si hay datos en el buffer de entrada UART lee
      if (GPRS.available() != 0){ // lee respuesta
        ans[i] = GPRS.read();
        i++;
      }
    } while ((GPRS.available() != 0) && ((millis() - prev + 5000) <
Timeout));
    ans[i] = GPRS.read(); //espera OK.
    Serial.print("Mensaje enviado: ");
    for (i = 0; Message[i] != '\0'; i++){
      Serial.print (Message[i]);
    }
    memset(Message,'\0',201);
    Serial.println();
    ans [0] = 'O'; ans [1] = 'K'; ans [2] = '\0'; //-----------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
----
  }
  if (strstr(ans, expected) != NULL) { //busca si la respuesta es la esperada
```

```
    flag = 1; //indica que obtuvo respuesta esperada
    Serial.println("SEND AT EXITOSO");
  }
  return flag; // (flag = 1 OK, flag = 0 Error)
}

void Code (char ans[250]){ // Parece OK
  char c;
  byte indeX = 0;
  char funcion[6];
  funcion[5] = '\0';
  Archivo = SD.open("FUNCION.txt", FILE_READ);
  char buf[100]; // No deberia tener mas de 20 caracteres por renglon (mas 2
al finalizar el renglon)
  Message[0] = '\0'; Messageind = 0;
  while ((Archivo.available()) && (Message[0] == '\0')){
    c = Archivo.read();
    Serial.print(c);
    if (c == '\n') { // Busca <cr>
      buf[indeX+1]='\0'; // indica fin de buf
      indeX = 0;
      do {
        if (buf[indeX]=='-') {
        } //busca "-"
        if (buf[indeX] != '-') {
          indeX++;
        } else {
          if (buf[indeX] == '\0'){
            break;
          }
        }
      } while (buf[indeX] != '-'); //encuentra "-" o fin de buf
      if ((buf[indeX] == '-') && (buf[indeX+6] = ':')){
        funcion[0] = buf[indeX+1];
        funcion[1] = buf[indeX+2];
        funcion[2] = buf[indeX+3];
        funcion[3] = buf[indeX+4];
        funcion[4] = buf[indeX+5];
        indeX = indeX+6;
        if (strstr(ans, funcion) != NULL){ //Busca CODIGO de funcion en el
mensaje
          Serial.print("funcion encontrada dentro de CODE: ");
Serial.print(funcion);
          for (indeX=6;buf[indeX]!='\0';indeX++){
            if (buf[indeX] == 'Q' && buf[indeX+1] == '0'){
              Message[Messageind] = 'Q';
              Message[Messageind+1] = '0';
              Message[Messageind+2] = '.';
              Messageind = Messageind+3;
              Code2 (3,buf[indeX+3],Output0);
              indeX=indeX+3; // EJ. I0.1 | indeX = 0 => buf[indeX] = 'I' |
indeX = 4 => buf[indeX] = '1'
            } else {
              if ((buf[indeX] == 'Q') && (buf[indeX+1] == '1')){
                Message[Messageind] = 'Q';
                Message[Messageind+1] = '1';
                Message[Messageind+2] = '.';
```

```
                Messageind = Messageind+3;
                Code2 (4,buf[indeX+3],Output1);
                indeX=indeX+3;
              } else {
                if ((buf[indeX] == 'I') && (buf[indeX+1] == '0')){
                  Message[Messageind] = 'I';
                  Message[Messageind+1] = '0';
                  Message[Messageind+2] = '.';
                  Messageind = Messageind+3;
                  Code2 (0,buf[indeX+3],Input0);
                  indeX=indeX+3; // EJ. I0.1 | indeX = 0 => buf[indeX] = 'I'
| indeX = 4 => buf[indeX] = '1'
                } else {
                  if ((buf[indeX] == 'I') && (buf[indeX+1] == '1')){
                    Message[Messageind] = 'I';
                    Message[Messageind+1] = '1';
                    Message[Messageind+2] = '.';
                    Messageind = Messageind+3;
                    Code2 (1,buf[indeX+3],Input1);
                    indeX=indeX+3; // EJ. I0.1 | indeX = 0 => buf[indeX] =
'I' | indeX = 4 => buf[indeX] = '1'
                  } else {
                    if ((buf[indeX] == 'I') && (buf[indeX+1] == '2')){
                      Message[Messageind] = 'I';
                      Message[Messageind+1] = '2';
                      Message[Messageind+2] = '.';
                      Messageind = Messageind+3;
                      Code2 (2,buf[indeX+3],Input2);
                      indeX=indeX+3; // EJ. I0.1 | indeX = 0 => buf[indeX] =
'I' | indeX = 4 => buf[indeX] = '1'
                    } else {
                      if ((buf[indeX] == 'E') && (buf[indeX+1] == '0')){
                        Message[Messageind] = 'I';
                        Message[Messageind+1] = '1';
                        Message[Messageind+2] = '.';
                        Messageind = Messageind+3;
                        indeX=indeX+3; // EJ. I0.1 | indeX = 0 => buf[indeX]
= 'I' | indeX = 4 => buf[indeX] = '1'
                        Code2 (6,buf[indeX+3],Input3);
                      } else {
                        if ((buf[indeX] == 'S') && (buf[indeX+1] == '6')) {
                         // DEFINIR PARA ENCENDER O APAGAR POR X TIEMPO -----
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-----------------
                        }
                      }
                    }
                  }
                }
              }
            }
          } // END FOR
        } else { Serial.println("LA FUNCION NO CONCUERDA");}
      }
      memset(buf,'\0',100); // limpia buf
```

```
      indeX = 0;
    }else{
      if ((c < 126) && (c > 31)){ // verifica que el caracter leido se
encuentre dentro de lo esperado (ignora caracteres de comando)
        buf[indeX] = c;
        indeX++;
      }
    }
    if (c == '*') { break; }
  }
  if (Message[0] == '\0') { strcpy(Message, "Funcion no encontrada"); }
  Message[Messageind] = '\0';
  Archivo.close(); // Cierra el archivo;
  Serial.println();
  Serial.println("FIN DE CODE");
}

void Code2 (byte c1,char c2,char OI [8][16]){
  byte j;
  Message[Messageind] = c2;
  Message[Messageind+1] = ':';
  Messageind = Messageind+2;
  switch (c2){
    case '0': j = 0; break;
    case '1': j = 1; break;
    case '2': j = 2; break;
    case '3': j = 3; break;
    case '4': j = 4; break;
    case '5': j = 5; break;
    case '6': j = 6; break;
    case '7': j = 7; break;
  }
  if (c1 != 6){
    if (Estado[c1][j] == 1) {
      Message[Messageind] = ' ';
      Message[Messageind+1] = 'O';
      Message[Messageind+2] = 'N';
      Messageind = Messageind+3;
    } else {
      Message[Messageind] = 'O';
      Message[Messageind+1] = 'F';
      Message[Messageind+2] = 'F';
      Messageind = Messageind+3;
    }
  } else {
    if (Estado[5][j] == 1) {
      Message[Messageind] = ' ';
      Message[Messageind+1] = 'O';
      Message[Messageind+2] = 'N';
      Messageind = Messageind+3;
    } else {
      Message[Messageind] = 'O';
      Message[Messageind+1] = 'F';
      Message[Messageind+2] = 'F';
      Messageind = Messageind+3;
    }
  }
```

```
  Message[Messageind] = '|';
  Messageind++;
  for (byte m=0;((m<15)||(OI[j][m] == '\0'));m++){
    Message[Messageind] = OI[j][m];
    if (Message[Messageind]>32){ Messageind++; } // Ignora comandos
  }
  Message[Messageind] = ' ';
  Messageind++;
  Message[Messageind+1] = '\0';
}

void ONF(byte S,char ans [250]){
  byte j = 11;
  byte k = 10;
  int t = -1;
  char J; // j pero en CHAR
  char aux[5];
  memset (aux,'\0',5);
  for (byte m = 0; ans[m] != ';' ; m++){
    if ((ans[m] == 'S') && (ans[m-1] == ':')){ // ans: -ON:S1,2;
      j = 10;
      m++;
    }

    switch (ans[m]){
      case '1': k = 1; if (t == -1) { J = '1'; } break;
      case '2': k = 2; if (t == -1) { J = '2'; } break;
      case '3': k = 3; if (t == -1) { J = '3'; } break;
      case '4': k = 4; if (t == -1) { J = '4'; } break;
      case '5': k = 5; if (t == -1) { J = '5'; } break;
      case '6': k = 6; if (t == -1) { J = '6'; } break;
      case '7': k = 7; if (t == -1) { J = '7'; } break;
      case '8': k = 8; break;
      case '9': k = 9; break;
      case '0': k = 0; if (t == -1) { J = '0'; } break;
      default: k = 10; break;
    }
    if (k != 10){ // En caso de leer un numero
      if (t == -1) { // lee numero de salida
        j = k;
        if ((j > -1) && (j < 8)) {
          strcat(Message,"Se ha ");
          if (S == 1) {
            digitalWrite (OUT[j], HIGH);
            strcat(Message,"encendido ");
          } else {
            digitalWrite (OUT[j], LOW);
            strcat(Message,"apagado ");
            TS[j] = 0;
          }
          Serial.print("Valor de j = "); Serial.println(j);
          switch (j){
            case 1: strcat(Message,"la salida 1"+'\0'); break;
            case 2: strcat(Message,"la salida 2"+'\0'); break;
            case 3: strcat(Message,"la salida 3"+'\0'); break;
            case 4: strcat(Message,"la salida 4"+'\0'); break;
            case 5: strcat(Message,"la salida 5"+'\0'); break;
```

```
            case 6: strcat(Message,"la salida 6"+'\0'); break;
            case 7: strcat(Message,"la salida 7"+'\0'); break;
            case 0: strcat(Message,"la salida 0"+'\0'); break;
          }
        }
      }
      if ((t > 0)||(j<5)) { // ya ha leido el primer digito
        t = t*10 + k; // Mueve un digito
        strcat(Message,ans[m]);
        aux[j]=ans[m];
        j++;
      }
      if (t == 0) {
        t = k;
        j = 0;
        strcat(Message," por ");
        aux[j]=ans[m];
        j++;
      }
    }
    if (ans [m] == ','){ //comienza a leer tiempo
      t = 0;
    }
  }
  if (t > 1500) {
    t = 1500;
    aux[0] = '1'; aux[1] = '5'; aux[2] = '0'; aux[3] = '0'; aux[4] = '\0';
  }
  Serial.print("Valor leido de aux : "); Serial.println(aux);
  strcat(Message,aux);
  strcat(Message," minutos. ");
  Serial.print("TIEMPO FINAL DEFINIDO = "); Serial.println(t);
  if ((t == -1) || (t == 0)) { // verifica si se especifico tiempo
    TS [j] = 0;
  } else {
    TS[j] = t * 60000 + millis(); // de minuto a milisegundo
  }
  if (Message[0] != 'S') { // Message = "Se ha..."
    strcat(Message,"La salida indicada no es valida");
  }
  Serial.print("Menaje a enviar: ");
  Serial.println(Message);
}

void OutputARD(){ //Devuelve el estado de las salidas de la placa arduino;
  byte j;
  memset(Message,'\0',201);
  for (j=0;j<8;j++){
    strcat(Message,"Salida");
    switch (j){
      case 1: strcat(Message,"1"); break;
      case 2: strcat(Message,"2"); break;
      case 3: strcat(Message,"3"); break;
      case 4: strcat(Message,"4"); break;
      case 5: strcat(Message,"5"); break;
      case 6: strcat(Message,"6"); break;
      case 7: strcat(Message,"7"); break;
```

```
      case 0: strcat(Message,"0"); break;
    }
    Serial.print("Salida");
    Serial.print(j);
    if (digitalRead(OUT[j])==LOW) {
      strcat(Message,":OFF|");
      Serial.println(": OFF");
    } else {
      strcat(Message,": ON |");
      Serial.println(": ON");
    }
  }
}

// Funciones Pantalla

void LCDclear (byte r) { // Limpia pantalla
  if ((r == 0) || ( r == 4)) {
    lcd.setCursor (0, 0); //Posiciona el cursor en posicion 0,0 (primer
linea)
    lcd.print ("                    "); // Despeja la linea
    lcd.setCursor (0, 0);
  }
  if ((r == 1) || ( r == 4)) {
    lcd.setCursor (0, 1); //Posiciona el cursor en posicion 0,1 (segunda
linea)
    lcd.print ("                    "); // Despeja la linea
    lcd.setCursor (0, 1);
  }
  if ((r == 2) || ( r == 4)) {
    lcd.setCursor (0, 2); //Posiciona el cursor en posicion 0,2 (tercer
linea)
    lcd.print ("                    "); // Despeja la linea
    lcd.setCursor (0, 2);
  }
  if ((r == 3) || ( r == 4)) {
    lcd.setCursor (0, 3); //Posiciona el cursor en posicion 0,3 (cuarta
linea)
    lcd.print ("                    "); // Despeja la linea
    lcd.setCursor (0, 3);
  }
  if (r == 4) {
    lcd.setCursor (0,0);
  }
}

// Funciones SD

void Action(byte val){ //Falta testear--------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------
  char c;
  byte m = 0;
  byte j = 0;
  int indeX = 0;
  char B,C,D,E,F;
```

```
  char Evtext [61];
  char aux[4];
  memset[Evtext,'\0',61];
  Archivo = SD.open("PLCST.txt", FILE_READ);
  char buf[250]; // Puede que sea necesario que sea mayor para nombres largos
  while (Archivo.available()){
    c = Archivo.read();
    if(c == '\n'){ // Busca <cr>
       buf[indeX] = '\0'; // Agrega caracter vacio para indicar fin.
       Serial.println(buf);
       if ((buf[0] == 'Q') && (buf[1] == '0')){
           m = 3;
        }
        if ((buf[0] == 'Q') && (buf[1] == '1')){
           m = 4;
        }
        if ((buf[0] == 'I') && (buf[1] == '0')){
           m = 0;
        }
        if ((buf[0] == 'I') && (buf[1] == '1')){
           m = 1;
        }
        if ((buf[0] == 'I') && (buf[1] == '2')){
           m = 2;
        }
        if ((buf[0] == 'E') && (buf[1] == '0')){
           m = 5;
        }
//-------------------------------------------------------------------------
------AGREGAR FUNCION DE ENTRADAS Y SALIDAS ---------------------------------
-------------
        switch (buf[3]) {
           case '0':
              j = 0;
           break;
           case '1':
              j = 1;
           break;
           case '2':
              j = 2;
           break;
           case '3':
              j = 3;
           break;
           case '4':
              j = 4;
           break;
           case '5':
              j = 5;
           break;
           case '6':
              j = 6;
           break;
           case '7':
              j = 7;
           break;
        }
```

```
        if ((m == mn) && (j == i)){
           Serial.println(" m - mn - j - i");
           Serial.print(" ");
           Serial.print(m);
           Serial.print(" = ");
           Serial.print(" ");
           Serial.print(mn);
           Serial.print(" - ");
           Serial.print(" ");
           Serial.print(j);
           Serial.print(" = ");
           Serial.print(" ");
           Serial.println(i);
           B = buf[7];
           C = buf[9];
           D = buf[11];
           E = buf[13];
           F = buf[15];
           memset(texto,'\0',61); // Vacia texto
           Serial.println("Texto registrado");
           for(indeX = 0; buf[indeX+17] != ';';indeX++){
             texto [indeX] = buf[indeX+17];
             if (buf[indeX+17+1] == ';'){
               texto [indeX+1] = '(';
               if (val == 0) { texto[indeX+2] = '0'; }
                 else { texto[indeX+2] = '1';}
               texto [indeX+3] = ')';
             }
             if (indeX+17 == 74) {
               break; //evita se que sobrepase el valor
             }
           }
           Serial.println("");
           Serial.print("Texto = ");
           Serial.println(texto);
           indeX=0;
           break; // Sale del loop de lectura al encontrar lo que busca
        }
        indeX = 0;
     } else {
        buf[indeX] = c;
        indeX++;
     }
  }
  Archivo.close(); // Cierra el archivo
  if ((m == mn) && (j == i)){
    Serial.println("Se encontro la en bin[i], m == n y j == i");
    // INFORMAR QUE NO SE ENCONTRO LA BUSQUEDA ERROR ----------------------
----------------------------------------------------------------------
  } else {
    Serial.println("ERROR NO SE ENCONTRO BIN[I], m != mn o j != i");
  }
  if (bin [i] != B){ // Verifica si la tiene un valor diferente al habitual 0
o 1. En caso de ser 2 informa.
    Serial.println("El valor de la E/S es diferente al esperado");
     if (C == 'S'){
        Serial.println("Se registra evento");
```

```
        Event (texto,0); // Registra Texto en Archivo Event.txt y actualiza
Eventos // Verificar Testear ------------------------------------------------
----------
        Serial.print("Texto a registrar: ");
        Serial.println(texto);
     }
     if (D == 'S'){ // Envio de mensaje de texto?
        strcat(Message,texto);
        if (E == 'S'){ // Coniderar horario laboral?
          Serial.println("Se registra evento importante");
          Event(texto,1);
          //-------------------------------------------------------------------
---------------- ULTIMO EVENTO-----------------------------------------------
----------
          if (digitalRead(IN[0]) == LOW) { // E1 = LOW ==> No hay nadie en el
lugar de trabajo
              Serial.println("Se deberia enviar mensaje de texto");
              m = sendAT("AT+CMGS=", "OK", 10000); // Envia mensaje
            if (m == "0"){
              strcpy(Evtext,"Error en envio de mensaje de texto, texto a
enviar: ");
              strcat(Evtext,texto);
              Event (Evtext,2);
            }
          }
          // ENVIA SMS SI NO LO ESTA
        }
     }
     if (F != '0'){
        switch (F){
          case '1':
              initialmodbus();
           break;
           case '2':
              // Funciones a programar-------------------------------------
-----------------------------------------------------------------
           break;
           case '3':
              // Funciones a programar-------------------------------------
-----------------------------------------------------------------
           break;
        }
     }
  }
}

void Monitorinit () { // Lee variales del archivo PLCST.txt y ejecuta DivAct
al terminar el renglon ((TESTEADO OK))
  int indeX = 0;
  char c;
  Archivo = SD.open("PLCST.txt", FILE_READ);
  char buf[250]; // Puede que sea necesario que sea mayor para nombres largos
  Serial.println("Leyendo PLCST.TXT");
  while (Archivo.available()){
    c = Archivo.read();
    if(c == '\n'){ // Busca <cr>
       buf[indeX] = '\0'; // Agrega caracter vacio para indicar fin.
```

```
        Serial.println(buf);
        if ((buf[0] == 'Q') && (buf[1] == '0')){
           n = 3;
         }
         if ((buf[0] == 'Q') && (buf[1] == '1')){
           n = 4;
         }
         if ((buf[0] == 'I') && (buf[1] == '0')){
           n = 0;
         }
         if ((buf[0] == 'I') && (buf[1] == '1')){
           n = 1;
         }
         if ((buf[0] == 'I') && (buf[1] == '2')){
           n = 2;
         }
         if ((buf[0] == 'E') && (buf[1] == '0')){
           n = 5;
         }
         switch (buf[3]) {
            case '0': i = 0; break;
            case '1': i = 1; break;
            case '2': i = 2; break;
            case '3': i = 3; break;
            case '4': i = 4; break;
            case '5': i = 5; break;
            case '6': i = 6; break;
            case '7': i = 7; break;
         }
         if (buf[5] == 'S'){
            Monitor[n][i] = 1;
         }
         indeX = 0;
      } else {
         buf[indeX] = c;
         indeX++;
      }
   }
   n = 0;
   i = 0;
   Serial.println("Se ha cargado exitosamente la base de datos");
   Archivo.close(); // Cierra el archivo
}

void SetupFromSD () { // Lee variales del archivo Setup.txt
   Archivo = SD.open("Setup.txt", FILE_READ);
   char buf[100]; // Puede que sea necesario que sea mayor para nombres largos
   char c;
   i = 0;
   Serial.println("Comienzo de carga de variables");
   while (Archivo.available())
   {
     c = Archivo.read();
     if(c == '\n' || c == '\r'){ // Busca <cr> o <lf>
       Divide(buf);
       i = 0;
       buf[i] = '\0'; // Agrega '\0' para indicar final
```

```
    } else {
      buf[i] = c;
      i++;
    }
  }
  Archivo.close(); // Cierra el archivo
  i = 0;
  Serial.print("El numero leido es : "); Serial.println(Number);
  Serial.print("El Codigo leido es : "); Serial.println(Codigo);
}

void Divide(char buf[100]){ // OK
  char* ptrbuf; //Puntero del array buf
  ptrbuf = buf;
  char *Name = strtok(ptrbuf, " ="); // Guarda en "Name" todo aquello que
este antes de " ="
  if(Name){
    char *junk = strtok(NULL, " "); // Limpia caracteres " "
    if(junk){
      char *valu = strtok(NULL, " "); // Guarda todo aquello luego de " "
      if(valu){
        char* Comp = "Number"; // Asigna "Comp"
        if(strstr(Name, Comp) != NULL){// Si Name = Telefono => devuelve 0
          Number = valu;
        }
        Comp = "Codigo"; // Asigna "Comp"
        if(strstr(Name, Comp) != NULL){// Si Name = nueva variable =>
devuelve 0
          Codigo = valu;
        }
//        Comp = "Otra variable" // Si se quiere agregar nuevas variables a
leer
//        if(strcmp(Name, Comp) == 0){// Si Name = nueva variable => devuelve
0
//          Otra variable = valu;
//        }
        for (i=0;i<20;i++){
        valu[i] = '\0'; //Limpia valu
        Name[i] = '\0'; //Limpia Name
        }
      }
    }
  }
}


  //Aqui se agregan nuevas variables a leer en SETUP.txt----------------------
--------------------------------------------------------------------------------


void Event (char Ev[61],byte opc) { // Registra evento en Event.txt
  Archivo = SD.open("Event.txt", FILE_WRITE); //Abre el archivo (lo crea en
caso de no existir) y lo prepara para la escritura
  String DT = GetDT(); // Obtiene fecha y hora - No funciona el RTC
  Serial.print("La hora es : ");
  Serial.println(DT);
  Archivo.print (DT); // Impime la fecha y hora del evento
```

```
  Archivo.print(" : ");
  Archivo.print(Ev);
  Archivo.println("."); // Imprime el informe
  if (opc == 0) {
    Upd8Event(Eventos,DT,Ev);
  } else {
    if (opc == 1) {
      Upd8Event(Eventosimp,DT,Ev);
    } else {
      Upd8Event(Eventos,DT,Ev);
      Upd8Event(Eventosimp,DT,Ev);
    }
  }
  Archivo.close (); // Cierra el archivo
}

void Upd8Event (char Evento [450],String DT, char Ev[61]) { // 0 / 75 / 150 /
225 / 350 / 375 / 450
  Serial.println("Se updatea evento");
  int I; //se utiliza I en lugar de i dado que i esta siendo utilizada en
otra funcion.
  for (I = 0; I<5;I++){
    memmove(Evento+I*75,Evento+75+I*75,75); // Copia bloques de memoria de 75
caracteres
  }
  for (I = 0; I<5;I++) {
    Evento[I+375] = DT [I]; // Agrega DD/MM en Evento [9] de 0 a 4
    Serial.print(Evento[I+375]);
    UltimoEvento[I] = DT [I];
    Evento[I+5+375] = DT [I+12]; //Agrega -hh:m en Evento [9] de 5 a 9
    UltimoEvento [I+5] = DT [I+13];
    if (I != 4) {
      Evento [10+I+375] = DT[17+I]; //Agrega m:ss en Evento [9] de 10 a 14
      UltimoEvento [10+I]; // DD/MM/YYYY - HH:MM:SS
    }
  }
  Evento [389] = '|';
  UltimoEvento[14] = '|';
  for (I = 0; I<60; I++) { // Adiciona el Evento en caso de ver '\0' o de
llegar a 60 caracteres termina.
    if (Ev [I] != '\0'){
      Evento [15+I+375] = Ev [I];
      UltimoEvento [15+I] = Ev [I];
    } else {
      I = 60;
    }
  }
}

// Funcion de pantalla

void upd8screen() {
  LCDclear(4);
  lcd.print (GetDAY());
  lcd.print ("/");
  lcd.print (GetMONTH());
  lcd.setCursor(15,0);
```

```
lcd.print (GetHOUR());
lcd.print (":");
lcd.print (GetMINUTE());
LCDclear(1);
switch (nivel){
  case 0:
    if (func == 1) {
      lcd.print ("   Ultimo evento    ");
      for (di=5;di<10;di++){
        lcd.setCursor(di-5,2);
        lcd.print(UltimoEvento[di]);
      }
      for (di=14;di<29;di++){
        if (UltimoEvento[di]!= '\0'){
          lcd.setCursor(di-9,2);
          lcd.print(UltimoEvento[di]);
        } else { di = 50; }
      }
      for (di=29;di<49;di++){
        if (UltimoEvento[di]!= '\0'){
          lcd.setCursor(di-29,3);
          lcd.print(UltimoEvento[di]);
        } else {di = 50; }
      }
    } else {
      if (index == 0) {
        lcd.print ("->Eventos");
      } else {
        lcd.print ("  Eventos");
      }
      LCDclear(2);
      if (index == 1) {
        lcd.print ("->Eventos Informados");
      } else {
        lcd.print ("  Eventos Informados");
      }
      LCDclear(3);
      if (index == 2) {
        lcd.print ("->Entradas/Salidas");
      } else {
        lcd.print ("  Entradas/Salidas");
      }
    }
  break;
  case 1:
    Menu23(Eventos);
    break;
  case 2:
    Menu23(Eventosimp);
    break;
  case 3:
    if (func == 0){
      LCDclear(0);
      lcd.print ("  Entradas/Salidas  ");
      if (index == 0){
        LCDclear(1);
        lcd.print("  Volver     ");
```

```
  LCDclear(2);
  lcd.print("->Entrada 0.X  ");
  LCDclear(3);
  lcd.print("  Entrada 1.X  ");
} else {
  if (index == 1){
    LCDclear(1);
    lcd.print("  Entrada 0.X  ");
    LCDclear(2);
    lcd.print("->Entrada 1.X  ");
    LCDclear(3);
    lcd.print("  Entrada 2.X  ");
  } else {
    if (index == 2) {
      LCDclear(1);
      lcd.print("  Entrada 1.X ");
      LCDclear(2);
      lcd.print("->Entrada 2.X ");
      LCDclear(3);
      lcd.print("  Salida 0.X  ");
    } else {
      if (index == 3) {
        LCDclear(1);
        lcd.print("  Entrada 2.X ");
        LCDclear(2);
        lcd.print("->Salida 0.X ");
        LCDclear(3);
        lcd.print("  Salida 1.X ");
      } else {
        if (index == 4) {
          LCDclear(1);
          lcd.print("  Salida 0.X");
          LCDclear(2);
          lcd.print("->Salida 1.X");
          LCDclear(3);
          lcd.print("  Entrada 3.X(Ard)");
        } else {
          if (index == 5) {
            LCDclear(1);
            lcd.print("  Salida 1.X");
            LCDclear(2);
            lcd.print("->Entrada 3.X(Ard)");
            LCDclear(3);
            lcd.print("  Salida 2.X(Ard)");
          } else {
            if (index == 6) {
              LCDclear(1);
              lcd.print("  Entrada 3.X(Ard)");
              LCDclear(2);
              lcd.print("->Salida 2.X(Ard)");
              LCDclear(3);
              lcd.print("  Volver");
            } else {
              if (index == 7) {
                LCDclear(1);
                lcd.print("  Salida 2.X(Ard)");
                LCDclear(2);
```

```
                        lcd.print("->Volver");
                        LCDclear(3);
                        lcd.print("  Entrada 0.X");
                      }
                    }
                  }
                }
              }
            }
          }
        } else {
          LCDclear (0);
          if (index == 0) { // Selecciona Input0
            Menu4func (Input0);
          }
          if (index == 1) {
            Menu4func (Input1);
          }
          if (index == 2) {
            Menu4func (Input2);
          }
          if (index == 5) {
            Menu4func (Input3);
          }
          if (index == 3) {
            Menu4func (Output0);
          }
          if (index == 4) {
            Menu4func (Output1);
          }
          if (index == 6) {
            Menu4func (Output2);
          }
        }
      break;
    }
}

void Menu23(char Eve [450]){ // Menu nivel 2 y 3
  byte DN = index;
  if (func == 1){
    LCDclear(3);
    LCDclear(2);
    LCDclear(1);
    LCDclear(0);
    Serial.print("L0 : ");
    for (di = 0; di <14; di++){
      lcd.setCursor(di+3,0);
      lcd.print(Eve[75*index+di]);
      Serial.print(Eve[75*index+di]);
    }
    Serial.println();
    Serial.print("L1 : ");
    for (di = 0; di < 59; di++){
      if (Eve[75*index+di+15] == '\0') {
        di = 60;
```

```
      } else {
        if (di<20) {
          lcd.setCursor(di,1);
        } else {
          if ((di>19) && (di<40)){
            lcd.setCursor(di-20,2);
          } else { lcd.setCursor(di-40,3); }
        }
      }
      lcd.print(Eve[75*index+di+15]);
      Serial.print(Eve[75*index+di+15]);
    }
    Serial.println();
  } else {
    LCDclear(0);
    if (nivel == 1) {
      lcd.print (" Eventos");
    } else {
      lcd.print (" Eventos Informados ");
    }
    if (DN == 0) {
      DN = 6;
    } else {
      DN--;
    }
    for (byte cont = 1; cont<4; cont++){
      LCDclear(cont);
      Serial.println(DN);
      if (cont == 2) {
        lcd.print("->");
      } else {
        lcd.print(DN);
        lcd.print(")");
      }
      if (DN == 6) {
        lcd.print ("Vovler");
      } else {
        for (di = 0; di <5; di++){
          lcd.print(Eve[DN*75+di+6]); //hh:mm
        }
        for (di = 0; di<13; di++){
          if (Eve[DN*75+di+14-75] != '\0') {
            lcd.print(Eve[DN*75+di+14]);//|event
          } else {
          di = 13;
          }
        }
      }
      DN++;
      if (DN == 7) { // DN [1-7] 7 = volver
        DN = 0;
      }
    }
  }
}

void Menu4func (char OI[8][16]) {
```

```
  switch (index){
    case 0: lcd.print("Entrada 0.X"); break;
    case 1: lcd.print("Entrada 1.X"); break;
    case 2: lcd.print("Entrada 2.X"); break;
    case 5: lcd.print("Entrada 3.X"); break;
    case 3: lcd.print("Salida 0.X"); break;
    case 4: lcd.print("Salida 1.X"); break;
    case 6: lcd.print("Salida 2.X"); break;
  }
  if (index <7){
    Menu4func2(OI,Estado[index]); // 0 a 6 Matriz Estado
  }
}

void Menu4func2 (char OI[8][16], byte estado[8]){
  byte DN = dn;
  byte cont;
  lcd.setCursor(12,0);
  Serial.print("Lista de variables: ");
  for (cont = 0;cont<8;cont++){
    lcd.print(estado[cont]);
    Serial.print(estado[cont]);
  }
  if (DN == 0) {
    DN = 7;
  } else {
    DN--;
  }
  for (cont = 1; cont<4; cont++){
    LCDclear(cont);
    lcd.print(DN);
    lcd.print(":");
    if (estado[DN] == 0) {
      lcd.print("OFF|");
    } else {
      lcd.print("ON |");
    }
    for (di=0;di<14;di++){
      if (OI[DN][di] == '\0'){
        di = 14;
      } else {
        lcd.print(OI[DN][di]);
      }
    }
    DN++;
    if (DN == 8) { // DN [0-7]
      DN = 0;
    }
  }
}

// Funciones RTC (Real Time Clock ---------------- agregar que si se conecta
por un Serial Actualice la pila

String str10(String str){
```

```
  if ((str == "0") || (str == "1") || (str == "2") || (str == "3") || (str ==
"4") || (str == "5") || (str == "6") || (str == "7") || (str == "8") || (str
== "9")){
    str = "0"+str;
  }
 return str;
}

String GetMONTH() { //Devuelve un string con la fecha
  String Month;
  DateTime now = rtc.now();
  Month = now.month(), DEC;
  Month = str10(Month);
  return Month;
}

String GetDAY() { //Devuelve un string con la fecha
  String Day;
  DateTime now = rtc.now();
  Day = now.day(), DEC;
  Day = str10(Day);
  return Day;
}

String Getdate() { //Devuelve un string con la fecha
  String Date, Year, Month, Day;
  DateTime now = rtc.now();
  Year = now.year(), DEC;
  Month = GetMONTH();
  Day = GetDAY();
  Date = Day + '/' + Month + '/' + Year;
  return Date;
}

String GetHOUR() { //Devuelve un string con la fecha
  String Hour;
  DateTime now = rtc.now();
  Hour = now.hour(), DEC;
  Hour = str10(Hour);
  return Hour;
}
String GetMINUTE() { //Devuelve un string con la fecha
  String Minute;
  DateTime now = rtc.now();
  Minute = now.minute(), DEC;
  Minute = str10(Minute);
  return Minute;
}
String GetSECOND() { //Devuelve un string con la fecha
  String Second;
  DateTime now = rtc.now();
  Second = now.second(), DEC;
  Second = str10(Second);
  return Second;
}

String Gettime() { //Devuelve un string con la hora
```

```
  String Time, Hour, Minute, Second;
  DateTime now = rtc.now();;
  Hour = GetHOUR();
  Minute = GetMINUTE();
  Second = GetSECOND();
  Time = Hour + ':' + Minute + ':' + Second;
  return Time;
}

String GetDT() { //Devuelve un string con la fecha y hora
  String DT, Date, Time;
  Date = Getdate();
  Time = Gettime();
  DT = Date +' '+'-'+' '+ Time;
  return DT;
}

// INTERRUPCIONES

void inter2() // Arriba
{
  tact = millis();
  Serial.println(tact-tprev);
  Serial.println("ARRIBA");
  if(tact - tprev > 500) { // Evita que se disparen varias veces el mismo
boton
    tprev = tact;
    Serial.println("Boton arriba-------------------------------");
    boton = 1;
  }
}

void inter3() // Abajo
{
  tact = millis();
  Serial.println(tact-tprev);
  Serial.println("ABAJO");
  if(tact - tprev > 500) { // Evita que se disparen varias veces el mismo
boton
    tprev = tact;
    Serial.println("Boton abajo--------------------------------");
    boton = 2;
  }
}

void inter18() // Funcion
{
  tact = millis();
  Serial.println(tact-tprev);
  Serial.println("FUNCION");
  if(tact - tprev > 500) {// Evita que se disparen varias veces el mismo
boton
    tprev = tact;
    Serial.println("Boton Funcion------------------------------");
    boton = 3;
  }
}
```

```
void BOTON() {
  switch (boton) {
    case 1: // Arriba
      Serial.print("Input Arriba");
      Serial.print("Nivel: "); Serial.print(nivel); Serial.print("index: ");
Serial.print(index); Serial.print("Func : "); Serial.print(func);
Serial.print("dn : "); Serial.println(dn);
      boton = 4;
      if ((func == 1) && (nivel != 3 )) {
        func = 0;
      } else {
        switch (nivel){
          case 0: // Menu principal
            if (index == 0) {
              index =2;
            } else {
              index --;
            }
          break;
          case 1: //Eventos
            if (index == 0) {
              index = 6;
            } else {
              index--;
            }
          break;
          case 2: // Eventos imp
            if (index == 0) {
              index = 6;
            } else {
              index--;
            }
          break;
          case 3: // Entradas/Salidas
            if (func == 1) {
              if (dn == 0) {
                dn = 8;
              } else {
                dn--;
              }
            } else {
              if (index == 0){
                index = 6;
              } else {
                index--;
              }
            }
          break;
        }
      }
    break;
    case 2: // Abajo
    Serial.print("Input Abajo");
    Serial.print("Nivel: "); Serial.print(nivel); Serial.print("index: ");
Serial.print(index); Serial.print("Func : "); Serial.print(func);
Serial.print("dn : "); Serial.println(dn);
```

```
    boton = 4;
      if ((func == 1) && (nivel != 3 )) {
        func = 0;
      } else {
        switch (nivel){
          case 0: // Menu principal
            if (index == 2) {
              index =0;
            } else {
              index ++;
            }
          break;
          case 1: //Eventos
            if (index == 6) {
              index = 0;
            } else {
              index++;
            }
          break;
          case 2: // Eventos imp
            if (index == 6) {
              index = 0;
            } else {
              index++;
            }
          break;
          case 3: // Entradas/Salidas
            if (func == 1) {
              if (dn == 8) {
                dn = 0;
              } else {
                dn++;
              }
            } else {
              if (index == 7){
                index = 0;
              } else {
                index++;
              }
            }
          break;
        }
      }
    break;
    case 3: // Funcion
      Serial.print("input Funcion");
      Serial.print("Nivel: "); Serial.print(nivel); Serial.print("index: ");
Serial.print(index); Serial.print("Func : "); Serial.print(func);
Serial.print("dn : "); Serial.println(dn);
      boton = 4;
      switch (nivel) {
        case 0:
          if (func == 1) {
            func = 0;
          } else {
            if (index == 0) {
              nivel = 1;
```

```
      index = 5;
      func = 0;
    } else {
      if (index == 1) {
        nivel = 2;
        index = 5;
        func = 0;
      } else { // nivel == 2
        nivel = 3;
        index = 0;
        func = 0;
        dn = 1;
      }
    }
  }
break;
case 1:
  if (func == 1) {
    func = 0;
  } else {
    if (index == 6){
      nivel = 0;
      index = 0;
      func = 0;
    } else {
      func = 1;
    }
  }
break;
case 2:
  if (func == 1) {
    func = 0;
  } else {
    if (index == 6){
      nivel = 0;
      index = 1;
      func = 0;
    } else {
      func = 1;
    }
  }
break;
case 3:
  if (func == 0) {
    if (index == 7) {
      nivel = 0;
      index = 2;
      func = 0;
    } else {
      func = 1;
    }
  } else {
    func = 0;
  }
break;
case 4:
break;
```

```
      }
    break;
  }
  if (boton == 4) {
    Serial.println("Boton = 4");
    Serial.print("Nivel: "); Serial.print(nivel); Serial.print("index: ");
Serial.print(index); Serial.print("Func : "); Serial.print(func);
Serial.print("dn : "); Serial.println(dn);
    upd8screen();
    Serial.print("Nivel: "); Serial.print(nivel); Serial.print("index: ");
Serial.print(index); Serial.print("Func : "); Serial.print(func);
Serial.print("dn : "); Serial.println(dn);
    boton = 5;
  } else {
    if ((nivel == 0) && (func == 0)){
      LCDclear(0);
      lcd.print (GetDAY());
      lcd.print ("/");
      lcd.print (GetMONTH());
      lcd.setCursor(15,0);
      lcd.print (GetHOUR());
      lcd.print (":");
      lcd.print (GetMINUTE());
    }
  }
}

void TIMER () {
  if (millis() - TI > 5000){
    TI = millis();
    if (millis() - tprev > 20000){ //verifica que no se haya presionado
ningun boton en 20 segundos
      func = 1;
      nivel = 0;
      index = 0;
      upd8screen();
    }
  }
  for (byte i=0;i<8;i++){
    if ((millis()>TS[i]) && (TS[i] != 0) && (millis()-TS[i] < 300000000)){
//verifica que sea mayor y que no haya ocurrido un overflow de millis
      digitalWrite (OUT[i], LOW);
      TS[i] = 0;
    }
  }
}
```