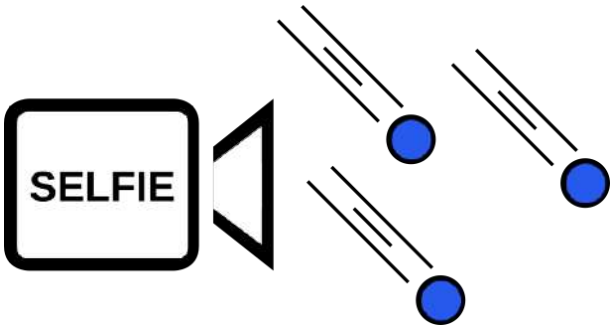
Detector de Radiación Ionizante para aplicaciones satelitales



Sr. Mariano Rolón
Estudiante

Sr. Mathías Sebastian García
Estudiante

Dr. Ing. Maximiliano Antonelli
Director

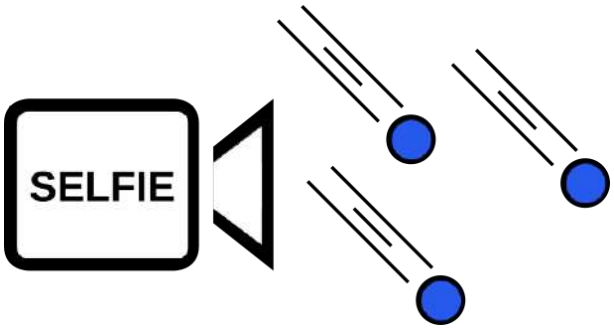
Dr. Ing. Claudio Marcelo González
Co-director

15 de diciembre de 2021

Laboratorio de Sistemas Caóticos

Departamento de Ingeniería Electrónica y Computación
Facultad de Ingeniería
Universidad Nacional de Mar del Plata
Argentina

Detector de Radiación Ionizante para aplicaciones satelitales



Sr. Mariano Rolón
Estudiante

Sr. Mathías Sebastian García
Estudiante

Dr. Ing. Maximiliano Antonelli
Director

Dr. Ing. Claudio Marcelo González
Co-director

15 de diciembre de 2021

Laboratorio de Sistemas Caóticos

Departamento de Ingeniería Electrónica y Computación
Facultad de Ingeniería
Universidad Nacional de Mar del Plata
Argentina

A mis padres, Marcela y Gabriel.
A mis hermanas, Camila y Florencia.
A mi abuelo, Nicolino.
Mariano.

A mis padres, Silvia y Eduardo.
A la memoria de mi abuelo, Aldo.
A mi novia, Abril.
Mathias.

Agradecimientos

A nuestro Director Maximiliano Antonelli, por toda su paciencia, predisposición y compromiso. Todos sus aportes, comentarios y puntos de vista fueron fundamentales para el desarrollo de este proyecto final.

A nuestro Co-Director Claudio González, por sus valiosos comentarios.

A Luciana De Micco, por ayudarnos a configurar las licencias y por sus excelentes aportes en conocimiento de HLS. Sin esta herramienta muchos bloques no se podrían haber llevado a cabo.

A Martín Pérez y José Lipovetzky, por acompañarnos a lo largo de este proyecto final, aclarando nuestras dudas y proveyéndonos tanto de conocimientos, como de materiales.

A nuestras familias, por animarnos siempre que las cosas no salían como queríamos.

Resumen

En este trabajo se presenta el desarrollo e implementación de un sistema de adquisición y detección de eventos para su utilización en un detector de radiación ionizante. Dicho sistema fue implementado en FPGA y forma parte de un detector de radiación basado en sensores de imagen comerciales CMOS.

Este detector de radiación fue diseñado para la medición de flujo de partículas y *linear energy transfer* (LET) en constelaciones de satélites.

El uso de sensores de imagen comerciales tiene importantes ventajas con respecto a los circuitos integrados diseñados *ad hoc*, como el bajo costo, disponibilidad y proceso de diseño más simple.

Palabras clave – Diseño digital, Radiación ionizante, Aplicaciones satelitales.

Abstract

This work presents the development and implementation of an event acquisition and detection system for use in an ionizing radiation detector. This system was implemented in FPGA and is part of a radiation detector based on commercial CMOS image sensors.

This radiation detector was designed for the measurement of particle flux and *linear energy transfer* (LET) in satellite constellations.

The use of commercial image sensors has important advantages over integrated circuits designed *ad hoc*, such as low cost, availability and simpler design process.

Keywords – Digital design, Ionizing radiation, Satellite applications.

Índice general

Capítulo 1

Introducción

La detección de radiación ionizante tiene importantes aplicaciones en radioprotección, aplicaciones médicas, técnicas experimentales para el estudio de la materia condensada, aplicaciones industriales, seguridad en fronteras, vigilancia ambiental, etc. Los sistemas tradicionales para la detección de radiación ionizante están constituidos por tubos Geiger, cámaras de ionización, detectores de superficie, sistemas centelladores, fotomultiplicadores, etc. En general, estos sistemas ocupan grandes volúmenes, poseen costos elevados, y presentan consumos de potencia muy altos, estas desventajas reducen su campo de aplicación [1].

Fuera de la atmósfera existen niveles de radiación significativamente mayores a las que se observan a nivel terrestre. En el espacio, la radiación está compuesta por partículas atrapadas en el campo magnético de la tierra, partículas generadas durante tormentas solares, y rayos cósmicos que son protones e iones pesados de alta energía. Los componentes electrónicos pueden sufrir fallas cuando son expuestos a partículas con altas energías.

Durante años, las agencias espaciales de diversos países han realizado experimentos para monitorear la radiación espacial y estudiar los efectos en dispositivos electrónicos. Se busca colectar información para validar los datos obtenidos en las irradiaciones realizadas en tierra y para probar el diseño de los procedimientos de corrección de errores [2]. Otra aplicación frecuente en el espacio, es el monitoreo de los niveles de radiación para condicionar la operación de ciertos equipos electrónicos o funciones. En algunos equipos dentro de satélites, el daño producido por radiación disminuye considerablemente cuando se encuentran apagados. De esta forma, cuando existen niveles altos de radiación, se desconectan ciertos equipos para prolongar su vida útil.

Recientemente, surgieron soluciones para la detección de radiación ionizante basada en sensores de imagen comerciales fabricados en tecnología CMOS [3], [4]. Este tipo de sensor de imagen es ampliamente utilizado en la industria electrónica de consumo masivo como cámaras de fotografía, teléfonos celulares, computadoras, etc. Al interactuar con el sensor, una partícula ionizante deposita carga, observándose un conjunto de píxeles que poseen intensidades mayores que el resto de la imagen. A este conjunto de píxeles causado por radiación se lo denomina evento y pueden tener distintas formas e intensidades. En la figura 1.1 se pueden observar los distintos eventos causados por una fuente de ^{241}Americio. Haciendo uso de estos sensores, se han desarrollado detectores de radiación de bajo costo con una resolución espacial que supera a la que poseen la mayoría de los detectores tradicionales y una alta resolución en dosis [5].

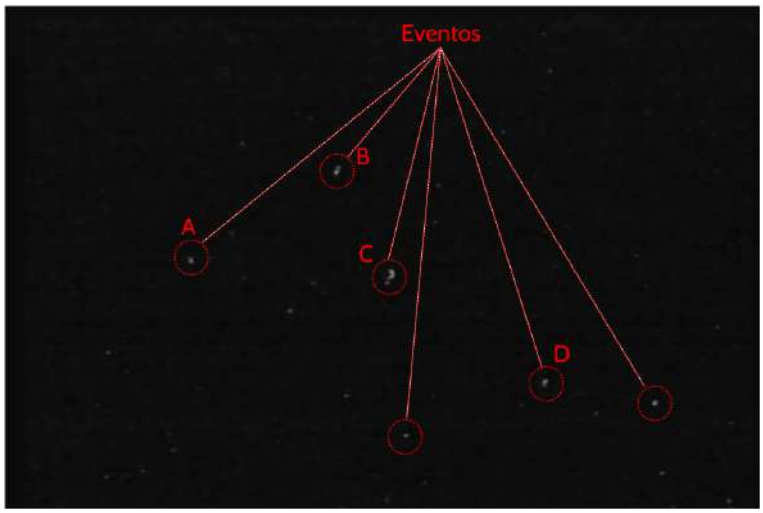


Figura 1.1: Eventos

El uso de sensores comerciales tiene importantes ventajas con respecto a los circuitos integrados diseñados *ad hoc* para una aplicación específica:

- Bajo costo de cada chip.
- Disponibilidad en el mercado y fácil adquisición.
- Abundante documentación.
- Soporte de los fabricantes.
- Por lo general, este tipo de circuitos integrados son *System On Chip*, es decir que poseen diferentes bloques (ADC, Memorias, etc.) que sirven para efectuar muchas operaciones dentro del mismo, facilitando el diseño de la electrónica de lectura que se utiliza para la adquisición de imágenes.

Este proyecto final trata sobre el desarrollo de un instrumento de medición para aplicaciones satelitales [6], [7]. Para esto, se utilizó como base la tesis de maestría del Dr. Ing. Martín Pérez [8], donde analiza la factibilidad del uso de sensores CMOS para la detección de radiación y evalúa la respuesta de los mismos frente a distintos tipos de radiación. A partir de la radiación recibida, el instrumento produce un histograma, donde la LET se informa en el eje de abscisas y el número de eventos con ese valor en el eje de ordenadas.

La necesidad y los requerimientos surgen del Laboratorio de Bajas Temperaturas del Instituto Balseiro, debido al interés de contar con un instrumento de medición de radiación barato y que pueda ir incorporado en un satélite.

El proyecto LabOSat fue el encargado de informar las restricciones, principalmente, en cuanto a consumo de potencia. Esto es debido a que el instrumento a desarrollar iría montado sobre su plataforma satelital.

Capítulo 2

Anteproyecto

El producto desarrollado es un instrumento para la medición de radiación ionizante en entornos espaciales, consiste en un sensor tipo COTS CMOS comercial y una FPGA donde se realiza la clasificación de las partículas que lo impactan. Internamente, se lo denominó SELFIE, acrónimo de **S**canner **El**ectrónico de **F**lujo de **I**ones **E**spaciales.

A continuación, se presentan secciones que detallan cuestiones de requerimientos y constructivas, alternativas de diseño y, para cerrar este capítulo, una introducción al plan de proyecto. Para leer con más detalles lo mencionado anteriormente, se invita a leer el **Apéndice A**, el cual comprende el plan de proyecto en su totalidad, y el **Apéndice B**, que contiene la especificación de requerimientos.

2.1. Requerimientos y aspectos constructivos

El instrumento posee tres requisitos fundamentales, detectar eventos, modificar la configuración e informar el estado del sensor. Sin embargo, con el fin de realizar pruebas con distintos tipos de radiación y verificar su funcionamiento en tierra, nacen dos requerimientos adicionales. El primero de ellos es la realización de dos modos de funcionamiento, automático y de pruebas. El automático, como su nombre lo indica, trabaja de manera autónoma capturando imágenes, procesándolas y almacenando los resultados en una memoria hasta que el satélite pueda transmitir. Una vez que son enviados a tierra, el procedimiento comienza nuevamente. En el modo de pruebas, quien esté controlando el instrumento debe poder capturar una cantidad arbitraria de imágenes y recuperar los datos del histograma de *LET* realizado. Por otro lado, el segundo de estos requisitos es poder capturar la imagen para enviarla a una computadora, donde se pueda reconstruir con el propósito de comparar los resultados del instrumento con un análisis realizado en dicho equipo.

En cuanto a la construcción del instrumento, en primera instancia se tiene en cuenta el uso de redundancias para mitigar los efectos de SET y SEE [9], por lo tanto la implementación del algoritmo debe ocupar la menor cantidad de área posible para poder ser replicado al menos tres veces. Adicionalmente, debe utilizarse tan poca memoria como sea posible, ya que en las FPGAs tipo *RADHARD* este recurso es muy escaso.

2.2. Alternativas de implementación de filtro de píxeles muertos

En el análisis de eventos hay que tener en cuenta la aparición de píxeles muertos. Luego de ser irradiado el sensor, puede suceder que algunos píxeles sobre los que impactaron las partículas radiactivas queden encendidos indefinidamente. Estos no representan un evento. Por lo tanto, se debe tener en cuenta su análisis ya que, cuanto mayor tiempo está el sensor expuesto a radiación, mayor cantidad de píxeles muertos aparecerán.

El principal problema del análisis de píxeles muertos es el uso de memoria, se analizaron tres posibles diseños, los cuales fueron:

- Una solución directa se presenta en la figura **2.1**, donde se utiliza un filtro autorregresivo para eliminarlos, a costa de requerir el almacenamiento de dos frames completos.

- Una optimización de este método se muestra en la figura **2.2**, donde se identifica la posición de los píxeles muertos y se le restan al frame entrante. La ventaja en este diseño es que solo requiere el almacenamiento de un frame completo.

- La tercera opción, que se puede apreciar en la figura **2.3**, toma el frame ingresante, lo procesa y en una etapa posterior clasifica los datos. Esta configuración prescinde completamente del almacenamiento de frames.

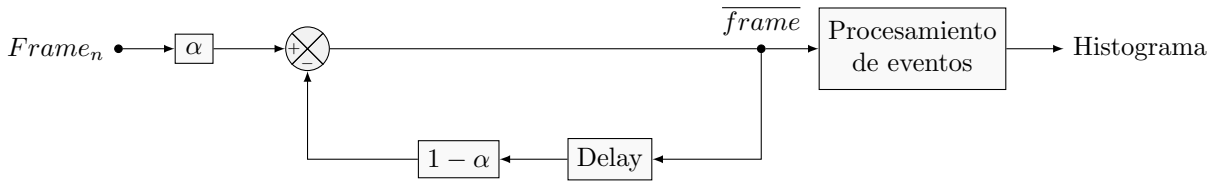


Figura 2.1: Alternativa 1 - Filtro autorregresivo

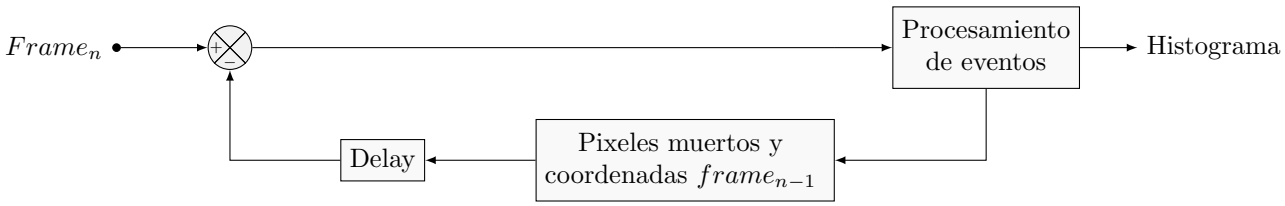


Figura 2.2: Alternativa 2 - Resta por coordenadas

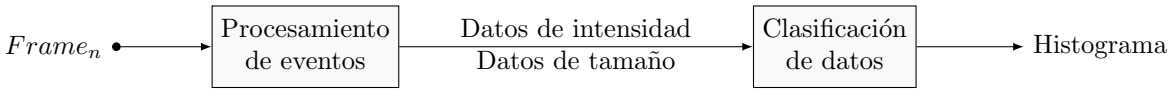


Figura 2.3: Alternativa 3 - Análisis a posteriori

En función de los siguientes criterios, se optó por implementar el tercer diseño.

- Memoria: El primer diseño requiere una cantidad excesiva debido al guardado de dos frames completos. El segundo, si bien lo disminuye, sigue siendo más de lo disponible. El tercero cumple con las condiciones de memoria, a costa de considerar píxeles muertos a eventos compuestos por un solo píxel.
- Reproducibilidad: Además del bajo consumo de memoria, el tercer diseño requiere menos lógica de control. Esto implica una menor cantidad de área consumida. Por lo tanto, brinda más libertad a la hora de elegir la FPGA a utilizar.

2.3. Introducción al plan de proyecto

El desarrollo comenzó en 2019 utilizando una FPGA Zedboard Zynq7000 proporcionada por el Laboratorio de Sistemas Caóticos, así como los equipos necesarios. Las licencias para el uso del software Xilinx Vivado fueron provistas por la Dr. Ing. Luciana De Micco, quien también proporcionó conocimiento técnico en HLS.

Inicialmente, no se partió con un cronograma establecido. Se comenzó leyendo sobre los efectos de radiación mencionados previamente ([9]) y la tésis de maestría [8]. En función de ello se diagramó la transferencia de imágenes desde la FPGA hacia una computadora utilizando el protocolo AXI de Xilinx, y luego el bloque de configuración del sensor. Recién a partir de abril de 2021 se realizó una estimación de tiempo y proyectó un diagrama de Gantt, el cual se presenta en la Figura **2.4**. La fecha de inicio es el 28 de abril de 2021 y la fecha tentativa de finalización es el 18 de agosto de 2021.

Para la realización del proyecto, además, fueron partícipes necesarios los Dr. Ing. Martín Pérez y Dr. Ing. José Lipovetzky del Laboratorio de Bajas Temperaturas del Instituto Balseiro. Ellos proveyeron no solo la información técnica necesaria, si no también los sensores utilizados, ya que estos últimos necesitan un tratamiento especial para que respondan a la radiación.

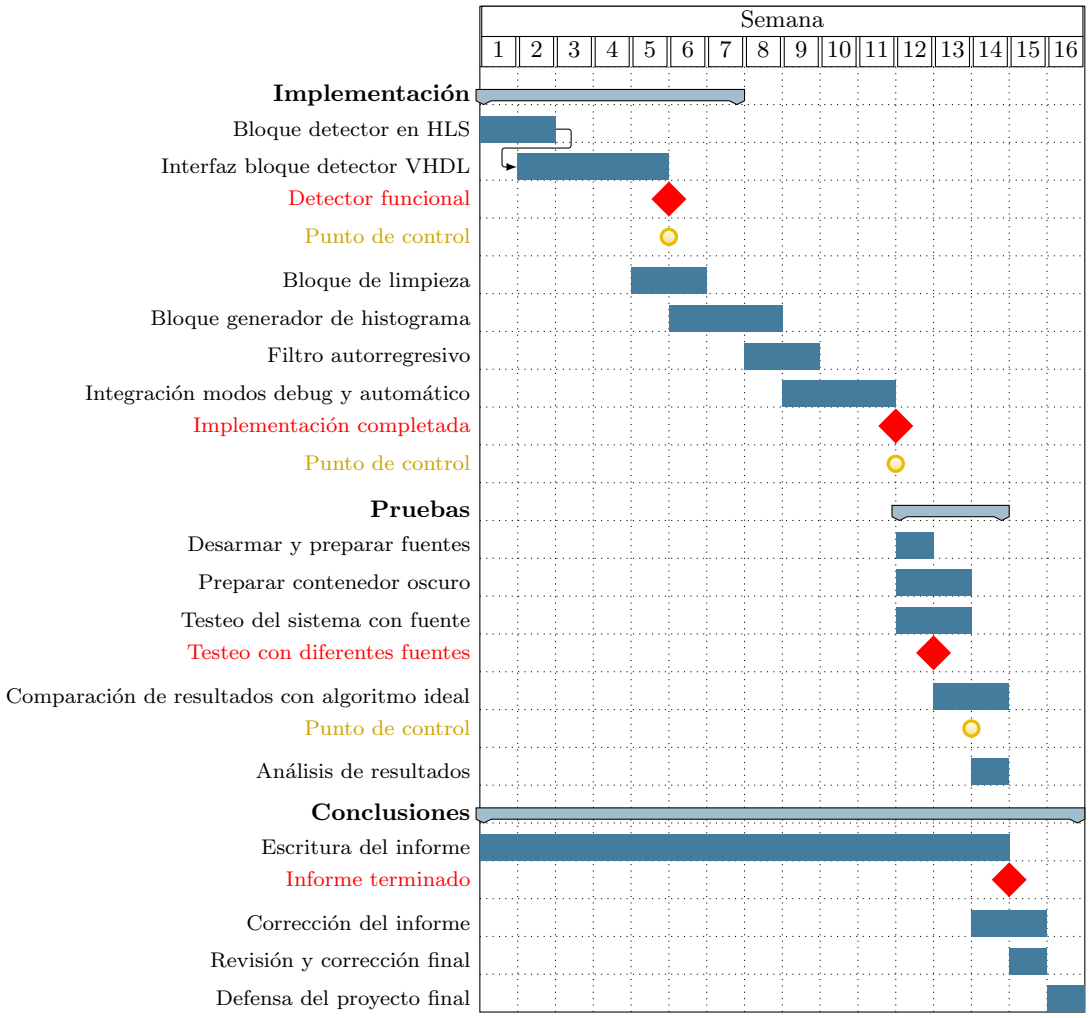


Figura 2.4: Diagrama de Gantt

Capítulo 3

Proyecto

3.1. Primera etapa: Interfaces

Para comenzar, se tomó la PCB provista para el sensor y se soldaron los componentes necesarios. Las resistencias y capacitores no presentaron dificultades. Por otro lado, las inductancias y el zócalo sobre el cual se monta el sensor son de montaje superficial, lo que volvió el proceso de soldado más complejo. La placa terminada se puede observar en la figura **3.1**.

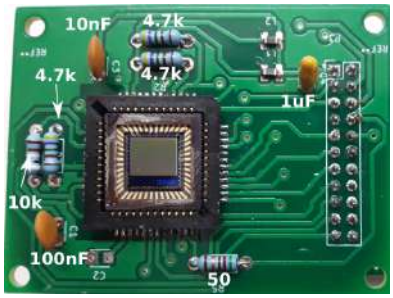


Figura 3.1: PCB con componentes y zócalo soldado

En esta etapa del proyecto se definieron los siguientes objetivos:

- Obtener una imagen del sensor y reconstruirla.
- Escribir y leer los registros de configuración del sensor.

Estos se corresponden con los requisitos funcionales **RF01** y **RF02**, mencionados en los Apéndices **B** y **C**.

Por lo tanto, se partió de la hoja de datos del sensor [10] para recabar información sobre su funcionamiento junto con las señales de interfaz que proporciona. Posteriormente, se conectó la placa de interfaz del sensor a los puertos PMOD de la ZedBoard, como se puede apreciar en la figura **3.2**.

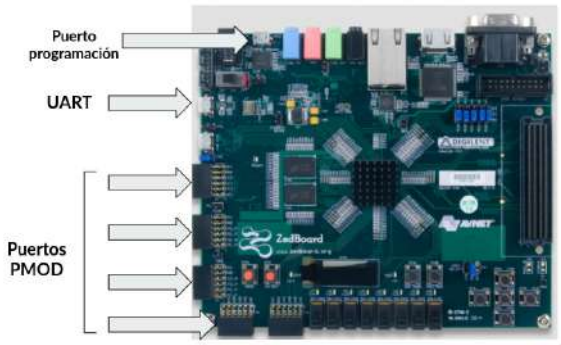


Figura 3.2: Puertos Zedboard

Siguiendo el orden presentado previamente, la primera cuestión a resolver fue la extracción de datos. Acorde a [10], el sensor recibe una señal de clock con frecuencia máxima 48 MHz y devuelve tres señales:

- **Frame Valid**: Indica que el frame a recibir es válido.

- **Line Valid**: Indica que la fila de píxeles es válida.
- **PIXCLK**: Señal de clock utilizada para registrar los datos, su frecuencia es igual al clock que recibe.

Las dos primeras señales mencionadas se emplean para saber cuando los datos son válidos, ya que la imagen se encuentra rodeada de zonas de blanqueo horizontal y vertical. En otras palabras, los datos son legítimos cuando *Frame Valid* y *Line Valid* se encuentran en alto, además debe ser registrado en el flanco descendente de *PIXCLK*. Cada dato es entregado en formato serie y consta de 10 bits en paralelo. Sin embargo, para simplificar la lógica y uso de espacio, se utilizaron los 8 bits más significativos.

Debido a que resulta necesario que, para la alternativa elegida en el anteproyecto, el algoritmo a implementar trabaje en tiempo real, se configura una frecuencia de trabajo de 100 MHz en la FPGA y se coloca un conversor de frecuencia para enviar una señal de clock de 10 MHz al sensor. Lo cual otorga un total de 10 ciclos de clock para procesar el dato recibido. Por otro lado, debido al cruce de dominio de clock (CCD), tanto los datos como las señales de entrada resultan metaestables. Para resolver este inconveniente, se debió implementar un sincronizador en la etapa de captura ([**11**]).

Una vez diseñado el sistema de captura, el siguiente paso consistía en enviar la imagen hacia una computadora. El primer enfoque fue utilizar los puertos GPIO (*General Purpose Input/Output*) que comunican la parte lógica (PL) con el processing system (PS) de la FPGA. A pesar de los múltiples intentos, esto no fue posible ya que estos puertos son lentos y no pueden trabajar a 10 MHz. A continuación, en la figura **3.3**, se muestra un esquema del conexionado utilizado:

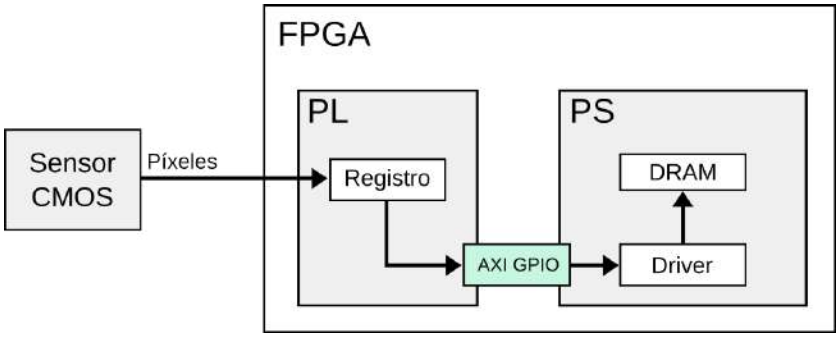


Figura 3.3: Primer enfoque

El segundo enfoque fue utilizar transferencias DMA (*Direct Memory Access*). La principal ventaja consiste en que no es necesario utilizar la CPU para manejarlas. Esto se traduce en un considerable aumento en la velocidad de transferencia. La figura **3.4** muestra el enfoque mencionado.

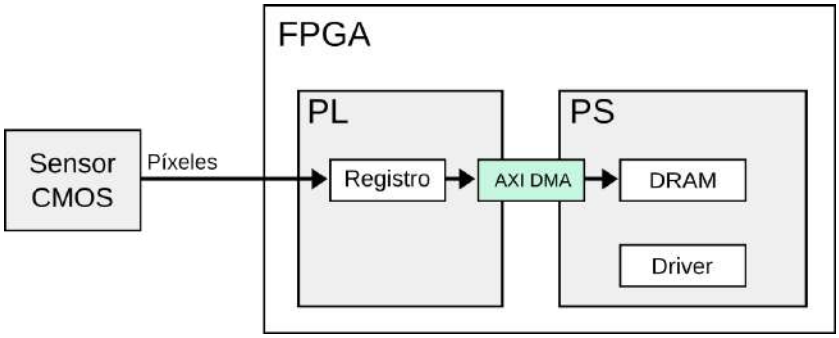


Figura 3.4: Segundo enfoque

Finalmente, tras una intensa lectura sobre el protocolo AXI, los bloques necesarios para utilizar transferencias DMA, su configuración y señales intermediarias requeridas, se logró obtener la primera imagen, que se puede observar en la figura **3.5**.

Una vez finalizada la recepción y transmisión de la imagen, el siguiente paso fue el desarrollo de un módulo que permita la lectura y escritura de los registros de configuración. Según indica el fabricante ([**10**]), el protocolo propietario posee una enorme similitud al estándar I2C, por lo tanto se decidió adaptar un módulo ya funcional que implemente dicho estándar. La descripción en detalle de la máquina de estados que implementa el acondicionamiento mencionado se encuentra en el **Apéndice D**.

Con el fin de corroborar el funcionamiento del bloque se realizaron dos tests según lo especificado en el **Apéndice E**. El primero fue leer el registro 0x00, el cual contiene un valor fijo propio de los sensores (0x8431).

Figura 3.5: Primera captura realizada

El segundo fue poner al sensor en modo de pruebas, esto produce un patrón de salida, el cual se puede apreciar en la figura **3.6**.

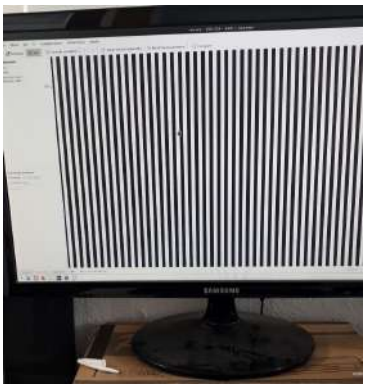


Figura 3.6: Patrón de prueba

Concluida esta implementación inicial, se propuso el diseño de una interfaz que facilite tanto la captura de imágenes, como la conexión por puerto UART y la lectura/escritura de registros. La interfaz se programó en Python y se presenta en la figura **3.7**. Originalmente se utilizó la librería Tkinter y constaba de solo dos pestañas, las cuales contenían una versión más primitiva de las tres primeras pestañas descriptas en los Apéndices **C** y **D**.

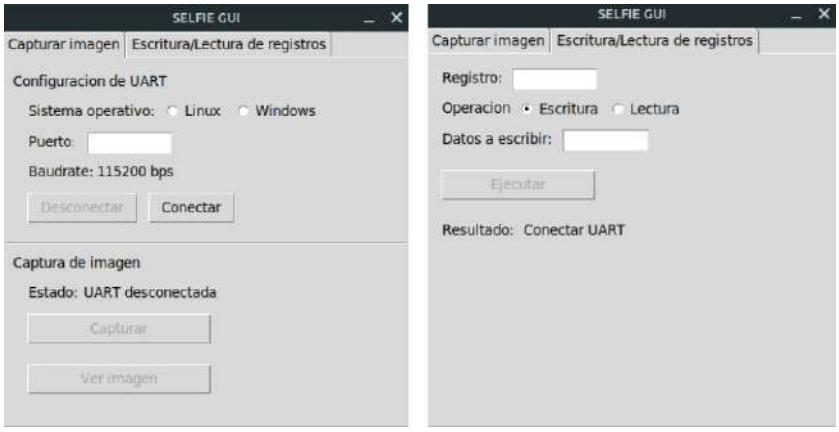


Figura 3.7: Interfaz inicial

Tras realizar múltiples pruebas en cuanto a captura y reconstrucción de imágenes, así como de modificaciones de registros, y con el uso de un lente y un ajuste iterativo de la ganancia del sensor, se logró obtener una imagen considerablemente más nítida, la cual es presentada en la figura **3.8**.

Figura 3.8: Captura mejorada

3.2. Segunda etapa: Bloques de procesamiento

Una vez que las cuestiones relacionadas al interfaceado para la configuración del sensor, captura y transmisión de imágenes fue resulta, se procedió al diseño de los algoritmos. En el inicio de esta etapa, se propusieron tres objetivos claros:

- Algoritmo de detección: se partió en tres secciones. Primero en alto nivel, luego se simuló un bajo nivel y finalmente se implementó en HLS.
- Algoritmo de generación de histograma: tras analizar sus funciones, se decidió implementarlo directamente en HLS.
- Acabado de la interfaz: tomar la versión reducida que se hizo para probar el interfaceado y completar su diseño según se especifica en los Apéndices **B** y **C**.

Para poder realizar mediciones de radiación usando sensores de imagen CMOS, primero se le debe remover el vidrio protector para que las partículas ionizantes puedan interactuar mas fácil con las junturas PN de los transistores. Como se mencionó en la introducción, los impactos de las partículas ionizantes sobre estas junturas producen lo que se denomina *evento*, y cada una de estas tiene una LET asociada. La figura **1.1**, correspondiente a una irradiación del sensor con una fuente de ^{241}Americio, muestra cómo se ven tales eventos.

Según se describe en el **RF03**, el instrumento debe ser capaz de realizar un análisis de cantidad, tamaño e intensidad de estos eventos a partir de la lectura de los píxeles. Siendo la LET el parámetro al cual denominamos intensidad, ya que desde el punto de vista del eléctrico, representará un valor en el rango [0, 255]. Luego, generando un histograma de intensidades, se puede identificar qué partículas impactaron al sensor.

Por otro lado, debido al ruido propio del sensor, es necesario definir un umbral. Los valores por debajo de este umbral son descartados inmediatamente, mientras que aquellos por encima deben ser clasificados como eventos, siendo uno nuevo o parte de otro ya conocido. Para obtener este umbral se configura una ganancia y se captura una ráfaga de imágenes en condiciones de oscuridad. Luego, se hace un ajuste de los datos con una campana de Gauss obteniéndose el valor medio y la desviación estándar. El valor umbral se considera en la media más cinco sigmas, $\bar{\mu} + 5\sigma$.

3.2.1. Detección

Previo a diseñar el algoritmo de detección a implementar sobre la FPGA, se realizó una versión ideal implementada en Python. En otras palabras, uno que trabaje con la imagen completa. Esto se hizo debido a que las imágenes de radiación que fueron provistas para probar el algoritmo eran de 480x720 y 1280x1024, realizar una comparación manual se vuelve en extremo tedioso. Para verificar que su funcionamiento sea el adecuado, se utilizaron imágenes de 5x5, 10x10 y 25x25 que fueron corroboradas de manera manual, según se indica en el ID3 del Apéndice **E**.

A continuación, se procedió a diseñar el algoritmo real de detección. La versión inicial, siendo de alto nivel, se desarrolló en MATLAB. Esta etapa fue relativamente corta, ya que se pudieron obtener resultados aceptables rápidamente. La próxima versión se continuó en Python, donde se simuló en bajo nivel; por ejemplo, no usando funciones de búsqueda en vectores. El objetivo era imitar la implementación en hardware, la cual requiere poco consumo de potencia y que trabaje en tiempo real. Además, durante este adaptación se arregló un error severo detectado en su predecesora, el cual en ocasiones producía que los eventos se dividan en dos o más.

Acorde al ID4 del Apéndice **E**, se utilizaron imágenes de 5x5, 10x10 y 25x25 en una verificación manual de los resultados. Al finalizar con este diseño, se llegó a dos algoritmos que funcionalmente eran iguales, pero sus implementaciones resultaban diametralmente opuestas, como se puede observar en el cuadro **3.1**.

Característica	**Algoritmo ideal**	**Algoritmo real**
Optimización	Baja	Elevada
Uso de memoria	Elevado	Bajo
Consumo de recursos	Elevado	Bajo
Velocidad	Lento	Rápido (paralelismo)
Errores	Sin errores	Existe un porcentaje muy bajo de error

Cuadro 3.1: Comparación de algoritmos

Luego, siguiendo con el ID5 del plan de pruebas, se enviaron las mismas imágenes a los dos algoritmos, se compararon los resultados y se obtuvo un porcentaje de acierto. Como se presenta visualmente en la figura **3.9**.

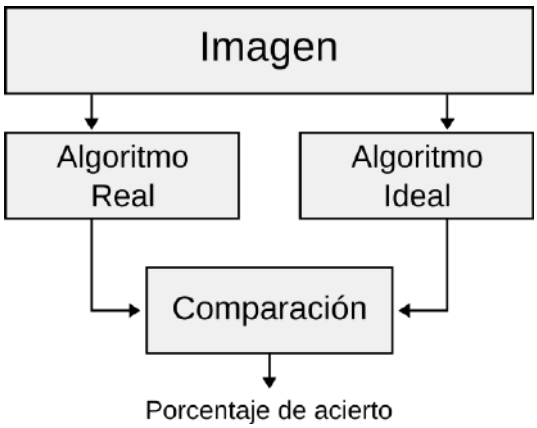
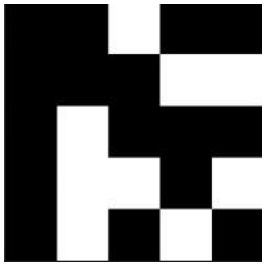


Figura 3.9: Validación de algoritmos

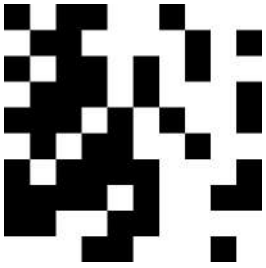
Para este proceso, se utilizaron tanto imágenes generadas de manera aleatoria como de capturas reales. Estas últimas fueron extraídas de los vídeos de radiación provistos por el Instituto Balseiro. A los algoritmos se les envió:

- 100 imágenes de 5x5 aleatorias.
- 100 imágenes de 10x10 aleatorias.
- 100 imágenes de 50x50 aleatorias.
- 628 imágenes de capturas reales.

En la figura **3.10** se presentan ejemplos de las imágenes utilizadas.

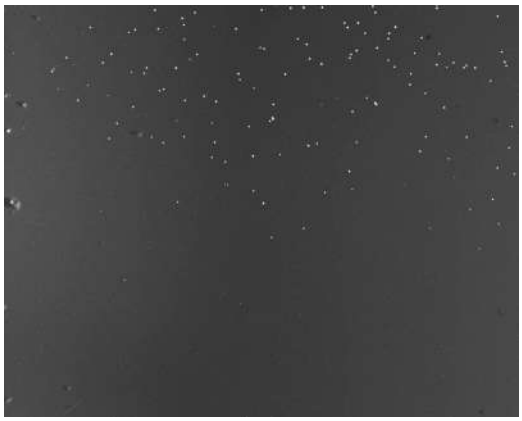


(a) Aleatoria de 5x5



(b) Aleatoria de 10x10

(c) Aleatoria de 50x50



(d) Captura de radiación real

Figura 3.10: Ejemplos de imágenes utilizadas para el test

Como resultado de estas pruebas, se obtuvo que el algoritmo real presentaba un error promedio de 8.82 % en las imágenes aleatorias. La causa de este era una disposición en particular que se presentaba cuando los eventos estaban conectados por un único píxel en diagonal, como se puede apreciar en la figura **3.11**. Si bien este error fue corregido en la versión final, en la práctica no suelen aparecer. De hecho, en las imágenes de irradiación

extraídas de los vídeos, no se observan eventos con estas características. En las imágenes de capturas reales, el error absoluto, en porcentaje, fue de 2 %.

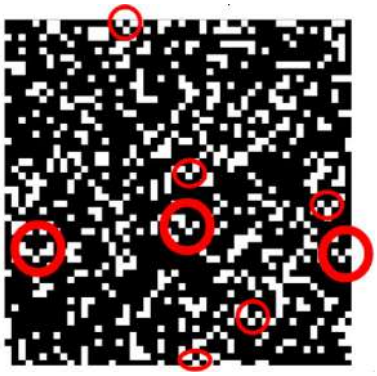


Figura 3.11: Disposición problemática

Inmediatamente tras finalizar la versión en bajo nivel del algoritmo de detección, se comenzó con la lectura de información relacionada al diseño de hardware en High Level Synthesis (HLS) [**12**]. El principal objetivo era comprender los aspectos relacionados a cómo la herramienta infiere y produce RTL. HLS permite escribir código en una versión reducida del lenguaje C++ y el software infiere, de manera automática, estructuras de RTL. Por lo tanto, resulta particularmente útil para desarrollar implementaciones en hardware de algoritmos complejos y que podría insumir una cantidad de tiempo considerable realizarlo en VHDL.

Tras finalizar la lectura mencionada, en congruencia con lo planificado, se siguió con la traducción del algoritmo de detección de Python a C++. En esta última etapa del diseño, la meta se centraba no solo en la obtención del módulo VHDL funcional, si no también en la configuración correcta de la interfaz con las señales propias de los bloques sintetizados en HLS.

Una vez completado, se inició un proceso iterativo para optimizar el bloque en respuesta al requisito de trabajar en tiempo real. En cada iteración, se implementaba un cambio, se testeaba el bloque con alguna imagen conocida y se verificaba de forma manual que los resultados sean correctos. En la figura **3.12**, se muestran distintos gráficos que representan la optimización de los tiempo de procesamiento del bloque a medida que se iba iterando.

Además, fue necesario tener en cuenta el tamaño del bloque sintetizado, no solo para que no supere la capacidad de la ZedBoard, si no para que sea posible implementar redundancias en un futuro. A continuación, se muestran los reportes de la herramienta de HLS antes y después de las optimizaciones finales:

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	717	818
FIFO	-	-	-	-
Instance	-	-	8256	27701
Memory	3	-	0	0
Multiplexer	-	-	-	11982
Register	-	-	15823	-
Total	3	0	24796	40501
Available	280	220	106400	53200
Utilization (%)	1	0	23	76

(a) Reporte antes de las optimizaciones

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	717	800
FIFO	-	-	-	-
Instance	-	-	8256	14231
Memory	3	-	0	0
Multiplexer	-	-	-	12012
Register	-	-	15822	-
Total	3	0	24795	27043
Available	280	220	106400	53200
Utilization (%)	1	0	23	50

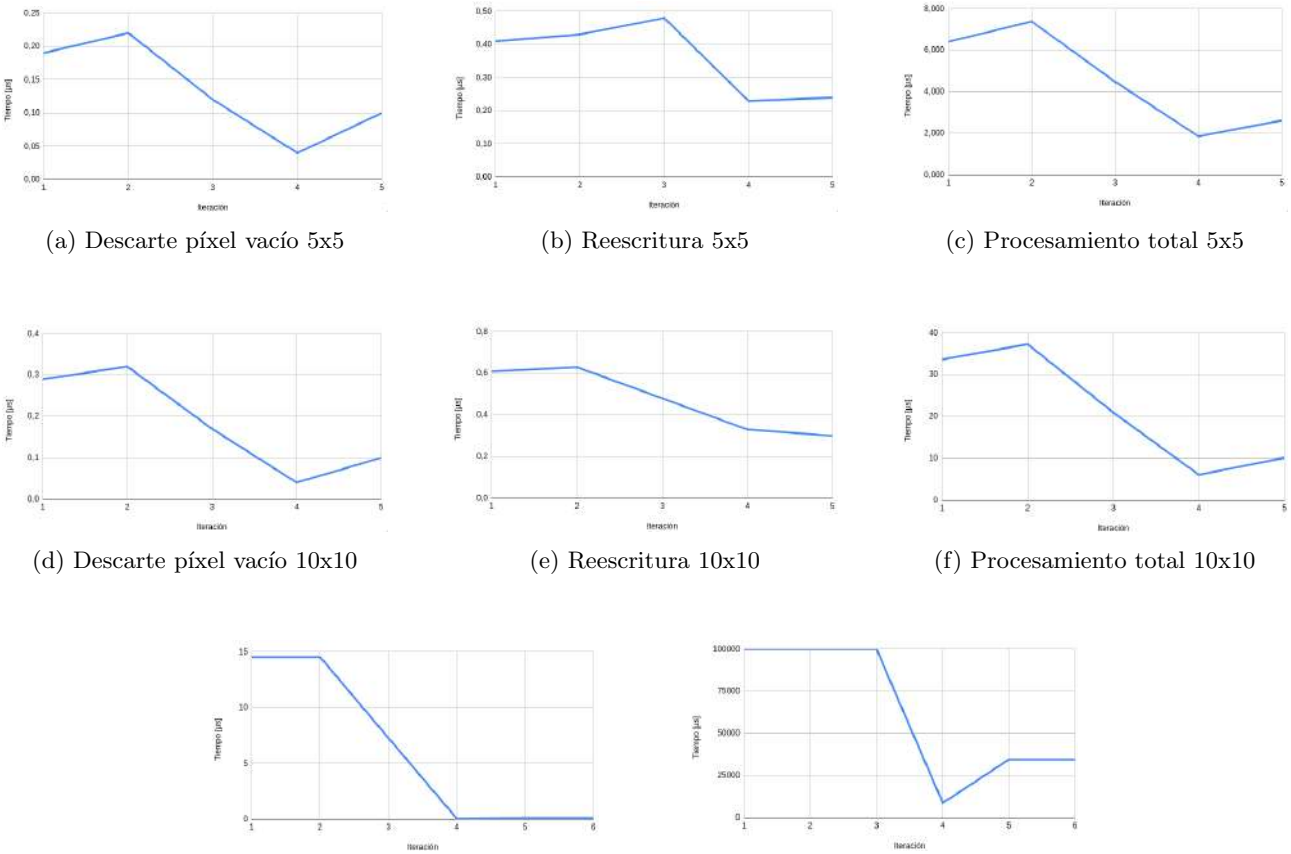
(b) Reporte después de las optimizaciones

Figura 3.13: Consumo de área del detector

Inicialmente se comenzó con 24796 FlipFlops y 40501 LUTs y se logró reducir hasta 24792 FlipFlops y 27043 LUTs sin modificar el trabajo en tiempo real. Lo que se traduce en una reducción del 33 % en el uso de LUTs.

Por otro lado, durante esta etapa hubo una reunión con Gabriel Sanca, responsable de LabOSat, donde se definió que el consumo de potencia se restringe a 1 Watt. En este punto, se empezó a hacer un análisis de potencia para cumplir con el requisito. Durante un primer reporte, la herramienta informó un consumo de 20.4W. Sin embargo, rápidamente se encontró que el consumo se debía a un clock mal conectado. Tras corregir este defecto, el reporte en esta etapa de desarrollo fue de 0.291W.

Con este resultado, quedó concluido el diseño del bloque de detección.



(a) Descarte píxel vacío 5x5 (b) Reescritura 5x5 (c) Procesamiento total 5x5

(d) Descarte píxel vacío 10x10 (e) Reescritura 10x10 (f) Procesamiento total 10x10

(g) Descarte píxel vacío 480x720 (h) Procesamiento total 480x720

Figura 3.12: Iteración [n] vs. Tiempo de procesamiento [μs]

3.2.2. Generación de histograma

El siguiente bloque a realizar, acorde al plan de proyecto, era el módulo limpiador de píxeles muertos. En otras palabras, de aquellos eventos cuyo tamaño es de un píxel. Sin embargo, se notó que la funcionalidad podía ser incluida en un único bloque junto con el generador de histogramas.

Previo a comenzar con el desarrollo del bloque, se validó la idea de descartar los píxeles de tamaño unitario ya que se consideró que eran resultado de píxeles muertos. En esta ocasión, se emplearon 125 de las 628 imágenes de irradiaciones provistas. Para hallar los píxeles muertos que en verdad existen se realizó una multiplicación punto a punto entre el frame n y $n-1$. Si el resultado era distinto de cero, se podía considerar al píxel como muerto, ya que es sumamente improbable que un mismo píxel sea impactado en dos frames consecutivos. Por otro lado, para obtener los píxeles muertos de la implementación propuesta, se envió el frame n al detector y se extrajeron las coordenadas de los eventos de un solo píxel. Luego, se compararon las posiciones de los píxeles obtenidos con ambos métodos. El proceso se ejemplifica en la figura **3.14**. A partir de los píxeles descartados erróneamente y la cantidad total de píxeles muertos según el análisis ideal, se calculó el error de la implementación. Esta prueba arrojó una disminución de casi 2 % en la precisión, dando como resultado casi un 96 % de acierto.

Asimismo, se realizaron múltiples histogramas de las imágenes de irradiaciones provistas, con el objetivo de encontrar el valor máximo para fijar un margen de seguridad. Las imágenes facilitadas representan una muestra pequeña de capturas en ciertas condiciones de irradiación. Por lo tanto, se quiere asegurar que el instrumento pueda funcionar en ambientes donde la radiación sea mayor. En la figura **3.15** se puede observar el histograma, donde el eje de abscisas representa la intensidad del evento y el eje de ordenadas el número de ocurrencias. El valor máximo hallado estuvo levemente por debajo de 28000, se consideró un margen de seguridad de 40 %, colocándose el valor máximo del histograma en 40000.

Con la propuesta validada, se prosiguió con la implementación en HLS. Gracias a la experiencia adquirida durante el desarrollo del bloque de detección, el tiempo requerido por ese módulo fue considerablemente menor. Además, el hecho de que la lógica haya sido simple, favoreció a la elaboración.

Debido a que se optó por considerar como píxeles muertos a los eventos cuyo tamaño sea igual a 1, las primeras $2 \times umbral$ posiciones del histograma quedan vacías. Se aprovechó este espacio para informar el estado del sensor, cumpliendo así el requisito **RF05**.

El testeo de esta imagen se realizó enviando vectores de eventos e intensidades al bloque y observando, manualmente, la salida. Según el test **ID8** del plan de pruebas.

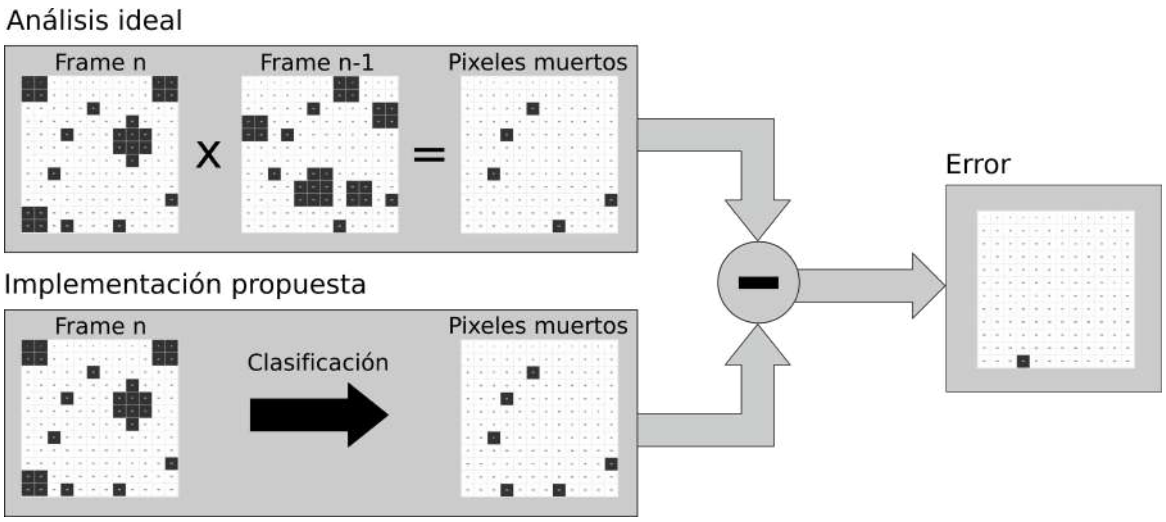


Figura 3.14: Proceso para hallar el error en la clasificación

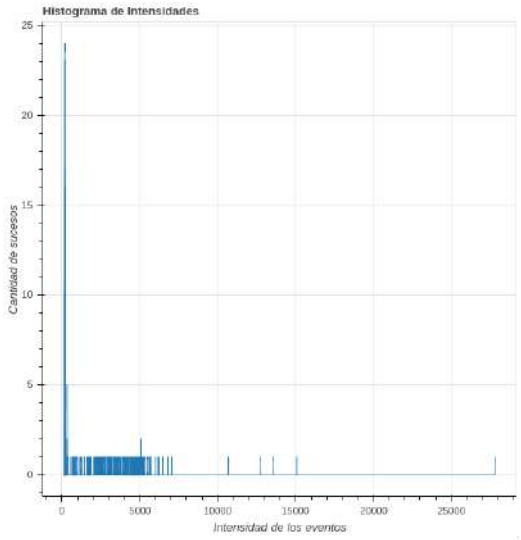


Figura 3.15: Histograma de intensidad

Una vez que los bloques fueron desarrollados, solo faltaba integrarlos y correr el test **ID10**. El esquema de testeo se realizó según se indica en la figura **3.16**.

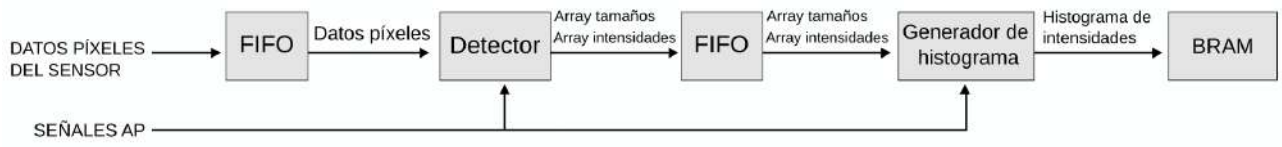


Figura 3.16: Banco de pruebas para la integración

Se realizaron las siguientes pruebas sobre este banco:

- Imágenes aleatorias de 5x5.
- Imágenes aleatorias de 10x10.
- Frame de 480x720.
- Frames de 1280x1024.

3.2.3. Interfaz

Para finalizar esta etapa, se procedió con el acabado de la interfaz. En principio, durante el interfaceado se realizó una versión precaria con el objetivo de poder verificar la captura de imágenes y la lectura/escritura de los

registros. Esto se hizo debido a que el envío de comandos utilizando la terminal se volvía tedioso e insumía una cantidad de tiempo considerable. Ahora, era necesario agregar las funciones restantes: obtener el histograma y enviar imágenes desde un archivo. Además, había que reconstruir una pestaña de captura acorde a las funciones de captura: umbral, cantidad de imágenes, cantidad de capturas y la posibilidad de que el detector esté en funcionamiento por tiempo indeterminado.

Como se mencionó a principios del capítulo, la interfaz se realizó utilizando Tkinter, sin embargo, este framework está lejos de ser el adecuado para diseñar la parte visual. Considerando lo expuesto anteriormente, se decidió utilizar PyQT5 para continuar con su desarrollo. En términos de diseño, resultó mucho más cómodo e intuitivo, lo cual facilitó notablemente la transición.

Tal y como se describe en la especificaciones funcional y técnica (Apéndices C y D), la interfaz diseñada se presenta en al figura **3.17**. Con el fin de incrementar la productividad, se implementaron threads. Gracias a esto, la interfaz no quedaba congelada durante una acción. Además, permitía realizar la reconstrucción de imágenes e histogramas mientras que, en paralelo, el instrumento realizaba mediciones y se guardaban los nuevos datos en la computadora.

(a) Pestaña 1: Configuración UART

(b) Pestaña 2: Captura de imágenes

(c) Pestaña 3: Registros

(d) Pestaña 4: Carga de imágenes

(e) Pestaña 5: Histogramas

Figura 3.17: Interfaz

3.3. Tercera etapa: Lógica de Control

Con los bloques anteriores ya funcionales, se comenzó a implementar los bloques de control principales que posibilitan cumplir con el **RF04**, los cuales son:

- Automatic Mode Controller (AMC).
- Debug Mode Controller (DMC).

Mediante estos bloques se coordinan los módulos de detección e histograma para que el instrumento funcione de manera correcta, además de entregar datos para poder validar su funcionamiento. Sin embargo, estos controladores son una cáscara que engloba diferentes sub-bloques según lo definido en la especificación técnica (Apéndice **D**).

Durante esta etapa del desarrollo se consideró tener un modo de funcionamiento que permita insertar una imagen directamente en la entrada de datos. En otras palabras, se deshabilita el sensor y, en la FIFO de entrada, se colocan los datos de la imagen. De esta manera se pueden envíar una captura conocida, con un histograma ya hecho y, tras procesar los píxeles con el instrumento, comparar los histogramas. Dicho de otra forma, posibilita validar el funcionamiento del instrumento sin la necesidad de exponer el sensor a radiación.

3.3.1. Automatic Mode Controller

Para el desarrollo del Automatic Mode Controller se examinó la lógica desarrollada previamente, en la primer etapa. Se encontró con que había secciones del código que se podían reutilizar y otras que no. Siguiendo con la especificación técnica y utilizando el paper de Xilinx sobre como implementar máquinas de estado [13], se completó el desarrollo íntegramente en VHDL.

3.3.2. Debug Mode Controller

El desarrollo de este bloque no presentó complicaciones. Si bien hay un sub-bloque que implementa una interfaz AXI, gracias a la experiencia obtenida en la primer etapa se pudo realizar el mismo de manera rápida. El resto del módulo fue, al igual que el AMC, desarrollado en VHDL.

3.4. Driver

A medida que se iban creando los bloques, se debía ampliar las funcionalidades del driver básico realizado en la primer etapa. El driver es el nexo que permite que se ejecute un comando en la interfaz y que la parte de lógica programable cumpla con la función esperada. Hasta este punto sólamente había dos comandos implementados en el driver: capturar imagen y leer/escribir registros.

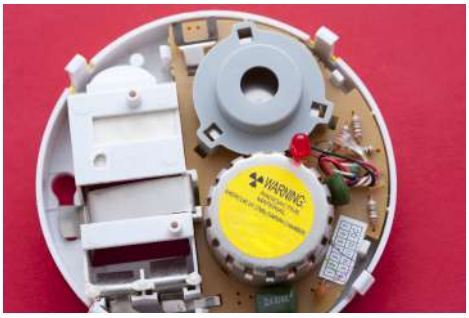
Por lo tanto, se siguió con el desarrollo de los modos de funcionamiento que faltaban:

- Iniciar captura.
- Obtener histograma.
- Insertar imagen y obtener su histograma.

Entonces, se fueron agregando los modos faltantes haciendo un uso extenso de los AXI GPIOs que permiten comunicar al driver con la lógica programable (PL).

3.5. Integración y testeos finales

Una vez que el desarrollo del instrumento estuvo completado, se realizó un testeo integral con una fuente de radiación real. Se trató de una fuente de Americio 241 que emite partículas alfa. Para obtener la fuente, y ante la imposibilidad de viajar a Bariloche para hacer una irradiación con el reactor experimental RA6, se decidió desarmar un sensor de humo y extraer la piedra de Americio que lleva en su interior. A continuación, se muestran imágenes del sensor de humo y la fuente radioactiva:



(a) Detector de humo desarmado

(b) Fuente de Americio 241

Figura 3.18: Fuente de radiación

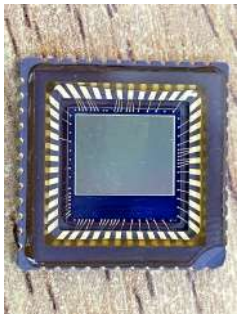
Además de una fuente radioactiva, se necesita disponer de un contenedor oscuro para minimizar los efectos de la luz natural y que los eventos se correspondan únicamente con los generados por la fuente. Por lo tanto, se tomó una caja de motherboard y se la selló con cinta de aluminio, de la siguiente forma:

(a) Cerrado

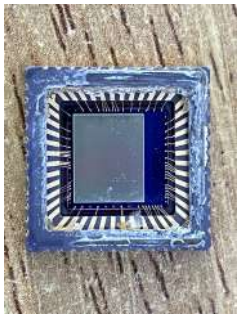
(b) Abierto

Figura 3.19: Contenedor oscuro

Las partículas emitidas por la fuente de Americio tienen un rango muy corto de acción, una simple hoja de papel puede detenerlas completamente. Por lo tanto, al sensor CMOS es necesario removerle el vidrio que lleva por encima del área activa de los píxeles. Dichos sensores con vidrios removidos fueron provistos por el Laboratorio de Bajas Temperaturas del Instituto Balseiro. A continuación, se muestra una imagen del sensor con vidrio y sin vidrio:



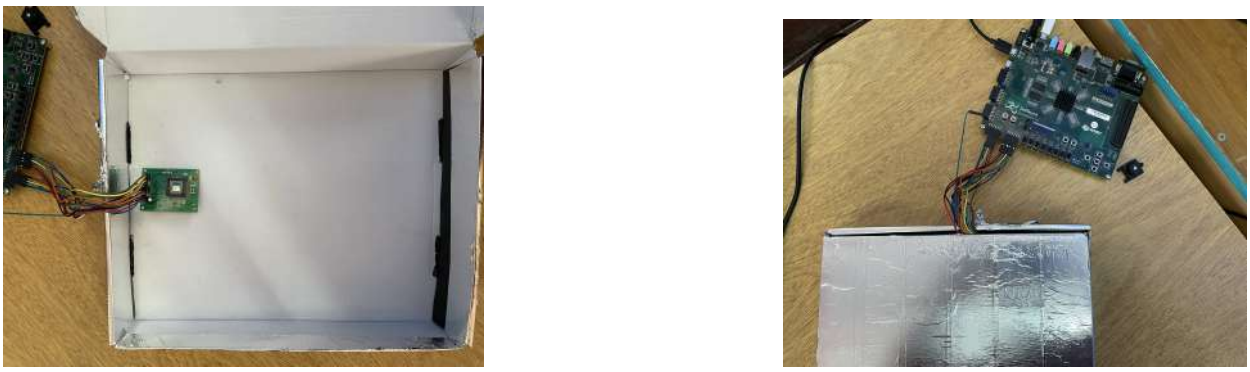
(a) Con vidrio



(b) Sin vidrio

Figura 3.20: Sensores

De manera que la plataforma de testeo quedó compuesta de la siguiente manera:



(a) Caja abierta (b) Caja cerrada

Figura 3.21: Esquema de testeo

Se realizaron capturas con y sin la fuente radiactiva según lo especificado en ID20 del configurando distintas ganancias del sensor. En la figura **3.22**, se muestran tres histogramas con la fuente de Americio 231 sobre el sensor, con ganancias x1, x8 y x15. Cabe aclarar que se decidió no graficar los píxeles muertos en los histogramas, debido a que los gráficos quedarían fuera de escala.

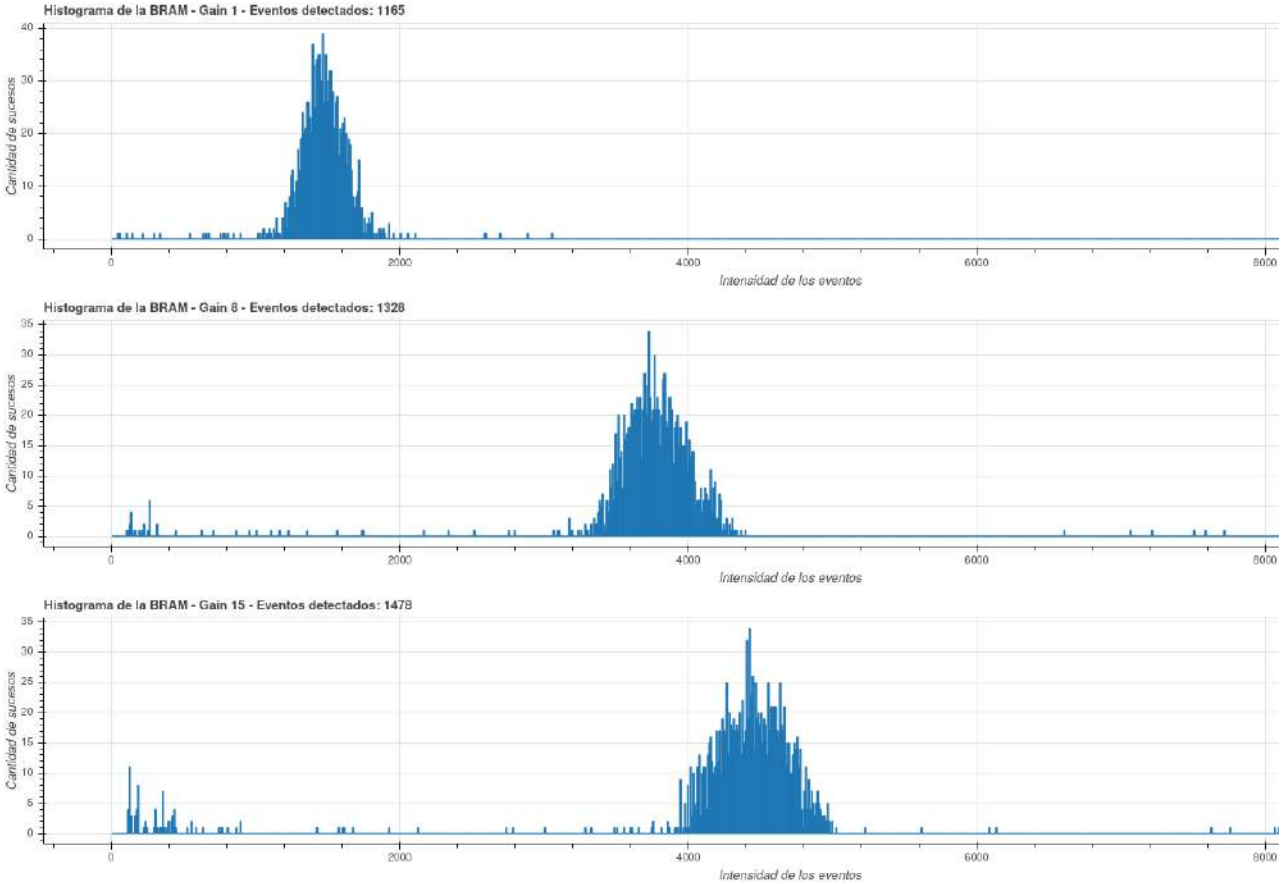


Figura 3.22: Comparación de histogramas para ganancias x1, x8, y x15

También, se obtuvo la cantidad de píxeles muertos antes, durante y luego de la irradiación para distintas ganancias. En la siguiente tabla se muestran los resultados obtenidos (cuadro **3.2)**:

Ganancia	Preirradiación	Irradiación	Postirradiación
x1	713	11596	10812
x8	17159	52632	65790
x15	7	83791	109905

Cuadro 3.2: Tabla de píxeles muertos

Luego, se procedió a realizar el test ID21, acorde al plan de pruebas. Al igual que en la prueba anterior, se utilizó el contenedor oscuro y la piedra de ^{241}Am, pero con el sensor tapado por una hoja de papel. Esto se realizó ya que una hoja de papel deja pasar partículas gamma, pero frena las alfas.

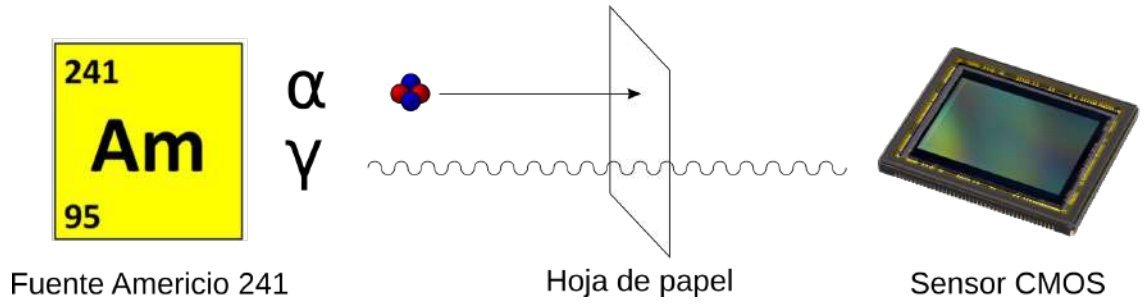


Figura 3.23: Esquema de testeo para partículas gamma

Esta prueba debía durar 24 horas, ya que la cantidad de partículas gamma que genera la fuente de radiación es muy baja. Sin embargo, los archivos del SDK de Xilinx se corrompieron a las 3 horas, impidiendo que se lleve a cabo la prueba.

Capítulo 4

Conclusiones

4.1. Resultados del instrumento

Tras irradiar al instrumento con la fuente de ^{241}Am, se esperaría ver una distribución normal cuyo valor medio y varianza sea dependiente de la ganancia con la que esté configurado el sensor. A medida que se incrementa la ganancia, se debería observar que el centro se desplaza hacia la derecha y se ensancha. Es decir, la media se traslada hacia valores más altos de LET.

En la figura **3.22**, donde se muestran los histogramas obtenidos durante la irradiación con distintas ganancias, se puede apreciar como efectivamente sucede lo esperado. El valor medio de la gaussiana se desplaza hacia valores más altos de LET a medida que la ganancia es incrementada, así como la varianza de la misma. Se puede concluir que es consistente con lo esperado.

A medida que el sensor es irradiado, los píxeles que conforman el área activa se van rompiendo. Por lo tanto, es esperable que luego de una irradiación la cantidad de píxeles muertos aumente. Esto es justamente lo que se puede observar en la tabla 3.2.

A continuación, se muestra el reporte final de timing, uso de área y consumo de potencia del instrumento desarrollado:

Timing		Recursos		Potencia	
WNS	0.091 ns	LUT	47.58 %	PS	1.533W
WHS	0.021 ns	LUTRAM	1.44 %	PL	0.452W
WPWS	3 ns	FF	23.72 %	Total	1.985W
		BRAM	25.71 %		
		IO	9 %		
		MMCM	25 %		

Cuadro 4.1: Reporte final de la implementación

Si bien la potencia total consumida fue de 1.985 W, se puede notar que el PS consumió 1.533 W. En la forma de operación deseada, es decir, en modo autónomo, el PS no estaría. Por lo tanto, el consumo se reduciría a aproximadamente 0.452 W. Sumando el consumo del sensor de 0.363 W, el total quedó en 0.815 W. De manera que se cumplió la restricción de potencia mencionada en el Apéndice **B**.

Por otro lado, el consumo de área fue elevado. El instrumento hizo uso de aproximadamente la mitad de las LUTs disponibles. Por lo tanto, se debió descartar la idea inicial de replicar la lógica del instrumento para poder implementar redundancias.

4.2. Experiencias de la gestión del proyecto

El proyecto insumió 202 horas. El valor es aproximado, ya que durante la primera etapa del proyecto no se llevó a cabo un registro de horas de las tareas realizadas. En la segunda etapa, que coincidió con la cursada del seminario de proyecto final de ingeniería electrónica, se realizó un registro detallado de tareas llevadas a cabo. La siguiente tabla contiene las horas insumidas por cada tarea:

Tarea	Cantidad de horas
Documentación	48
Bloque detector	24
Bloque histograma	7.5
Integración detector e histograma	11
Análisis de imágenes de Radiación	10.5
Desarrollo modos Debug y Automático	65
Diseño interfaz	15
Pruebas	16.5
Puntos de Control	4.5
Total registrado	202

Cuadro 4.2: Horas registradas

A continuación, en la figura **4.1**, se muestra el diagrama de Gantt final del proyecto. Para comparar, el diagrama inicial se encuentra en la figura **2.4**.

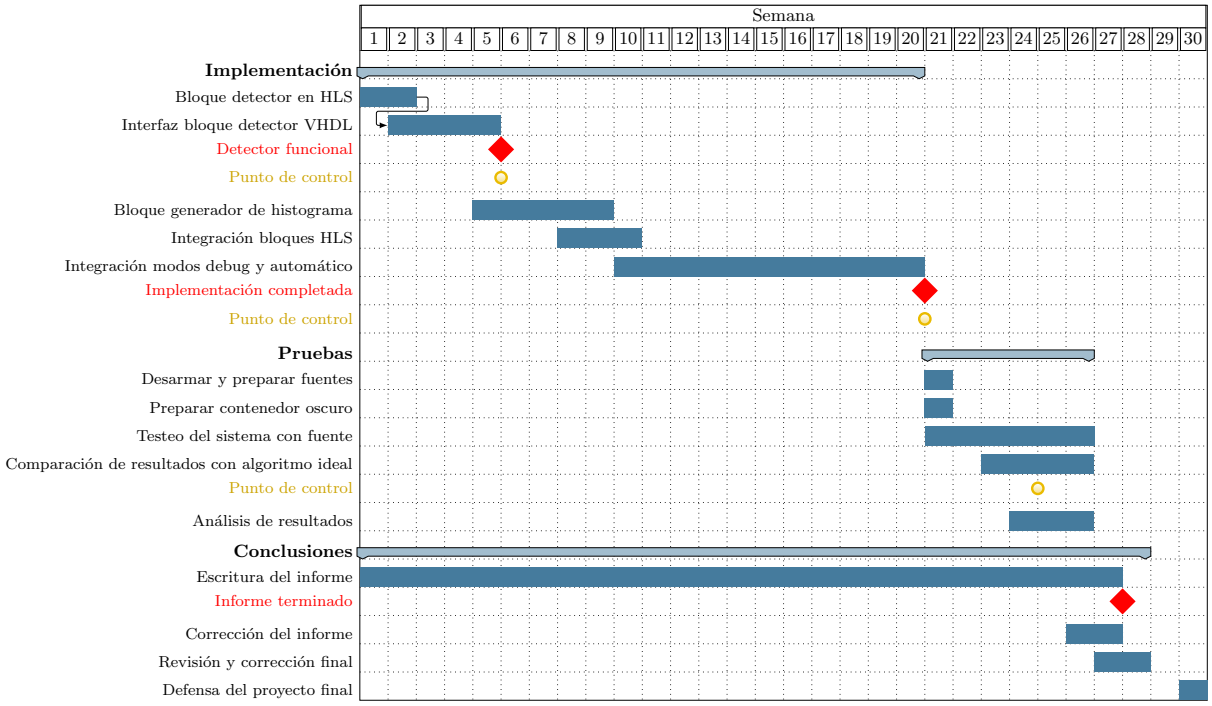


Figura 4.1: Gantt Final

Como primer contratiempo se puede mencionar un problema que surgió respecto a la colocación del sensor en su zócalo. En las primeras instancias del desarrollo del instrumento, luego de la lectura de [8], [9] y [14], se procedió con el soldado de la placa, la colocación del sensor y la verificación de sus señales. Esta comprobación se hizo observando las señales que entrega y verificando que se correspondan con lo esperado. Se utilizó Vivado en primera instancia y, posteriormente, un osciloscopio. Con las puntas de prueba en los pines correspondientes del zócalo se midieron *frame valid*, *line valid*, *PIXCLOCK* y V_{cc}. Durante esta prueba se encontró que las señales no eran las esperadas. Tras un período de 3-4 semanas, se detectó que el problema estaba en la colocación del sensor, y no en el dispositivo en sí. El mismo debía ser apretado en el zócalo hasta hacer un clic audible.

En cuanto a la divergencia mas notoria, se puede mencionar que la fecha estimativa de finalización se vio corrida por aproximadamente tres meses. Esto se debe a que cuando se desarrolló el Gantt en el plan de proyecto no se tuvo en cuenta las fechas de finales y el tiempo de estudio. También durante agosto Mariano consiguió trabajo, por lo que el desarrollo se pudo llevar a cabo al finalizar el horario laboral, lo que disminuyó considerablemente el tiempo disponible para el desarrollo del proyecto final. Además, aquellas pruebas que requirieran la utilización de la FPGA se debieron realizar los días sábados, ya que la ZedBoard se encontraba en el Laboratorio de Sistemas Caóticos y el sábado era el único día en que ambos alumnos estaban disponibles.

Por otro lado, se subestimaron los tiempos de desarrollo. Principalmente de los bloques en HLS. Esto se debe a que para que la herramienta infiriera correctamente lo pretendido había que escribir código con ciertas directivas específicas. Al desconocer totalmente sobre el tema, se dedicó una semanas a leer el libro *HLS Bluebook* ([12]), con el objetivo de tener una noción inicial acerca del funcionamiento de este lenguaje y las herramientas asociadas. Sin embargo, el libro no mencionaba nada sobre las interfaces que implementan las

estructuras sintetizadas en HLS. Por lo tanto, se crearon mediante prueba y error bloques simples que facilitaran el aprendizaje de su funcionamiento. Una vez que se consideró que el conocimiento sobre HLS era suficiente, se prosiguió con las siguientes tareas de desarrollo.

Otro punto importante de divergencia fue durante las pruebas con la fuente de americio. Para que el sensor pueda reaccionar frente a partículas alfa, como se mencionó previamente, es importante removerle el vidrio al sensor CMOS. Durante las pruebas se debían hacer mediciones con tres ganancias, x1, x8 y x15. Sin embargo, luego de hacer la medición con la ganancia en x15, se observó que el sensor dejó de funcionar. Por lo tanto, se cambió el sensor defectuoso por otro. Cuando se leyó el registro 0x00, que debería entregar el valor 0x8431, se obtenía un valor que variaba de lectura en lectura. Se descartó el sensor defectuoso y se utilizó otro. Nuevamente, sucedió lo mismo que en el primer caso al ser irradiado con ganancia x15. De los tres sensores sin vidrio que se disponía no quedaba ninguno funcionando. Por lo tanto, se le solicitó a Martín Pérez y a José Lipovetzky el envío de sensores nuevos. El paquete demoró dos semanas en llegar. Mientras tanto, se dedujo que el problema surgía cuando se abría el contenedor oscuro luego de hacer la medición. La FPGA quedaba prendida, en consecuencia, el sensor también, y expuesto a la luz sin vidrio ni lente. Entonces, cuando la ganancia se configuraba en x15 y se abría el contenedor, la luz que incidía sobre el mismo era demasiada y el sensor se rompía.

4.3. Conocimientos aplicados y adquiridos

Durante el desarrollo del instrumento, se encontró con que, si bien se contaba con conocimientos básicos sobre VHDL, Python y C, estos no eran suficientes para poder llevar a cabo el proyecto. Por lo tanto, se tuvo que profundizar en el conocimiento de los distintos lenguajes de programación utilizados a medida que se iba avanzando en el proyecto. El hecho de desconocer y tener que aprender sobre varios conceptos desconocidos se vio reflejado en un atraso significativo durante la ejecución de las tareas planteadas en el diagrama de Gantt. Un claro ejemplo de esto fue el aprendizaje acerca del lenguaje HLS, sin el cual el desarrollo de los bloques de procesamiento no hubiera sido posible. Como se mencionó anteriormente, fue de vital importancia la lectura que se llevó a cabo de [12]. Este desconocimiento sobre algunos conceptos derivó en no tener muy clara la forma de abordar inicialmente los problemas a resolver, ya que, a primera vista, existen un sinnúmero de posibles implementaciones, todas igualmente válidas.

En cuanto a los conocimientos aplicados, fundamentalmente se utilizaron conceptos sobre el diseño de máquinas de estado. Se utilizó tanto para la lógica de control como para la implementación del protocolo de comunicación con el sensor. También se contaba con un entendimiento elemental de VHDL, sabiendo implementar operaciones aritméticas e instanciar módulos. De manera similar, gracias a la práctica adquirida durante la cursada de las materias de la carrera, el desarrollo inicial en MATLAB del algoritmo real fue sencillo y rápido. Otros conocimientos que resultaron útiles fueron los detectores de flanco, para detectar cambios en las señales del sensor, y la relación entre la señal a muestrear y la frecuencia de muestreo, para poder observar el comportamiento de las señales en las distintas pruebas.

Se considera que el progreso logrado en VHDL fue muy bueno. Con el objetivo de profundizar sobre la base previa, se leyó [15] para VHDL y y [13] para la implementación de máquinas de estado. El conocimiento se puso en práctica de manera inmediata. Primero, se diseñó un módulo de captura y acondicionamiento de señales para enviar una imagen hacia la computadora. Posteriormente, se interpretó un módulo I2C ya escrito y se modificó según fue necesario para que funcione con el protocolo propietario del sensor. En relación al acondicionamiento de señales, se realizó un contador para comprobar la cantidad de píxeles transmitidos, módulo que se reutilizó para la cantidad de imágenes a capturar. El detector de flanco fue de gran importancia para obtener los datos correctamente. Esto se debe a que los datos de los píxeles son válidos en el flanco descendente de la señal *PIXCLOCK*. Por otro lado, se pudo aprender y utilizar el concepto de parametrización de módulos. Esta técnica hace que los mismos puedan ser reutilizables. De esta manera, se evita la reescritura del módulo y se acortan los tiempos de desarrollo.

El driver creado para el correcto funcionamiento del PS y la comunicación con la interfaz supuso un gran desafío. Sin embargo, gracias a la gran cantidad de tutoriales disponibles y la abundante bibliografía provista por Xilinx, se pudo desarrollar sin mayores complicaciones. Durante el transcurso de su diseño, se profundizó sobre conceptos del lenguaje de programación C. En particular, se observó cómo se implementa para intercomunicar el PS y el PL, a través del protocolo AXI.

El uso de Python fue de vital importancia para la creación del instrumento. El hecho de ser un lenguaje de alto nivel facilita tanto la implementación como la legibilidad del código desarrollado. La buena documentación y facilidad de uso de la amplia variedad de librerías que presenta este lenguaje fue de gran ayuda para las distintas etapas del proyecto. Un claro ejemplo de esto fue el desarrollo de la interfaz. En una primer instancia se utilizaron las librerías TKinter y PAGE. Para la versión final se utilizó PyQT5 y Qt5 Designer, que presentan más y mejores funciones. Además, se hizo un uso extensivo de la librería Threading, para crear múltiples hilos de ejecución, y Bokeh, para la creación y visualización de histogramas interactivos. Por otra parte, durante las etapas de comparación de los algoritmos, se pudo crear rápidamente un entorno de pruebas que realizaba todas

las funciones necesarias: creaba las imágenes de manera aleatoria, ejecutaba ambos algoritmos y comparaba el resultado obtenido. En este entorno de pruebas se hizo un uso extensivo de la librería os, que permite recorrer las distintas carpetas y almacenar los resultados.

Por último, se puede concluir que las capacidades y conocimientos adquiridos durante el transcurso de la carrera fueron de vital importancia. A medida que se avanzaba en el proyecto surgieron un sinnúmero de dificultades y obstáculos. Sin embargo, las aptitudes obtenidas en cuanto a la forma de resolver y enfrentar problemas ayudaron a sortearlos. Como ejemplo de las facultades adquiridas, se puede destacar la destreza en el diseño y abstracción. Esto permitió que se realice un buen análisis previo que se tradujo en no tener que rediseñar partes ya realizadas. Por otro lado, los conocimientos técnico que provee la carrera, en muchos casos, sirvieron para sentar bases firmes sobre las cuales continuar con el aprendizaje.

Capítulo 5

Bibliografía

[1] G. Knoll, *Radiation Detection and Measurement*. Wiley, 2010, ISBN: 9780470131480. dirección: https://books.google.com.ar/books?id=4vTJ7UDel5IC.

[2] F. Bezerra, E. Lorfevre, R. Ecoffet, D. Falguere y P. Bourdoux, «CARMEN/MEX test board for the study of radiation effects on electronic components aboard JASON-2 and SAC-D satellites,» en *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, 2007, págs. 1-6. DOI: 10.1109/RADECS.2007.5205541.

[3] M. Pérez, J. Lipovetzky, M. Sofo Haro, I. Sidelnik, J. J. Blostein, F. Alcalde Bessia y M. G. Berisso, «Particle detection and classification using commercial off the shelf CMOS image sensors,» *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 827, págs. 171-180, 2016, ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2016.04.072. dirección: https://www.sciencedirect.com/science/article/pii/S0168900216302844.

[4] M. Pérez, M. S. Haro, I. Sidelnik, L. Tozzi, D. R. Brito, C. Mora, J. J. Blostein, M. G. Berisso y J. Lipovetzky, «Commercial CMOS pixel array for beta and gamma radiation particle counting,» en *2015 Argentine School of Micro-Nanoelectronics, Technology and Applications (EAMTA)*, 2015, págs. 11-16. DOI: 10.1109/EAMTA.2015.7237371.

[5] Y. Degerli, F. Guilloux y F. Orsini, «A novel CMOS sensor with in-pixel auto-zeroed discrimination for charged particle tracking,» *Journal of Instrumentation*, vol. 9, n.º 05, págs. C05018-C05018, mayo de 2014. DOI: 10.1088/1748-0221/9/05/c05018. dirección: https://doi.org/10.1088/1748-0221/9/05/c05018.

[6] R. Velazco, J. A. Clemente, G. Hubert, W. Mansour, C. Palomar, F. J. Franco, M. Baylac, S. Rey, O. Rosetto y F. Villa, «Evidence of the Robustness of a COTS Soft-Error Free SRAM to Neutron Radiation,» *IEEE Transactions on Nuclear Science*, vol. 61, n.º 6, págs. 3103-3108, 2014. DOI: 10.1109/TNS.2014.2363899.

[7] A. Karapetian, R. Some y J. Beahan, «Radiation fault modeling and fault rate estimation for a COTS based space-borne supercomputer,» en *Proceedings, IEEE Aerospace Conference*, vol. 5, 2002, págs. 5-5. DOI: 10.1109/AERO.2002.1035378.

[8] M. Pérez, *Detección de radiación ionizante utilizando sensores de imagen CMOS comerciales*. sep. de 2018.

[9] S. Azimi, B. Du, L. Sterpone, D. Codinachs, R. Grimoldi y L. Cattaneo, «A new CAD tool for Single Event Transient Analysis and mitigation on Flash-based FPGA,» vol. 67, págs. 73-81, 2019.

[10] Aptina, «1/2-Inch Megapixel CMOS Digital Image Sensor,» ver. K, dirección: https://media.digikey.com/pdf/Data%5C%20Sheets/APTINA%5C%20PDFs/MT9M001C12STM.pdf.

[11] Altera, «Metastability in Altera Devices,» ver. 4, mayo de 1999. dirección: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an042.pdf.

[12] M. Fingeroff y T. Bollaert, «High-Level Synthesis Blue Book,» 2010. dirección: https://www.cse.usf.edu/~haozheng/teach/cda4253/doc/hls/hls_bluebook_uv.pdf.

[13] Xilinx, «Finite State Machines,» 2015. dirección: https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/VHDL/docs-pdf/lab10.pdf.

[14] S. Azimi, B. Du y L. Sterpone, «A Single Event Effects Analysis and Mitigation Algorithm for Flash-based FPGAs,» *Transactions on Computer*,

[15] M. Sánchez-Élez, *Introducción a la programación en VHDL*. jul. de 2014. dirección: https://eprints.ucm.es/id/eprint/26200/1/intro_VHDL.pdf.

[16] T. support team Micron, *MT9M001 user manual*, 2004. dirección: https://pdf1.alldatasheet.com/datasheet-pdf/view/115168/MICRON/MT9M001.html.

[17] T. support team AVNET, *Zedboard user manual*, 2004. dirección: https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/zedboard/.

[18] L. team, *LabOSat: a customized Lab On a Satellite*, 2017. dirección: http://www.unsam.edu.ar/escuelas/ciencia/labosat/.

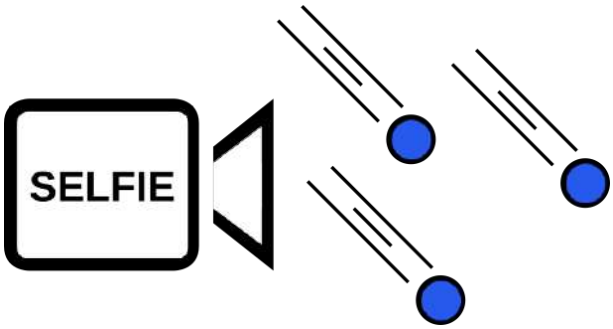
Apéndices

Apéndice A

Plan de proyecto

Carrera: Ingeniería Electrónica

Scanner Electrónico de Flujo de Iones Espaciales (SELFIE)



Detector de Radiación Ionizante para aplicaciones satelitales

Plan de proyecto

Introducción:

Título propuesto:	Detector de radiación ionizante para aplicaciones satelitales
Estudiantes:	García, Mathías Sebastián. Legajo: 9654 Rolón, Mariano. Legajo: 11564
Directores:	Dr. Ing. Maximiliano Antonelli Dr. Ing. Claudio González
Lugar de desarrollo:	Laboratorio de Sistemas Caóticos (LSC) Instituto de Investigaciones Científicas y Tecnológicas en Electrónica (ICyTE) Conicet - UNMdP

El siguiente documento correspondiente al Plan de Proyecto se compone de las siguientes partes:

- Un listado de tareas pre-planificación, donde se detallan las distintas actividades que fueron realizadas antes de hacer una planificación formal.
- Un cronograma propuesto, que se divide en cinco etapas: introducción, diseño, implementación, pruebas y conclusiones. Dentro de cada una de ellas se especifican las tareas a realizar y su duración. Se eligió la división del trabajo en semanas para tener una mayor flexibilidad para extender o acortar las tareas.
- Estimación de riesgos, junto con un plan de respuesta para resolver rápidamente el problema que pueda aparecer.

Tareas realizadas pre-planificación:

El listado de tareas realizadas pre-planificación es una estimación realizada a partir de un registro propio de avances y fechas de los distintos archivos generados, ya que el proyecto comenzó en diciembre del 2019 sin una planificación estricta de tiempos y tareas.

A partir de los registros mencionados anteriormente se puede armar la siguiente reconstrucción de tareas y duraciones:

- Investigación radiación, repaso lenguaje VHDL, soldado de PCB para montar el sensor CMOS y Comunicación entre FPGA y PC a través de UART: duración todo el mes de diciembre de 2019.
- Diseño e implementación del controlador RAM: inicio 3 de enero de 2020, duración 20 dias.
- Integración del controlador RAM y UART: inicio 23 de enero de 2020, duración 8 días.
- Programación del PS para transferencia DMA: inicio 31 de enero de 2020, duración 15 días.
- Primeras lecturas del sensor: 14 de febrero de 2020.
- Diseño e implementación controlador I2C: inicio 14 de febrero de 2020, duración 5 días.
- Diseño e implementación de interfaz de captura en Python: inicio 19 de febrero de 2020, duración 9 días.
- Diseño de algoritmo de detección: febrero y marzo de 2021.

Desde febrero del 2020 hasta febrero del 2021 no se realizaron tareas debido al cursado de materias y la pandemia COVID-19.

Cronograma propuesto:

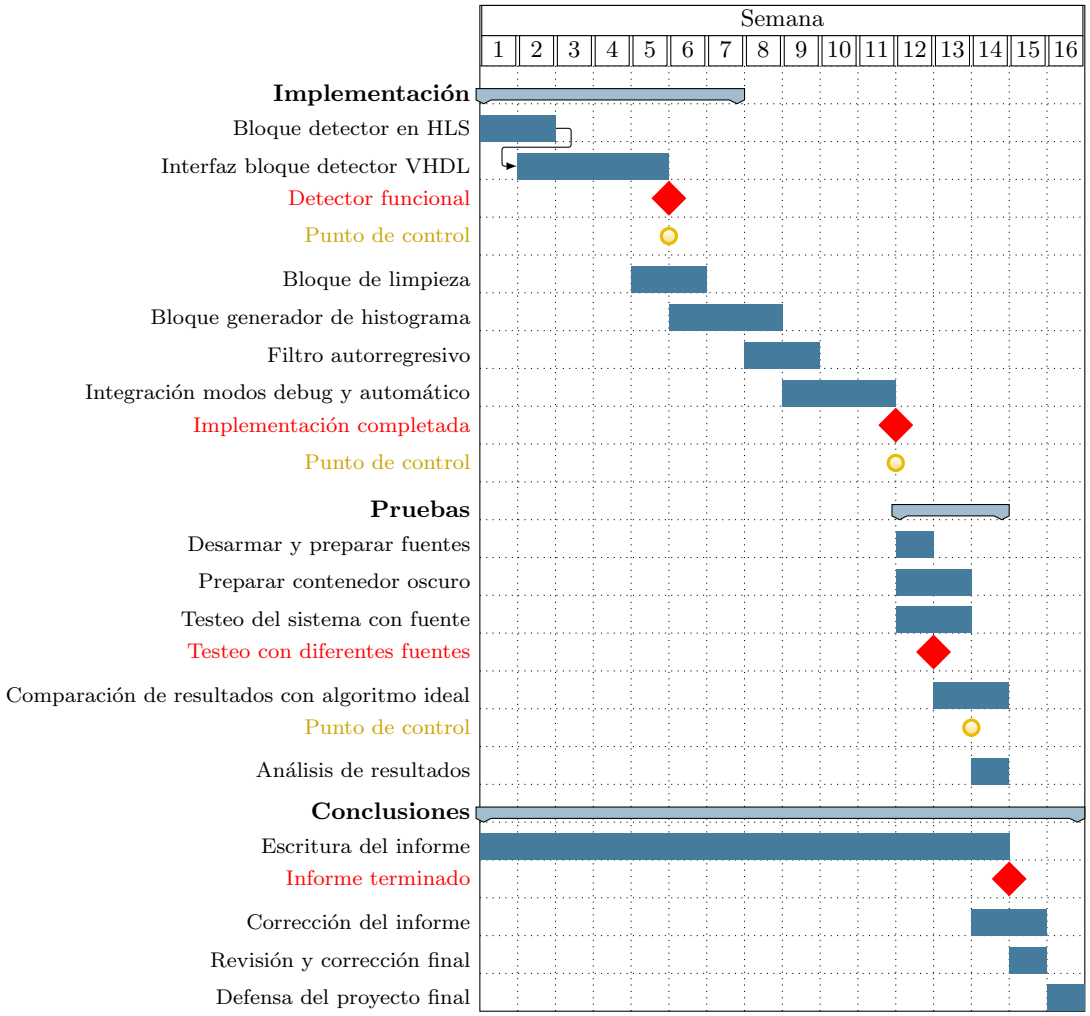
Una vez que comenzó la cursada del Seminario Taller para el Diseño de Soluciones Tecnológicas, se realizó la planificación de tareas y estimación de tiempos de manera formal mediante un diagrama de Gantt.

En el diagrama de Gantt, se especifican en amarillo los puntos de control y en rojo aparecen los hitos del proyecto. En caso de tener que hacer re-trabajo luego de un punto de control, se decide cuanto tiempo va a llevar y se reprograman las tareas, intentando respetar, siempre que se pueda, la división por semanas original.

En cada punto de control se concreta una reunión con los directores, la directora del Laboratorio de Sistemas Caóticos y especialistas del Instituto Balseiro. Durante la misma, se valida lo realizado desde el último punto hasta el momento y se define cómo se procederá hasta el próximo.

En cuanto a la división del trabajo, ambos autores del proyecto final cumplimos las mismas tareas al mismo tiempo, exceptuando las tareas de documentación. Consideramos que, para nuestro caso, es la mejor manera de realizar el desarrollo, debido a que tenemos que utilizar lenguajes de programación, tales como VHDL, C, y Python que, salvo una breve introducción a VHDL, no figuran dentro de ninguna materia del plan de estudios de la carrera de Ingeniería Electrónica. La decisión de realizar las tareas a la par se tomó con el objetivo de obtener un entendimiento similar en todas las cuestiones correspondientes al diseño e implementación del detector. Gracias a esto, confiamos que será posible evitar posibles problemas de interfaz entre bloques y lograr un desarrollo con menos errores de los distintos componentes.

La fecha de inicio del diagrama de Gantt es el 28 de abril de 2021 y la fecha tentativa de finalización es el 18 de agosto de 2021.



Riesgos identificados:

El cronograma se realizó considerando un tiempo extra en ciertas tareas con el fin de cubrir la aparición de riesgos y, de esta manera, minimizar el efecto negativo en el calendario.

En la tabla presentada, la severidad da un indicio de la cantidad de retraso que puede provocar la aparición del riesgo. Por otro lado, aquellos riesgos que tengan mayor probabilidad de aparición y cuya gravedad tenga un mayor impacto tendrán una prioridad mas alta.

El riesgo correspondiente a un incremento en las restricciones por COVID-19 es el único capaz de producir un bloqueo total del proyecto a la hora del testeo o a la hora de comprobar el correcto funcionamiento de las máquinas de estados de los controladores que conforman al detector. Sin embargo, se le asignó una severidad *media* debido a la baja probabilidad de que se repita un período de aislamiento como el sucedido en 2020. Respecto al resto de tareas, como se pueden realizar de manera remota, no producen un bloqueo en el proyecto si no un atraso, aunque esto se ve bastante aplacado debido a la definición de tareas realizada en los puntos de control sumado al tiempo extra considerado en el cronograma.

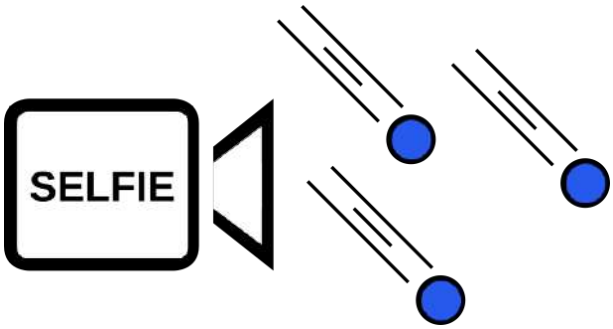
Riesgo	Probabilidad de aparición	Severidad	Plan de respuesta	Prioridad
Cambios de la plataforma de transmisión de datos	Baja	Media	Crear bloque acondicionador de señales	3
Incremento en las restricciones por COVID-19	Media	Media	Trabajar virtualmente (exceptuando la etapa de pruebas)	2
Problemas de interfaceado en bloques que involucren HLS	Alta	Alta	Consultar con expertos del tema y probar con distintos modos de interfaz	1
No disponibilidad de RAM para guardar una imagen completa	Media	Alta	Procesamiento píxel a píxel y análisis de píxeles muertos a posteriori	1
Rotura de fuente generadora de partículas durante el envío	Baja	Baja	Obtener otra fuente	3

Apéndice B

Especificación de requerimientos

Carrera: Ingeniería Electrónica

Scanner Electrónico de Flujo de Iones Espaciales (SELFIE)



Detector de Radiación Ionizante para aplicaciones satelitales

Especificación de requerimientos

B.1. Ficha del documento

Fecha	Versión	Descripción	Autor/es
15 de diciembre de 2021	1.2	Modificaciones varias	Mariano Rolon Mathias S. Garcia
08/06/2021	1.1	Aclaraciones	Mariano Rolon Mathias S. Garcia
13/05/2021	1.0	Versión inicial	Mariano Rolon Mathias S. Garcia

B.2. Introducción

B.2.1. Propósito

Este documento tiene como finalidad definir y describir los requerimientos funcionales y no funcionales del instrumento a desarrollar como proyecto final de los autores, correspondiente a la carrera de Ingeniería Electrónica. Junto con el plan de proyecto y el informe final conforman la documentación requerida para lograr la aceptación del trabajo final. Está dirigido a los autores del documento, al futuro estudiante quien continuará con el proyecto y al personal de LabOSat.

B.2.2. Ámbito del sistema

El instrumento, denominado internamente SELFIE (Scanner Electrónico de Flujo de Iones Espaciales), realiza capturas mediante un sensor CMOS comercial y procesa las señales utilizando una FPGA para caracterizar la radiación a la que está expuesto. Identifica los distintos impactos de partículas, llamados eventos, y los clasifica tanto en tamaño como en intensidad.

El objetivo principal es la reducción del costo que implica la fabricación de sensores específicos para radiación. Asimismo, el diseño del algoritmo buscará la minimización de memoria y de área en uso de FPGA, con el fin de poder -a futuro- realizar un sistema con redundancias para cubrir fallos.

La idea surge a partir de la tesis del Dr. Ing. Martín Pérez [1], donde se procesaban los eventos en una computadora y la FPGA funcionaba como interfaz entre esta y el sensor. Se busca integrar todo para que sea procesado por la parte lógica de una FPGA, prescindiendo completamente de un ordenador.

B.2.3. Personal involucrado

Nombre	Mariano Rolón
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	rolon.mariano96@gmail.com

Nombre	Mathías Sebastián García
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	mathiasgarcia.gs@gmail.com

Nombre	Maximiliano Antonelli
Rol	Director
Categoría profesional	Investigador Categoría IV
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	maxanto@fi.mdp.edu.ar

Nombre	Claudio Marcelo González
Rol	Codirector
Categoría profesional	Investigador categoría III
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	cmgonzal@fi.mdp.edu.ar

Nombre	Martín Pérez
Rol	Integrante del proyecto SELFIE
Categoría profesional	Investigador asistente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	ing.perezmartin@gmail.com

Nombre	José Lipovetzky
Rol	Investigador del proyecto SELFIE
Categoría profesional	Investigador independiente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	joselipo@gmail.com

B.2.4. Definiciones, acrónimos y abreviaturas

Acrónimo	Significado
AXI	Advanced eXtensible Interface
DMA	Acceso directo a memoria
FSM	Máquina de estados finita
HLS	High Level Synthesis
SoC	System-On-Chip
TFEC	Trabajo final de las carreras de grado de Ingeniería Electrónica e Ingeniería en Computación
VHDL	Very High Speed Integrated Circuit Hardware Description Language

B.2.5. Referencias

Título del documento	Referencia
[1] Detección de radiación ionizante utilizando sensores de imagen CMOS comerciales	Dr. Ing. Martín Pérez - Instituto Balseiro
[2] A single event effects analysis and mitigation algorithm for flash-based FPGAs	L. Sterpone, S. Azimi - IEEE
[3] A new CAD tool for Single Event Transient Analysis and mitigation on Flash-based FPGAs	L. Sterpone, S. Azimi, D. M. Codinachs, R. Grimoldi, L. Cattaneo
[4] UG585: Zynq-7000 SoC Technical Reference Manual	Xilinx
[5] Lab Workbook: Finite State Machines	Xilinx
[6] Understanding Metastability in FPGAs	Altera
[7] UG871: Vivado Design Suite Tutorial. High-Level Synthesis	Xilinx
[8] UG902: Vivado Design Suite User Guide. High-Level Synthesis	Xilinx
[9] UG998: Introduction to FPGA Design with Vivado. High-Level Synthesis	Xilinx
[10] UG1253: SDx Pragma Reference Guide	Xilinx

B.2.6. Visión General del Documento

El documento se divide en cuatro secciones, y cada una en subsecciones detallando el contenido.

La primera, se utiliza para registrar las versiones, qué fue modificado y quien realizó dicho cambio.

En la segunda sección, se detalla una introducción y proporciona una visión general de los recursos del sistema.

En la tercera, se realiza una descripción general del sistema sin incluir detalles de forma exhaustiva. Su objetivo es dar a conocer las principales funciones que éste debe realizar, los datos asociados, factores, restricciones, supuestos y dependencias que afectan al desarrollo.

La cuarta, y última, sección detalla los requisitos que debe satisfacer el sistema.

B.3. Descripción general

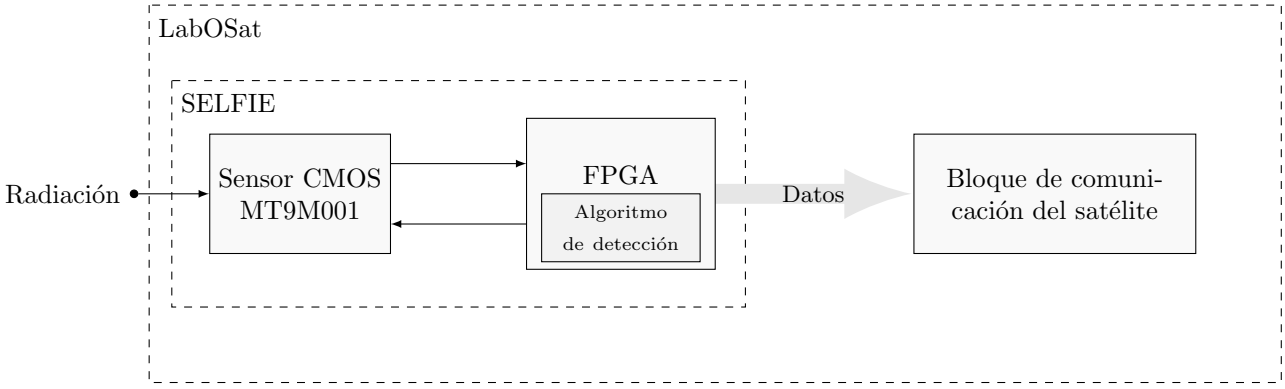
B.3.1. Perspectiva del producto

El instrumento SELFIE será un sistema diseñado para trabajar en entornos espaciales, integrado a satélites. En particular, este desarrollo será parte de la plataforma LabOSat. Por lo tanto, se deben adecuar las salidas del instrumento a aquellos requisitos que imponga la plataforma satelital. LabOSat es sinónimo de "Laboratorio en un Satélite". Es una plataforma electrónica que trabaja en la denominada Low Earth Orbit (LEO) para llevar a cabo experimentos en ambientes hostiles, tales como el espacio exterior, baja presión - condiciones de baja temperatura o bajo grandes dosis de radiación.

En cuanto a la entrada, es decir la radiación, no representará ningún problema ya que se desea detectar los niveles de radiación de todo el ambiente espacial en el que se encuentre el instrumento. Las condiciones de este ambiente fueron simuladas en [1], haciendo incidir haces de partículas ionizantes de distinto tipo, utilizando el reactor RA6.

Al tener como objetivo funcionar en ambientes espaciales y al ser el dispositivo en el que se va a implementar (FPGA) reprogramable, la tolerancia a fallos será un factor determinante a futuro. El método que se utilizará para dicha tolerancia son las redundancias, se planea replicar el mismo circuito varias veces sobre la misma placa, y luego realizar un sistema de votación para comparar resultados. Por consiguiente, se debe desarrollar un algoritmo que ocupe la menor área posible, ya que el tamaño del circuito resulta determinante para implementar la metodología debido a la restricción de área que impone la FPGA. Además se debe tener en cuenta que el algoritmo debe funcionar en tiempo real, es decir, su ejecución debe ser eficiente.

El otro objetivo es crear un instrumento de detección de radiación barato, en un mercado donde los costos son elevados. Los aspectos constructivos del instrumento, tales como la fabricación de la placa PCB, la elección de la FPGA, el soldado de componentes y las consideraciones necesarias para que la PCB funcione en ambientes espaciales serán responsabilidad del futuro estudiante que continúe con el desarrollo.



B.3.2. Funcionalidad del producto

El sistema que se detalla en el presente documento debe poder procesar la imagen del sensor CMOS, filtrar los píxeles muertos, detectar los eventos, su tamaño e intensidad para, posteriormente, ser enviados a tierra.

Además, se debe desarrollar un modo de pruebas en el que se pueda leer la imagen que ve el sensor para poder validar la detección de eventos.

B.3.3. Características de los usuarios

- Futuro estudiante: es el responsable de continuar con el desarrollo del instrumento, para que pueda ser integrado al satélite. Deberá contar con un sólido conocimiento de máquinas de estado y una base firme de VHDL.
- Dr. Ing. Martín Pérez y Dr. Ing. José Lipovetzky: interpretarán los datos enviados por el satélite a tierra.
- Personal de LabOSat: son los responsables del satélite en el que irá conectado el instrumento. Deberán contar con conocimiento técnico para conectar el instrumento al satélite.

B.3.4. Restricciones

Para el desarrollo del instrumento, se utilizará una FPGA de marca Xilinx y de la familia Zynq 7000. También se hará uso de sensores MTM9M001, de tipo CMOS Rolling Shutter.

B.3.5. Suposiciones y dependencias

Debido a que el instrumento está diseñado para funcionar en el espacio, se debe tener en cuenta el efecto que tienen las partículas cargadas (radiación) en los componentes electrónicos. En el caso específico de las FPGAs, a causa de que su estructura de hardware es configurable, si una partícula cargada impacta sobre la misma puede cambiar su estructura de forma no deseable y, por lo tanto, dejarla inutilizable. En el caso de las FPGA tipo radhard/radtol, tienen secciones especialmente diseñadas para evitar estos efectos.

El instrumento presupone el uso de una FPGA radhard/radtol, con un área suficiente para montar el sistema de votación y que funcione correctamente. Se recuerda que esto implica replicar el diseño del instrumento una cantidad impar de veces en la FPGA.

Además, tener en cuenta el área extra que implica añadir el bloque de votos, el direccionamiento sincrónico de los píxeles y acondicionamiento apropiado de finalización para captar una nueva imagen.

B.3.6. Requisitos futuros

El futuro estudiante que continúe con el desarrollo del instrumento debe tener en cuenta que:

- El instrumento, al ir integrado en la plataforma LabOSat, funcionará en la denominada Low Earth Orbit (LEO).
- En el espacio no hay presión, por lo tanto el estaño que se usa para soldar componentes en la PCB se gasifica.
- La PCB donde irá montada la FPGA debe ser formato PC/104.

B.4. Requisitos específicos

B.4.1. Interfaces Externas

El instrumento cuenta con dos tipos de interfaces.

La primera es la interfaz de comunicación entre el instrumento y el satélite. Mediante ella, se acondicionan los datos de salida para que el satélite pueda leerlos y transmitirlos a tierra correctamente.

La segunda es la utilizada por el usuario (Software), permite capturar imágenes arbitrariamente, así como escribir los registros de configuración y leer los eventos procesados para poder hacer una verificación del funcionamiento.

En todo momento se debe indicar, al usuario, cual es el estado en el que se encuentra el sistema.

B.4.2. Funciones

Identificación de requerimiento	RF01
Nombre del requerimiento	Captura de imagen.
Descripción del requerimiento	El sistema permitirá al usuario contar con la opción de capturar y transferir la imagen del sensor cuando se use en modo de pruebas.
Prioridad del requerimiento	Alta.

Identificación de requerimiento	RF02
Nombre del requerimiento	Lectura y escritura de los registros de configuración del sensor.
Descripción del requerimiento	Deberán poder leerse y escribirse todos los registros disponibles del sensor.
Prioridad del requerimiento	Alta

Identificación de requerimiento	RF03
Nombre del requerimiento	Detección de eventos
Descripción del requerimiento	El sistema debe ser capaz de leer el valor de los píxeles y analizar la cantidad de eventos, su tamaño y su intensidad.
Prioridad del requerimiento	Alta

Identificación de requerimiento	RF04
Nombre del requerimiento	Modo pruebas y modo automático
Descripción del requerimiento	El sistema debe contar con dos modos. De forma que uno de ellos se utilice para hacer pruebas y modificar la configuración del sensor. El segundo modo debe realizar las captura y procesamiento de forma automática, devolviendo al usuario únicamente los datos procesados. Ambos modos serán probados irradiando el dispositivo con el reactor nuclear RA6 así como utilizando fuentes de americio extraídas de detectores de humo.
Prioridad del requerimiento	Alta

Identificación de requerimiento	RF05
Nombre del requerimiento	Estado del sensor
Descripción del requerimiento	El algoritmo debe incluir un análisis de pixeles muertes con el fin de estimar en que porcentaje se encuentra funcionando el sensor. Dato que se utilizará para evaluar si es necesario cambiarlo.
Prioridad del requerimiento	Alta

B.4.3. Requisitos de Rendimiento

El instrumento debe trabajar en tiempo real y no debe perder datos de los píxeles del sensor.

B.4.4. Restricciones de Diseño

- Lenguajes: VHDL.
- Limitaciones de Hardware: imposibilidad de contar con memoria DRAM en el instrumento.
- Limitaciones de Área: el diseño debe ocupar el menor espacio posible en la FPGA, para no limitar la futura elección donde se terminará implementando el sistema.
- Interfaz con el satélite: la computadora de a bordo del satélite LabOSat tiene disponibles como interfaces Ethernet, SPI, GPIO y UART.
- Tolerancia a fallos: el sistema debe poder repetirse varias veces sobre la misma placa, para que en un futuro sea parte de un sistema de votación.
- Potencia: el LabOSat podrá alimentar al sistema con un máximo de 1 Watt.

B.4.5. Atributos del Sistema

El modo automático, al ser descripto en su totalidad en VHDL, permite ser implementado en cualquier tipo de FPGA.

B.4.6. Otros Requisitos

No aplica.

B.5. Apéndice

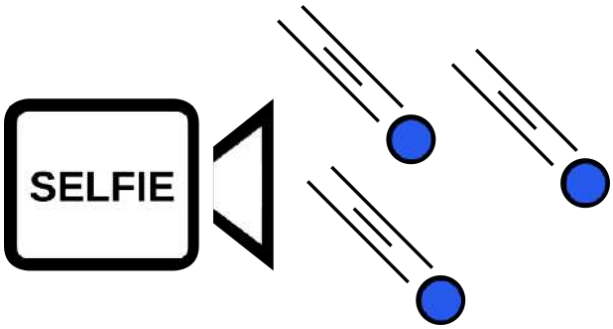
No aplica.

Apéndice C

Especificación funcional

Carrera: Ingeniería Electrónica

Scanner Electrónico de Flujo de Iones Espaciales (SELFIE)



Detector de Radiación Ionizante para aplicaciones satelitales

Especificación funcional

C.1. Ficha del documento

Fecha	Versión	Descripción	Autor/es
15 de diciembre de 2021	1.2	Modificación de la interfaz Ampliación de RFs	Mariano Rolon Mathias S. Garcia
10/08/2021	1.1	Modificaciones varias	Mariano Rolon Mathias S. Garcia
29/07/2021	1.0	Versión inicial	Mariano Rolon Mathias S. Garcia

C.2. Introducción

Este documento corresponde a la Especificación Funcional del instrumento SELFIE. Esta especificación se ha estructurado basándose en la información mencionada en el documento de Especificación de Requerimientos [1].

C.2.1. Propósito

El presente documento tiene como propósito proveer información detallada de cómo funcionará el instrumento y cuáles serán sus comportamientos deseados, con base en los requerimientos anteriormente definidos en la Especificación de Requerimientos.

Está dirigido a:

- Los desarrolladores del instrumento, quienes lo construirán.
- Los directores y solicitantes del proyecto, así como a quienes lo evaluarán, con el fin de que corroboren su funcionamiento.
- El futuro estudiante, que basándose en este prototipo construirá la versión final.

C.2.2. Alcance del proyecto

El proyecto involucra el diseño de un instrumento que debe poder procesar la imagen del sensor CMOS, filtrar los píxeles muertos, detectar los eventos, su tamaño e intensidad para, posteriormente, ser enviados a tierra.

Además, se debe desarrollar un modo de pruebas en el que se pueda leer la información generada por los distintos bloques, y comandar la captura de imágenes desde una PC.

El instrumento desarrollado en este trabajo final se trata de un prototipo. Si bien el algoritmo de detección estará escrito en VHDL en su totalidad para ser implementado en cualquier FPGA, puede verse modificado en un futuro por el estudiante que continúe con el trabajo realizado. Cuestiones como la construcción de la placa PCB y la elección final de la FPGA serán responsabilidad de él.

C.2.3. Personal involucrado

Nombre	Mariano Rolón
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	rolon.mariano96@gmail.com

Nombre	Mathías Sebastián García
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	mathiasgarcia.gs@gmail.com

Nombre	Maximiliano Antonelli
Rol	Director
Categoría profesional	Investigador Categoría IV
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	maxanto@fi.mdp.edu.ar

Nombre	Claudio Marcelo González
Rol	Codirector
Categoría profesional	Investigador categoría III
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	cmgonzal@fi.mdp.edu.ar

Nombre	Martín Pérez
Rol	Integrante del proyecto SELFIE
Categoría profesional	Investigador asistente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	ing.perezmartin@gmail.com

Nombre	José Lipovetzky
Rol	Investigador del proyecto SELFIE
Categoría profesional	Investigador independiente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	joselipo@gmail.com

C.2.4. Definiciones, acrónimos y abreviaturas

Acrónimo	Significado
RF	Requerimiento Funcional
CMOS	Complementary Metal-Oxide-Semiconductor
FPGA	Field-programmable gate array
LET	Linear Energy Transfer

C.2.5. Referencias

Título del documento	Referencia
[1] SELFIE - Especificación de requerimientos	M. S. García, M. Rolón

C.2.6. Visión general del documento

El documento se divide en cinco secciones, y, excepto la primera, cada una se encuentra dividida en distintas subsecciones.

La primera se utiliza para registrar las versiones, qué fue modificado y quién realizó dicho/s cambio/s.

En la segunda sección, se detalla una introducción al documento, los involucrados, aclaraciones para la lectura y se proporciona una visión general del mismo.

En la tercera, se describe el dispositivo y las etapas que lo componen. Su objetivo es presentar un primera versión de cómo debe funcionar el instrumento a realizar.

En la cuarta se detallan las especificaciones funcionales que debe tener, mientras que en la quinta los requerimientos no funcionales que fueron solicitados.

C.3. Descripción del dispositivo

El instrumento a desarrollar, denominado Scanner Electrónico de Flujo de Iones Espaciales (SELFIE), se compone de un sensor CMOS MT9M001 y una FPGA. Será parte de la plataforma LabOSat, según lo indica el siguiente esquema:

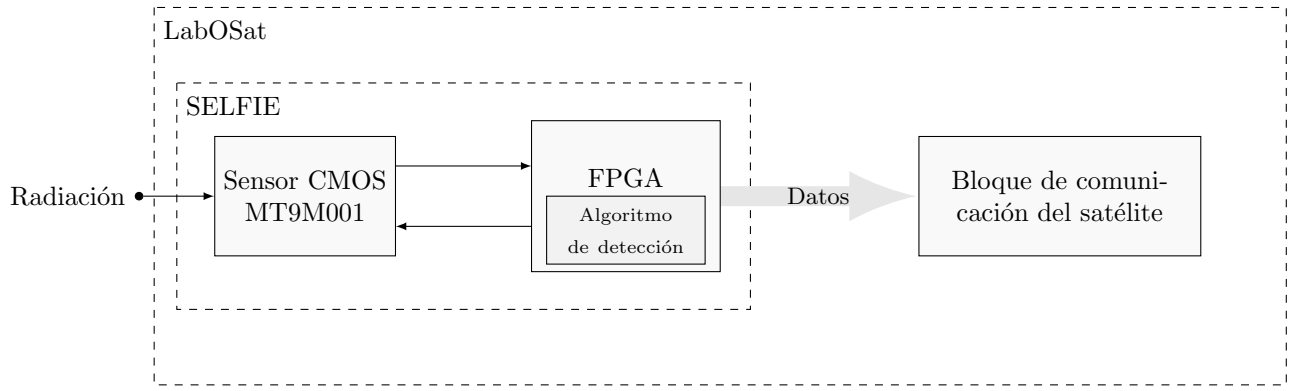


Figura C.1: Diagrama general

Cuando las partículas ionizantes impactan sobre el sensor, depositan su carga sobre los capacitores que lo conforman, lo que da como resultado un manchón blanco sobre la imagen obtenida. A los manchones se los denomina eventos y pueden tener distinta forma. También pueden tener distinta intensidad, lo que en la imagen se vería como un evento más blanco o mas negro.

Aprovechando el fenómeno descripto anteriormente, el instrumento debe obtener los datos de la intensidad de los píxeles que entrega el sensor, detectar la cantidad de eventos y su intensidad, y generar un histograma de intensidades. Posteriormente, ese histograma se almacena y luego se transmite a la controladora del satélite.

En la siguiente imagen, que corresponde a una captura real de radiación con el sensor, se pueden ver los manchones blancos que representan eventos:

Figura C.2: Imagen de irradiación del sensor

A continuación, se presenta un esquema que ejemplifica las funcionalidades del instrumento:

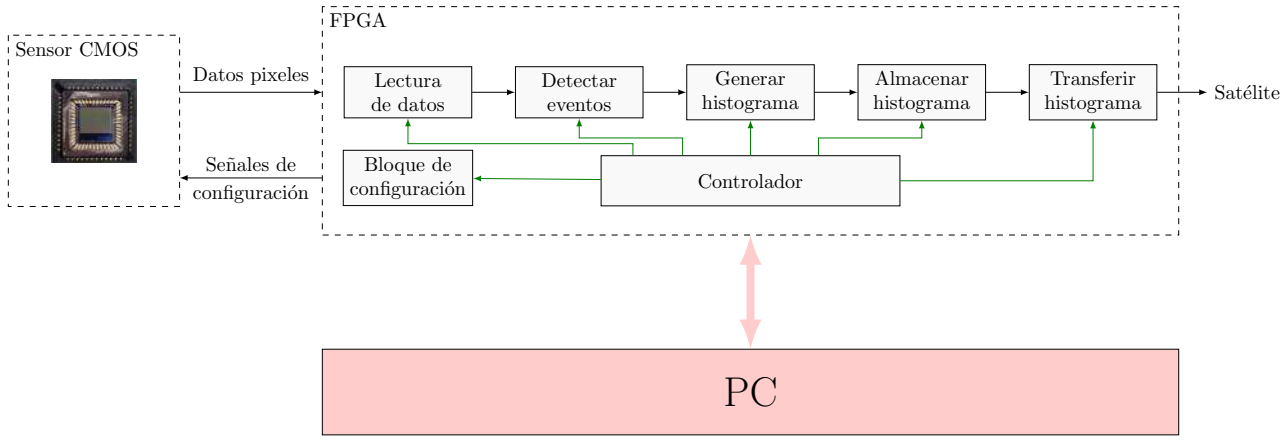


Figura C.3: Diagrama en bloques del instrumento

El instrumento tendrá dos modos de funcionamiento:

- Modo automático: el instrumento funciona de manera autónoma, es decir, que no necesita que un usuario le indique cuando debe iniciar la captura de imágenes.
- Modo debug: el instrumento puede conectarse a una computadora para iniciar la captura de imágenes, obtener las imágenes capturadas y obtener el histograma de intensidades generado, para verificar el correcto funcionamiento del bloque.

Es importante destacar que para ambos modos de funcionamiento el instrumento realiza las mismas funciones básicas. Es decir, capturar imágenes, procesarlas para reconstruir los eventos y generar un histograma de intensidades a partir de los eventos obtenidos. La única diferencia es que en el modo debug, se pueden obtener las imágenes capturadas, escribir o leer los registros del sensor e iniciar la captura desde una computadora. Las flechas en rojo indican las conexiones asociadas a este modo.

En las siguientes subsecciones se detallará la función de los bloques presentados en la Figura C.3 y de la interfaz de usuario del modo debug.

C.3.1. Bloque de configuración del sensor CMOS

Se encarga de generar las señales que requiere el sensor CMOS MT9M001 para funcionar correctamente. Asimismo, lo activa y lo desactiva convenientemente para reducir el consumo de potencia del instrumento. Además, si se encuentra activo el modo debug, debe permitir cambiar los registros de configuración del sensor y comandar el inicio de la captura de imágenes.

Este bloque responde al requerimiento funcional descripto en **C.4.2** y a la restricción de potencia de 1 Watt.

C.3.2. Bloque de adquisición de datos de píxeles

Recibe los datos de los píxeles, en formato 8 bits sin signo, y los almacena en una FIFO hasta que el bloque de detección los lea. Además, si se encuentra en modo debug, también se transmiten los datos leídos a la PC, según se indica en el requerimiento funcional descripto en **C.4.1**.

C.3.3. Bloque de detección y reconstrucción de eventos

Leyendo dato a dato, reconstruye los eventos presentes en la imagen y obtiene la intensidad total de cada evento. Tiene como salida una lista de eventos, con su intensidad y cantidad de píxeles. Responde al requerimiento funcional descripto en **C.4.3**.

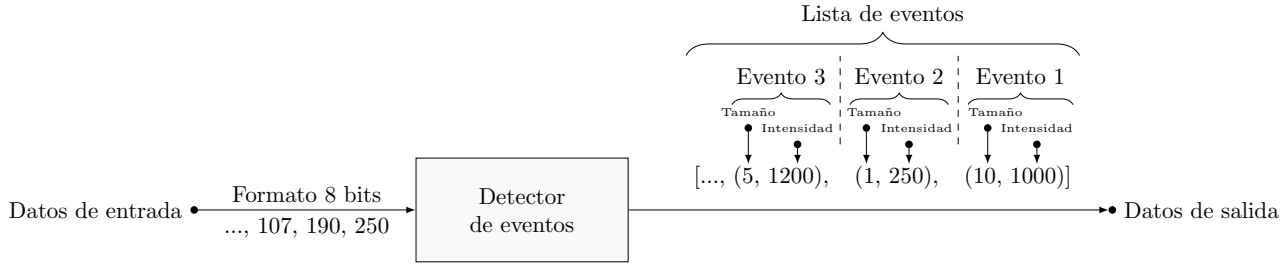


Figura C.4: Diagrama del detector

C.3.4. Bloque de generación de histograma de intensidades de eventos

Una vez finalizada la etapa de detección, se construye un histograma en base a las intensidades de los eventos, el ancho de cada bin del gráfico producido es de 1. Este bloque complementa el análisis de la imagen realizado en la etapa previa (**C.3.3**) y responde al mismo requerimiento funcional descripto en **C.4.3**.

Además, realiza el filtrado de los píxeles muertos e informa el estado del sensor en el bin 0 del histograma construido, en respuesta al requerimiento funcional en **C.4.5**.

C.3.5. Bloque de almacenamiento de histograma

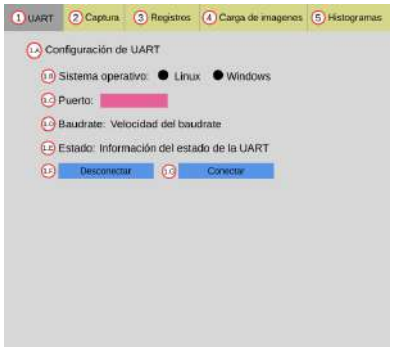
Se debe guardar el histograma de intensidades generados, ya que el satélite es el que indica cuando puede recibir datos. Además, si se encuentra en modo debug, la PC extrae los datos de este bloque.

C.3.6. Bloque de transmisión de datos

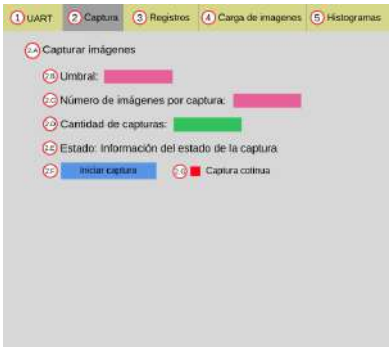
Cuando el satélite indique que está listo para recibir los datos, esta etapa será la encargada de enviarle los resultados obtenidos para el envío a tierra de los mismos.

C.3.7. Interfaz de usuario para el modo debug

La interfaz facilita el envío de comandos a través de la UART y realiza, de manera automática, la reconstrucción de la imagen a partir de los datos obtenidos del sensor. Tiene tres pestañas y se muestra a continuación:

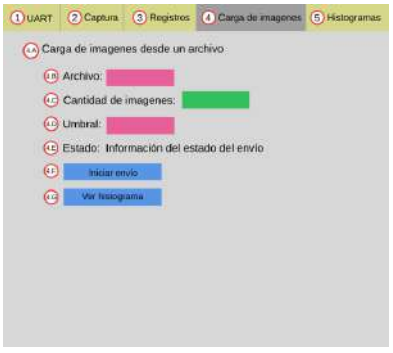


(a) Pestaña 1: Configuración UART



(b) Pestaña 2: Captura de imágenes

(c) Pestaña 3: Registros



(d) Pestaña 4: Carga de imágenes

(e) Pestaña 5: Histogramas

Figura C.5: Interfaz de usuario

Referencias:

- Campo para introducir datos.
- Seleccionar una opción.
- Botón.
- Pestaña seleccionada.
- Pestaña no seleccionada.

- Checkbox.
- Seleccionar de una lista.

Elementos:

1. **Pestaña "UART":**
   - **A.** Título "Configuración de UART".
   - **B.** Permite elegir el sistema operativo, siendo Windows y Linux las opciones.
   - **C.** Campo para introducir el puerto en el que se encuentra conectada la FPGA. En el caso de Windows, será *COM* acompañado de un número, y en el caso de Linux, *ACM* y un número.
   - **D.** Indicador donde se informa el BaudRate al que se trabaja, el cual es fijo en 115200 bits por segundo.
   - **E.** Indicador donde se informa el estado de la UART. Tiene dos estados posibles:
     - UART Desconectada.
     - UART Conectada.
   - **F.** Botón para desconectar la UART.
   - **G.** Botón para conectar la UART. El usuario solo lo podrá clickear si la UART se encuentra desconectada.
2. **Pestaña "Captura":**
   - **A.** Título "Capturar imágenes".
   - **B.** Campo para introducir el umbral que deben superar los píxeles para ser considerados parte de un evento.
   - **C.** Campo para introducir el número de imágenes que se tomarán por captura.
   - **D.** Campo para introducir el número de capturas que se realizarán.
   - **E.** Indicador de información del estado de la captura. Tiene 6 estados posibles:
     - Si la UART se encuentra desconectada, mostrará "Conectar UART".
     - Si la UART se encuentra conectada, mostrará "UART Conectada".
     - Si se clickea el botón de iniciar captura, mostrará "Iniciando captura".
     - Mientras extrae los datos, mostrará "Copiando contenido de la RAM".
     - Mientras acomoda los archivos en las carpetas correspondientes, mostrará "Moviendo archivos".
     - Tras finalizar la captura, mostrará "Captura finalizada".
   - **F.** Botón para enviar el comando para capturar imágenes.
   - **G.** Toggle de captura continua de imágenes.
3. **Pestaña "Registros":**
   - **A.** Título "Escritura/Lectura de registros".
   - **B.** Campo para completar con el registro de configuración que se desea modificar.
   - **C.** Selecciona la operación a realizar, lectura o escritura.
   - **D.** Campo para completar con los datos a escribir en el registro. Solo se habilita si el usuario ha seleccionado la operación "escritura".
   - **E.** Indicador del resultado de la operación.
     - Si la UART se encuentra desconectada, mostrará "Conectar UART".
     - Una vez conectada la UART, mostrará "UART Conectada".
     - Si la operación es lectura, mostrará el valor almacenado en el registro especificado.
     - Si la operación es escritura, mostrará el valor escrito y, luego, realiza una operación de lectura y muestra el valor leído. Esto se hace con el fin de confirmar la correcta modificación del valor.
   - **F.** Botón para ejecutar la operación seleccionada.
4. **Pestaña "Carga de Imagenes":**
   - **A.** Título "Carga de imágenes desde un archivo".
   - **B.** Campo para indicar la cantidad de imágenes a enviar desde el archivo.

C. Campo para introducir el umbral que deben superar los píxeles para ser considerados parte de un evento.

D. Indicador de información del estado de la captura. Tiene 5 estados posibles:

- Si la UART se encuentra desconectada, mostrará "Conectar UART".
- Mientras se convierte el archivo a .bin, mostrará "Convirtiendo datos".
- Durante la carga de imágenes, mostrará "Cargando imágenes".
- Durante el procesamiento de datos, mostrará "Procesando datos".
- Una vez finalizado, indicará "Captura finalizada".

E. Botón para iniciar el envío de imagen/imágenes.

F. Botón para ver el histograma resultante.

5. **Pestaña "Histogramas":**

A. Título "Creación de histogramas".

B. Listado para seleccionar carpetas que contienen datos de capturas (Imágenes y BRAM).

C. Botón para recargar las carpetas del listado anterior.

D. Listado para seleccionar el ancho de bines para realizar el histograma.

E. Indicador del estado del gráfico. Tiene 5 estados posibles:

- Si aún no ha ejecutado ninguna operación, mostrará "Esperando operación".
- Si se encuentra convirtiendo datos, mostrará "Convirtiendo datos".
- Si se encuentra creando imágenes, mostrará "Creando imágenes".
- Si está creando el histograma, mostrará "Creando histograma".
- Una vez finalizado, mostrará "Histograma finalizado".

F. Botón para crear el histograma.

G. Título "Visualizar histogramas".

H. Listado para seleccionar carpetas que contienen histogramas.

I. Botón para recargar las carpetas del listado anterior.

J. Listado para seleccionar los histogramas encontrados dentro de la carpeta seleccionada.

K. Botón para recargar los histogramas del listado anterior.

L. Botón para ver el histograma seleccionado.

El usuario solo podrá clickear los botones que corresponden a los elementos 9, 15, 16 y 22 cuando la UART esté conectada.

C.4. Especificaciones funcionales

C.4.1. RF01: Captura de imagen

El sistema cuenta con un modo de pruebas (debug) el cual se activa cuando, desde la interfaz, se conecta la UART. Una vez conectada, el instrumento queda a la espera de la señal de captura, la cual será enviada cuando se pulse en capturar. Para ello, se requieren tres parámetros adicionales:

- *Umbral de detección:* valor mínimo de un píxel para que sea considerado parte de un evento.
- *Cantidad de imágenes:* número de imágenes que se realizan por ráfaga de captura.
- *Cantidad de ráfagas:* número de ráfagas que se realizan, esto se debe a que la placa no podrá almacenar todos los píxeles.

La cantidad total de imágenes capturadas será (*Cantidad de imágenes* × *Cantidad de ráfagas*).

Sin embargo, debe haber una opción que permita hacer captura continua hasta que el usuario decida detenerlo. En este caso, se capturan imágenes según lo indicado, pero el parámetro *Cantidad de ráfagas* es ignorado.

Durante la captura, se deben almacenar los datos de 8 bits provenientes del sensor para luego poder generar, cuando el usuario lo requiera, las imágenes (en formato TIF) que caracterizan a dicha captura. Asimismo, se deben extraer los datos del histograma generado con el fin de visualizarlo en la PC.

C.4.2. RF02: Lectura y escritura de los registros de configuración del sensor

Deberán poder leerse y escribirse todos los registros disponibles del sensor. —**- new** —-

Desde la interfaz debe ser posible leer y escribir los registros del sensor según se indica en su hoja de datos.

C.4.3. RF03: Detección de eventos

El sistema debe ser capaz de leer el valor de los píxeles, analizar tanto la cantidad de eventos como el tamaño e intensidad de los mismos, para luego caracterizar la radiación recibida mediante un histograma de intensidad (LET). Este histograma permitirá al usuario reconocer qué tipo de partícula está incidiendo sobre el sensor (alpha, gamma, iones pesados, etc).

C.4.4. RF04: Modo de pruebas y modo automático

El sistema debe contar con dos modos, uno para pruebas y otro para captura automática. La función del primero (de pruebas o debug) es hacer diversas pruebas y modificar la configuración del sensor. También debe permitir el envío de imágenes desde la PC. Esto implica saltear al sensor, pasar datos directamente al bloque de detección y trabajar a partir de allí. En este último caso, no es necesario que se almacenen los datos de entrada dado que ya son conocidos. Estos dos modos permiten una verificación íntegra del sistema previo a la ejecución del modo automático.

En contraste, el modo automático debe realizar las capturas y el procesamiento sin almacenar los datos recibidos, devolviendo al usuario únicamente los datos del histograma generado.

C.4.5. RF05: Estado del sensor

El algoritmo debe incluir un análisis de píxeles muertos con el fin de estimar en que porcentaje se encuentra funcionando el sensor. Dato que se utilizará para evaluar el estado del sensor. Debido a que no es posible implementar un filtro autorregresivo, como consecuencia del espacio limitado de memoria, esta estadística debe ser producida a posteriori. Es decir, luego de la etapa de detección.

C.5. Requerimientos no funcionales

C.5.1. Requisitos de Rendimiento

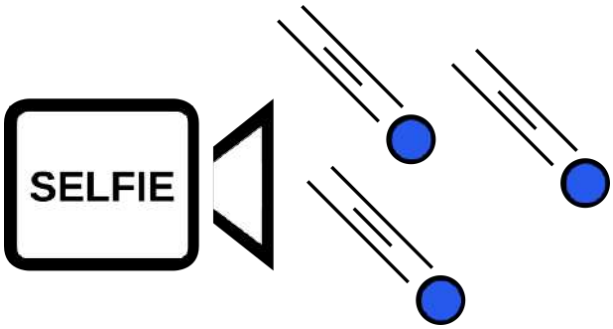
El instrumento debe trabajar en tiempo real para prevenir tanto la pérdida de datos como para maximizar la cantidad de radiación analizada durante el período de vida útil del sensor. Esto último se debe a que, incluso si se pudiesen almacenar frames para el análisis, el dispositivo seguirá recibiendo radiación mientras procesa los datos, provocando un menor aprovechamiento del sensor.

Apéndice D

Especificación técnica

Carrera: Ingeniería Electrónica

Scanner Electrónico de Flujo de Iones Espaciales (SELFIE)



Detector de Radiación Ionizante para aplicaciones satelitales

Especificación técnica

García, Mathías Sebastián
Rolón, Mariano

D.1. Ficha del documento

Fecha	Versión	Descripción	Autor/es
19/11/2021	1.0	Versión inicial	Mariano Rolon Mathias S. Garcia

D.2. Introducción

Este documento corresponde a la Especificación Técnica del instrumento SELFIE. Esta especificación se ha estructurado basándose en la información mencionada en el documento [1].

D.2.1. Propósito

Este documento tiene como finalidad describir como está compuesto el instrumento, definir como serán construidos los distintos bloques y como será la conexión entre ellos. Para ello se emplearán diagramas de flujo, descripción de estados y pseudocódigo para la correcta comprensión del trabajo a desarrollar.

D.2.2. Alcance del proyecto

El proyecto involucra el diseño de un instrumento que debe poder procesar la imagen del sensor CMOS, filtrar los píxeles muertos, detectar los eventos, su tamaño e intensidad para, posteriormente, ser enviados a tierra.

Además, se debe desarrollar un modo de pruebas en el que se pueda leer la información generada por los distintos bloques, y comandar la captura de imágenes desde una PC.

El instrumento desarrollado en este trabajo final se trata de un prototipo. Si bien el algoritmo de detección estará escrito en VHDL en su totalidad para ser implementado en cualquier FPGA, puede verse modificado en un futuro por el estudiante que continúe con el trabajo realizado. Cuestiones como la construcción de la placa PCB y la elección final de la FPGA serán responsabilidad de él.

D.2.3. Personal involucrado

Nombre	Mariano Rolón
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	rolon.mariano96@gmail.com

Nombre	Mathías Sebastián García
Rol	Desarrollador
Categoría profesional	Estudiante
Responsabilidad	Desarrollo y diseño del sistema
Información de contacto	mathiasgarcia.gs@gmail.com

Nombre	Maximiliano Antonelli
Rol	Director
Categoría profesional	Investigador Categoría IV
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	maxanto@fi.mdp.edu.ar

Nombre	Claudio Marcelo González
Rol	Codirector
Categoría profesional	Investigador categoría III
Responsabilidad	Tutelar y orientar en el diseño y seguimiento del desarrollo del proyecto
Información de contacto	cmgonzal@fi.mdp.edu.ar

Nombre	Martín Pérez
Rol	Integrante del proyecto SELFIE
Categoría profesional	Investigador asistente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	ing.perezmartin@gmail.com

Nombre	José Lipovetzky
Rol	Investigador del proyecto SELFIE
Categoría profesional	Investigador independiente
Responsabilidad	Proveer información técnica especializada sobre partículas
Información de contacto	joselipo@gmail.com

D.2.4. Definiciones, acrónimos y abreviaturas

Acrónimo	Significado
RF	Requerimiento Funcional
CMOS	Complementary Metal-Oxide-Semiconductor
FPGA	Field-programmable gate array
PL	Programmable Logic
PS	Processing System
REDGE	Rising Edge
FEDGE	Falling Edge
CLK	Clock
AMC	Automatic Mode Controller
DMC	Debug Mode Controller
SS	Self Start

D.2.5. Referencias

Título del documento	Referencia
[1] SELFIE - Especificación de requerimientos	M. S. García, M. Rolón
[2] MT9M001: 1/2-Inch Megapixel Digital Image Sensor	Aptina
[3] SELFIE - Especificación funcional	M. S. García, M. Rolón
[4] Understanding Mestastability in FPGA	Intel, Altera

D.2.6. Visión general del documento

El documento se divide en nueve secciones, siendo la primera la ficha y la segunda la introducción al mismo. La tercera sección muestra una visión general del instrumento y menciona cuáles serán las secciones a tratar posteriormente. Además, muestra como es el conexionado entre bloques a realizar.

El cuarto apartado describe, de manera íntegra, la lógica de control del dispositivo. Se detalla cada FSM, así como sus estados, las salidas que producen y los puertos que la componen. Asimismo, se muestran las conexiones internas que realiza cada bloque.

La quinta sección muestra los diagramas de flujo, muestra las señales de entrada y salida de los bloques de procesamiento. Refiere al bloque detector y al generador del histograma. Además, incluye una breve explicación a modo de complementar lo presentado anteriormente.

El sexto apartado trata sobre los bloques de almacenamiento utilizados, mientras que el séptimo detalla el comportamiento a implementar en el PS. Esto último se realiza en lenguaje C e incluye los nombres de los AXI GPIO que se utilizan para el intercambio de señales con el PL. Adicionalmente, incluye las posiciones de memoria a utilizar en función del la dirección de inicio de memoria.

La anteúltima sección realiza una muy minuciosa descripción de la máquina de estados necesaria para implementar el protocolo de comunicación del sensor. Para finalizar, se muestra la interfaz y se puntualizan las implementaciones de cada función necesaria para verificar el instrumento desarrollado.

D.3. Visión general del instrumento

El diagrama de los distintos componentes que conforman el instrumento pueden verse en la siguiente figura:

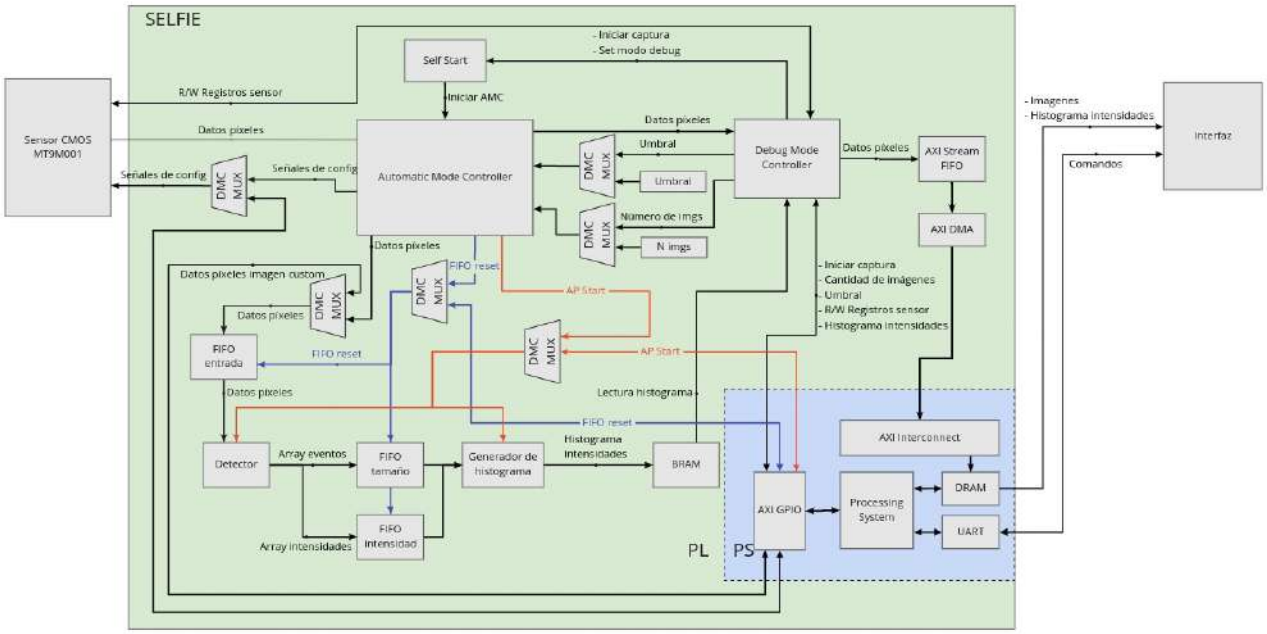


Figura D.1: Diagrama general del instrumento

Se puede notar que el instrumento se compone de:

- Sensor MTM9001 de la marca aptina, blanco y negro de 1280x1024.
- Parte lógica y de procesamiento del instrumento, implementada en una placa Zedboard, con una FPGA de la marca Xilinx Zynq7000 con dos núcleos ARM.
- Interfaz desarrollada en Python, desde donde se pueden cumplir las funciones especificadas en la EF, ver [**3**].

La parte lógica y de procesamiento del instrumento a su vez se compone de distintos bloques:

- Lógica de control, que se puede ver en la mitad superior de la figura. Compuesta por: Automatic Mode Controller, Debug Mode Controller, Self Start y el DMC Multiplexer.
- Bloques de procesamiento, compuesta por el Detector y el Generador de Histograma.
- Bloques de almacenamiento, integrado por varias FIFOs y una BRAM.
- Lógica de interfaz y configuración, implementada en el PS.

Todos los items descriptos seran detallados en profundidad en secciones posteriores.

D.3.1. Conexión de IO

El sensor CMOS se monta sobre una placa provista por el Laboratorio de Bajas Temperaturas. La misma posee puertos de entrada y salidas que se conectan a los puertos PMOD de la Zedboard. En la siguiente imagen se muestra el esquema del conexionado:

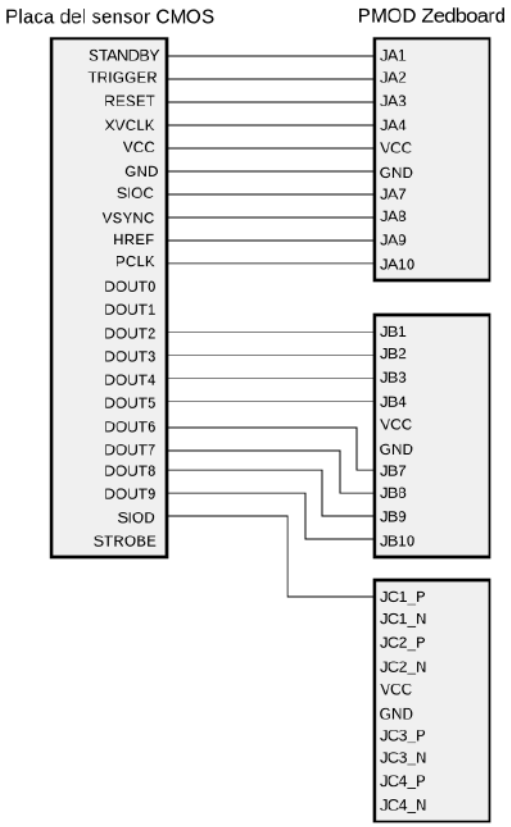


Figura D.2: Diagrama de conexiones entre el sensor y la Zedboard

Además, la Zedboard debe ser conectada a una PC a través de dos puertos USB como lo ejemplifica el siguiente esquema:

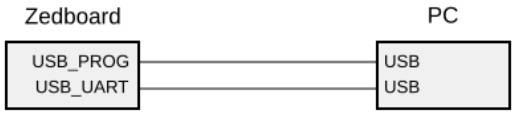


Figura D.3: Diagrama de conexiones entre la Zedboard y la PC

D.4. Lógica de control

El funcionamiento del instrumento está comandado por diversos bloques compuestos por máquinas de estado. Las mismas poseen una cantidad definida y limitada de estados, y para pasar de un estado a otro es necesaria la activación de diferentes señales, tanto internas al instrumento, como externas.

La lógica de control se compone de cuatro bloques. Estos son:

- Automatic Mode Controller (AMC).
- Debug Mode Controller (DMC).
- Self Start (SS).
- DMC Multiplexor (DMC_MUX).

D.4.1. Self Start

Este bloque espera la señal de presencia del DMC un tiempo determinado. Dependiendo de si dicha señal se encuentra en alto o en bajo pueden suceder dos casos:

- Si la señal no se encuentra presente significa que el instrumento debe funcionar en modo automático. Por lo tanto, deberá pasar al próximo estado para iniciar al AMC.
- Si la señal se encuentra activa, entonces el instrumento deberá funcionar en modo debug. Por lo tanto, esperará una señal proveniente del usuario a través de la interfaz descripta en la **Sección 7**. Una vez que la señal se activa, deberá pasar al próximo estado para iniciar al AMC.

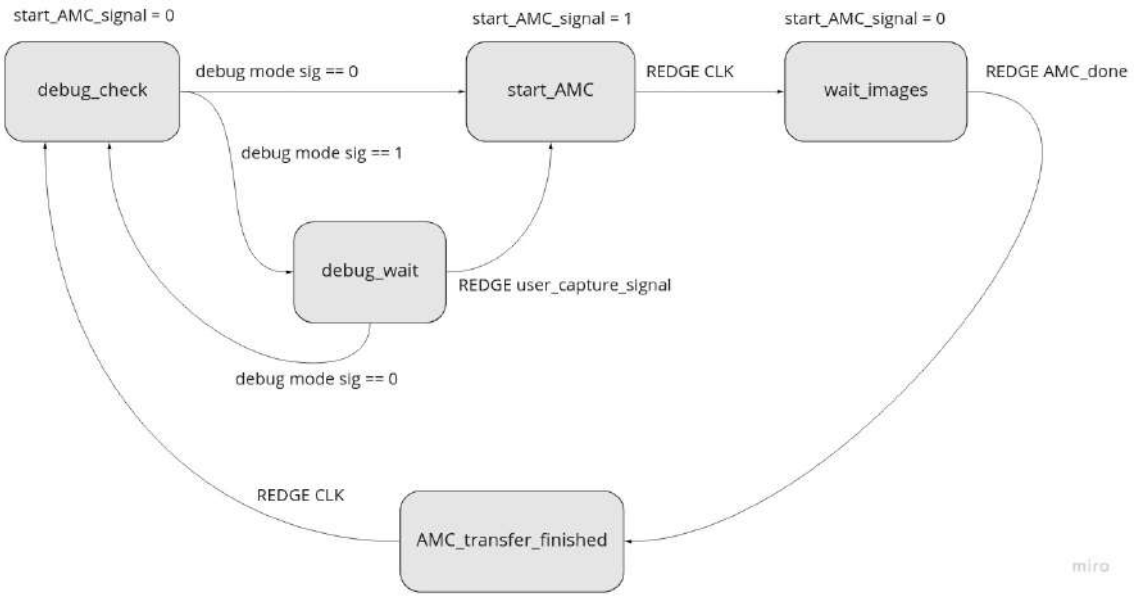


Figura D.4: Máquina de estados - Self Start

Detalle de las señales:

- Entradas:
  - **Clock**.
  - **Reset**: Vuelve al estado "Debug Check".
  - **debug_mode_signal**: Señal que indica si el modo debug está activo o no.
  - **user_capture_signal**: User Capture Signal. Señal de captura de usuario, habilita la captura de imágenes cuando el instrumento funciona en modo debug.
  - **AMC_done**: Señal que indica la finalización de la captura y procesamiento de imágenes.
- Salidas:

- **start_AMC_signal**: Le indica al AMC que debe iniciar con la captura.

La señal de "start_AMC_signal" debe enviar un pulso en alto en el estado "Start AMC" y permanecer en 0 en el resto de los estados.

D.4.2. Automatic Mode Controller

Se encarga de las siguientes tareas:

- Generar las señales del sensor de: reset, stand-by y trigger.
- Recibir y sincronizar al clock interno las señales de valid provenientes del sensor y los datos de cada pixel.
- Comandar la activación del detector, el generador de histograma y el bloque de comunicaciones.
- Resetear todas las FIFOs.

Está compuesto por los siguientes bloques:

- AMC FSM: Máquina de estados que se encarga de generar las señales de control.
- Capture Input Controller (CIC): Es el encargado de monitorear las señales del sensor para detectar cuando los datos de los píxeles son válidos.
- Sync: Bloque de sincronización para evitar la metaestabilidad.
- Pix Pulse Valid Generator: Bloque que genera una señal de dato válido para poder escribir en la FIFO de entrada.
- Pix Data FF: Flip Flop tipo D que almacena el dato hasta la llegada del siguiente.
- IMG Counter: Contador cuyo límite es la cantidad máxima de imágenes a capturar.
- AP Setup Counter: Contador para darle tiempo de establecimiento a los bloques AP.
- FIFO Reset Counter: Contador para darle tiempo de establecimiento a las FIFOs.

El diagrama de conexiones es:

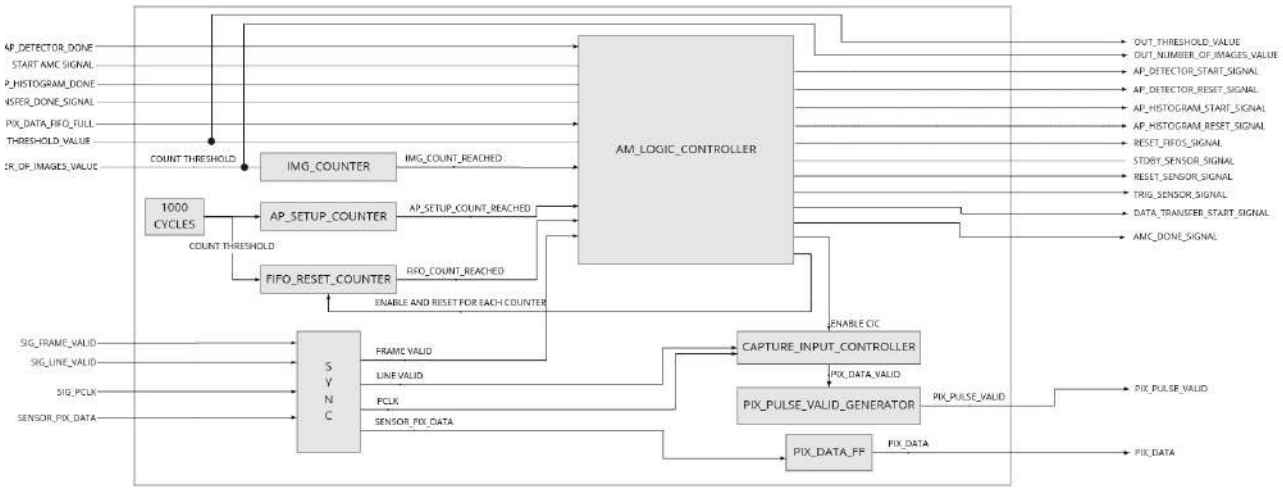


Figura D.5: Diagrama en bloques - AMC

D.4.2.1. AMC FSM

La máquina de estados presente en el AMC, llamada AMC FSM, tiene el siguiente diagrama de estados:

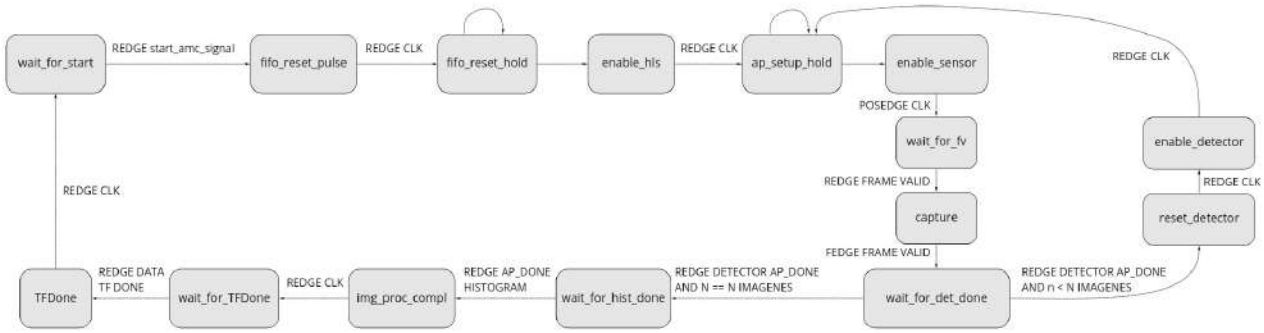


Figura D.6: Máquina de estados del Automatic Mode Controller

Las señales de salida que genera este bloque son:

- **ap_detector_reset_signal**: Reset del detector.
- **ap_detector_start_signal**: Start del detector.
- **ap_histogram_reset_signal**: Reset del bloque histograma.
- **ap_histogram_start_signal**: Start del bloque histograma.
- **reset_fifos_signal**: Reset de las fifos.
- **reset_sensor_signal**: Reset del sensor.
- **stdby_sensor_signal**: Stand-by del sensor.
- **trig_sensor_signal**: Trigger del sensor (siempre en bajo).
- **transfer_start_signal**: Avisa que los datos están listos para transferirse.
- **AMC_done_signal**: Avisa que el bloque AMC terminó de procesar y transferir los datos.
- **enable_CIC**: Habilita el bloque Capture Input Controller.
- **img_count_reset_out**: Reinicia el contador de cantidad de imágenes.
- **img_count_enable_out**: Habilita el incremento del contador cantidad de imágenes.
- **fifo_counter_reset**: Reinicia el contador de espera de las FIFOs.
- **fifo_counter_enable**: Habilita la cuenta de espera para las FIFOs.
- **ap_setup_counter_reset**: Reinicia el contador de espera para los bloques HLS.
- **ap_setup_counter_enable**: Habilita la cuenta de espera para los bloques HLS.

A continuación, se detallan los estados y los valores de las señales que producen:

- **wait_for_start**: Espera el flanco ascentente de la señal de start. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 0
```

```
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **fifo_reset_pulse**: Envía un pulso de reset a las FIFOs. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 1
reset_sensor_signal       = 0
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 1
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **fifo_reset_hold**: Espera a que las FIFOs se encuentren operativas. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 0
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 1
fifo_counter_enable       = 1
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **enable_hls**: Habilita los bloques HLS. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 1
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 1
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
```

```
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 1
ap_setup_counter_enable   = 0
```

- **ap_setup_hold**: Espera a que los bloques HLS se encuentren operativos. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 1
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 1
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 1
ap_setup_counter_enable   = 1
```

- **enable_sensor**: Habilita el sensor sacándolo del modo Stand-by. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
img_count_enable_out      = 1
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **wait_for_fv**: Espera el flanco ascendente de la señal “Frame Valid”. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
```

```
enable_CIC               = 0
img_count_reset_out      = 1
img_count_enable_out     = 0
fifo_counter_reset       = 0
fifo_counter_enable      = 0
ap_setup_counter_reset   = 0
ap_setup_counter_enable  = 0
```

- **capture**: Espera a que termine de capturar la imagen. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 1
img_count_reset_out       = 1
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **wait_for_det_done**: Espera a que el detector termine de procesar la imagen. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **reset_detector**: Reinicia el detector. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
```

```
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **enable_detector**: Habilita nuevamente el detector. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 1
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 1
stdby_sensor_signal       = 0
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 1
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 1
ap_setup_counter_enable   = 0
```

- **wait_for_hist_done**: Espera a que el bloque de histograma termine de procesar. Señales:

```
ap_detector_reset_signal  = 0
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 0
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 0
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **img_proc_compl**: Envía un pulso para iniciar la transferencia de datos. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 0
```

```
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 1
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **wait_for_TFDone**: Espera a que finalice la transferencia de datos. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 0
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 0
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

- **TFDone**: Genera un pulso de "AMC_done" avisandole al bloque Self Start que terminó su operación. Señales:

```
ap_detector_reset_signal  = 1
ap_detector_start_signal  = 0
ap_histogram_reset_signal = 1
ap_histogram_start_signal = 0
reset_fifos_signal        = 0
reset_sensor_signal       = 0
stdby_sensor_signal       = 1
trig_sensor_signal        = 0
transfer_start_signal     = 0
AMC_done_signal           = 1
enable_CIC                = 0
img_count_reset_out       = 0
img_count_enable_out      = 0
fifo_counter_reset        = 0
fifo_counter_enable       = 0
ap_setup_counter_reset    = 0
ap_setup_counter_enable   = 0
```

D.4.2.2. Sync

Todas las señales de valid que genera el sensor y los datos que entrega deben ser sincronizados al clock interno de SELFIE. Esto se debe a que el clock del sensor tiene una frecuencia y una fase diferente, por lo tanto se debe agregar un bloque sincronizador para que elimine problemas de metaestabilidad.

Esto se logra con dos flip-flops tipo D en cascada para cada una de las señales según se especifica en [4].

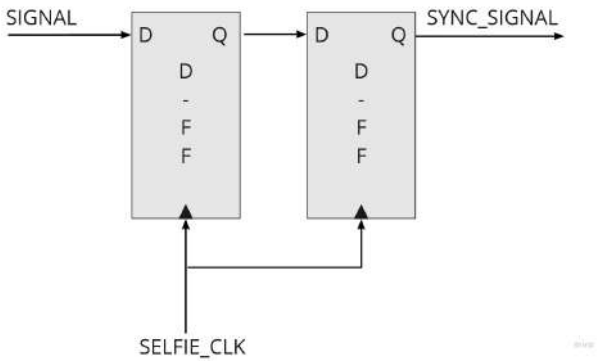


Figura D.7: Diagrama - Sync

D.4.2.3. Capture Input Controller

Este bloque se encarga de detectar el flanco descendiente de la señal PCLK proveniente del sensor y generar un pulso para latchear los datos de los píxeles.

Posee dos estados:

- Hold: espera a que llegue el flanco negativo de PCLK.
- Read: genera un pulso de PIX_DATA_VALID para señalizar que los datos de entrada son válidos y hay que leerlos. Cuando llega un flanco positivo de PCLK pasa al estado de hold.

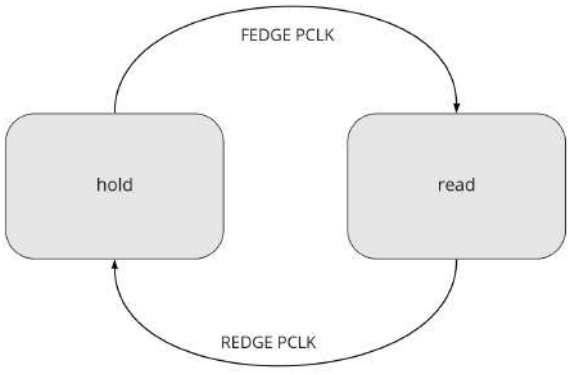


Figura D.8: Máquina de estados - Capture Input Controller

D.4.2.4. Pix Pulse Valid Generator

Debido a la diferencia entre la frecuencia de trabajo del sensor (10 MHz) y la frecuencia de procesamiento (100 MHz), es necesario un bloque que genere un pulso de 10 ns para que el dato se almacene solo una vez en la FIFO de entrada. La máquina de estados que realiza esta operación es:

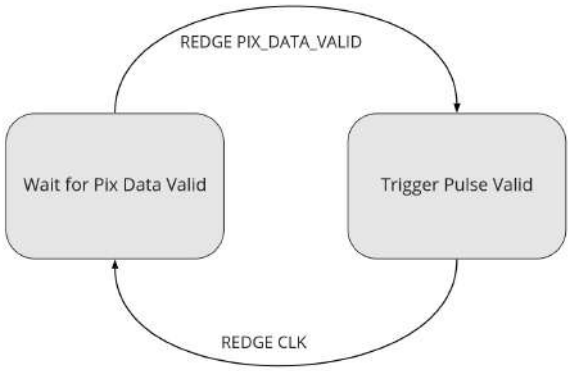


Figura D.9: Máquina de estados - Pix Pulse Valid Generator

Los estados son:

- Wait for Pix Data Valid: Espera el flanco ascendente de la señal "*PIX_DATA_VALID*" y pasa al estado *Trigger Pulse Valid*.
- Trigger Pulse Valid: Dispara un pulso en alto y vuelve a *Wait for Pix Data Valid* con el siguiente flanco de clock.

Las señales del bloque son:

- Entrada:
  - CLK: Clock.
  - RESET: tipo activo bajo. Setea el estado en *Wait for Pix Data Valid*.
  - PIX_DATA_VALID.
- Salida:
  - PIX_PULSE_VALID: Pulso para la escritura de datos.

D.4.2.5. Contadores

Su función es contar ciclos de clock cuando se le indique mediante una señal de *valid*, llamada *count_enable*. A continuación se presenta el pseudocódigo que lo modela:

```
if rising_edge(clock)
    begin
        if (!reset) or (count >= max_count)
            count = 0
        else if (count_enable)
            count = count + 1
        else
            count = count
    end

if (count == max_count)
    count_reached = 1
else
    count_reached = 0
```

Las señales son:

- Entradas:
  - Clock.
  - Reset.
  - count_enable: habilita la cuenta.

- max_count: valor máximo al que debe llegar el contador.

- Interna:
  - count: Señal de cuenta, se incrementa por cada ciclo de clock si "*count_enable*" está en alto.

- Salida:
  - count_reached: Pulso que indica que se llegó al valor máximo.

D.4.3. Debug Mode Controller

El debug mode controller tiene como finalidad enviar los datos de los píxeles capturados al PS y permitir la configuración del sensor. Su diagrama de conexiones es:

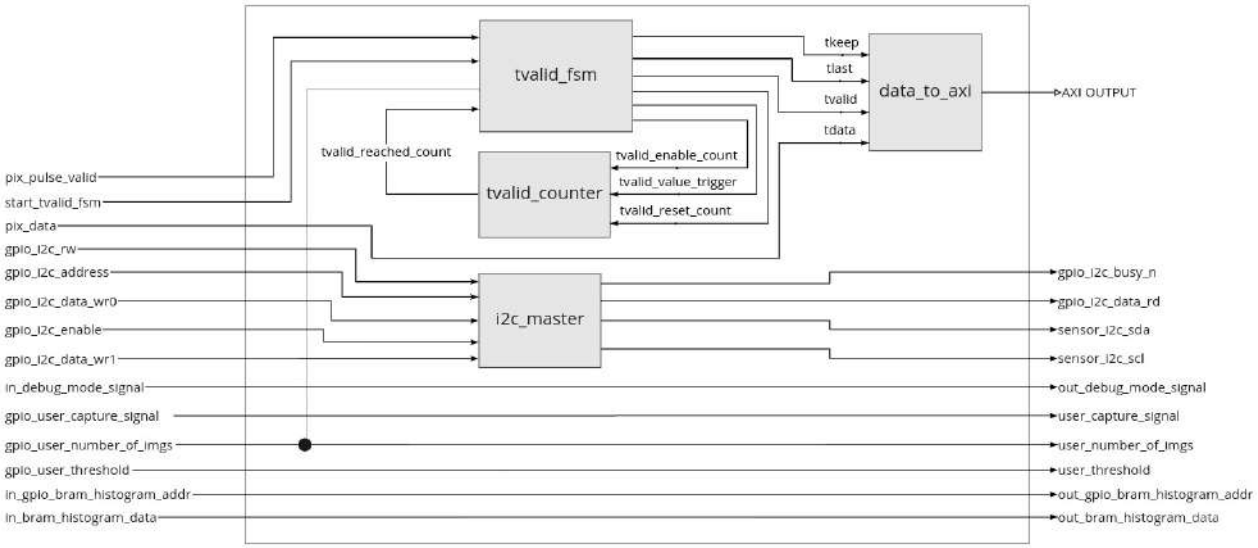


Figura D.10: Máquina de bloques - DMC

D.4.3.1. tvalid_fsm y tvalid_counter

El bloque tvalid_fsm se encarga de generar las señales necesarias por el bloque data_to_axi y enviarlas al *PS*. Por su parte, tvalid_counter, es un contador cuyo límite es la cantidad máxima de píxeles a enviar. Como el address que soporta la FIFO de transmisión es de 23 bits, se envían 6 imágenes (7864230 píxeles) como máximo y por captura.

La máquina de estados de tvalid_fsm es:

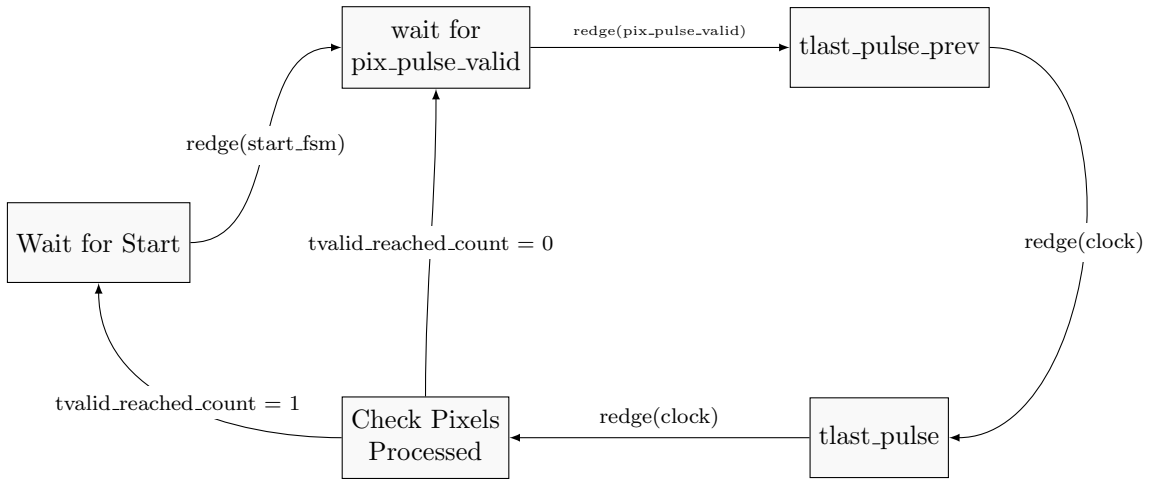


Figura D.11: Máquina de estados - tvalid_fsm

Señales de entrada:

- **clock**
- **reset**: tipo activo bajo.
- **start_fsm**: señal que da inicio al funciamiento de la FSM. Proviene del AMC.
- **pix_pulse_valid**: señal que indica que el dato es válido. Proviene del AMC.
- **number_of_images**: cantidad de imágenes a capturar.
- **tvalid_reached_count**: señal que avisa que se llegó al total de pixeles. Proviene del contador *tvalid_counter*.

Señales de salida:

- **tvalid_enable_count**: señal para incrementar el contador.
- **tvalid_reset_count**: señal para reiniciar el contador.
- **tvalid_value_trigger**: cantidad máxima de píxeles a transferir, $n_{imágenes} \times 1310720$.
- **tvalid**: señal para la transferencia al PS.
- **tkeep**: señal para la transferencia al PS.
- **tlast**: señal para la transferencia al PS.

Estados y valores de las señales de salida que generan:

- **Wait for Start**: estado de hold, espera la llegada del flanco ascendente de *start_fsm*.

```
tvalid              = 0
tvalid_enable_count = 0
tvalid_reset_count  = 0
tkeep               = 1
tlast               = 0
```

- **Wait for pix_pulse_valid**: espera el aviso de dato válido.

```
tvalid              = 0
tvalid_enable_count = 0
tvalid_reset_count  = 1
tkeep               = 1
tlast               = 0
```

- **tlast_pulse_prev**: estado para incrementar el contador

```
tvalid              = 0
tvalid_enable_count = 1
tvalid_reset_count  = 1
tkeep               = 1
tlast               = 0
```

- **tlast_pulse**: restado para generar un pulso de la señal *tlast*.

```
if tvalid_reached_count
    tlast           = 1
else
    tlast           = 0
tvalid              = 1
tvalid_enable_count = 0
tvalid_reset_count  = 1
tkeep               = 1
```

- **Check Pixels Processed**: estado para controlar si se capturaron todos los píxeles.

```
tvalid              = 0
tvalid_enable_count = 0
tvalid_reset_count  = 1
tkeep               = 1
tlast               = 0
```

D.4.3.2. data_to_axi

Es un IP Core AXI4 Lite encargado de convertir las señales de entrada a señales AXI.

Señales de entrada:

- in_axis_tlast
- in_axis_tkeep
- in_axis_tvalid
- in_data

Señales de salida:

- s_axis_tdata
- s_axis_tkeep
- s_axis_tlast
- s_axis_tvalid

D.4.3.3. i2c_master

Este bloque se encarga de la configuración del sensor. Debido a su extensión, se encuentra detallado completamente en la **Sección 6.**

D.4.4. DMC Multiplexor

Su función es la selección de parámetros y elección de señales a enviar a distintos bloques.

Su diagrama de conexión es:

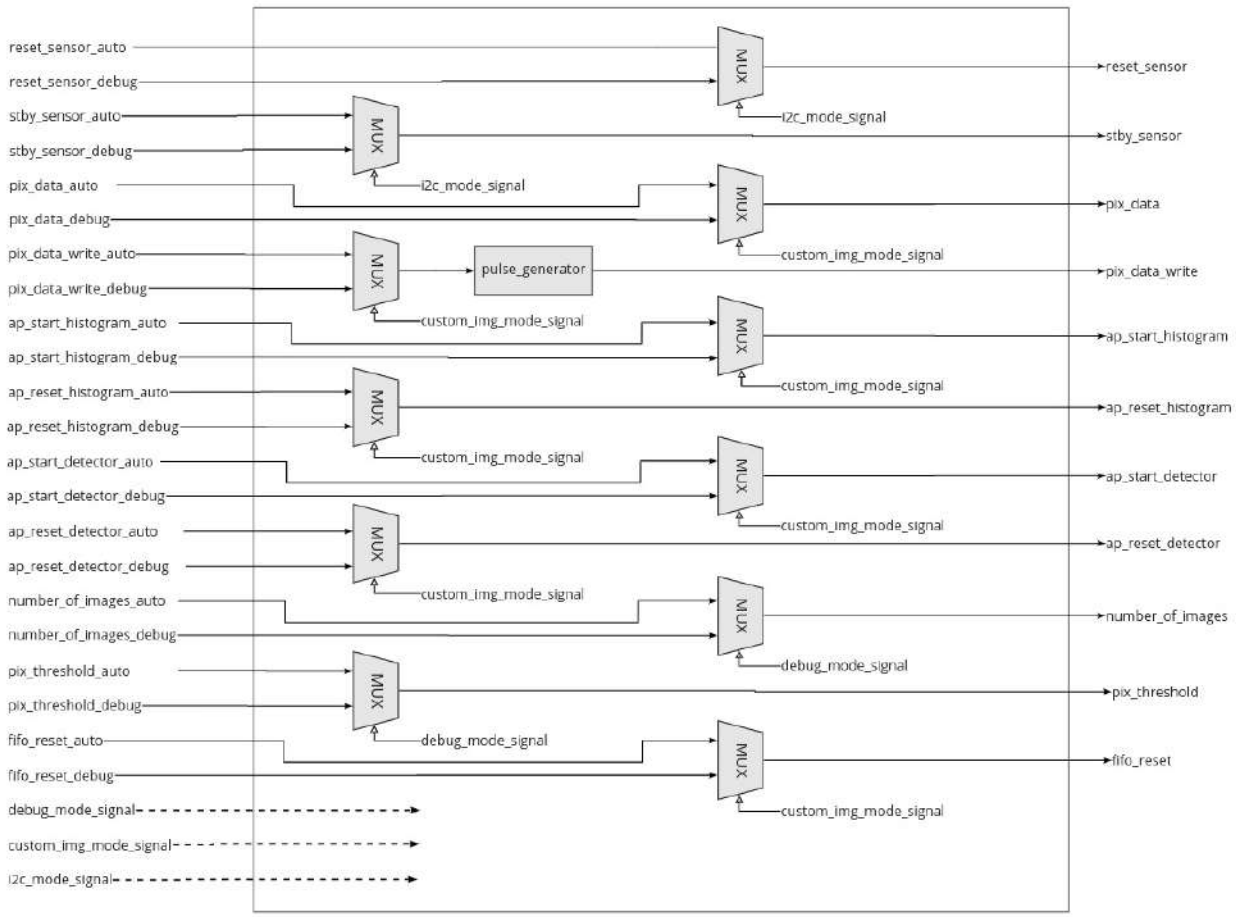


Figura D.12: Diagrama de conexiones - DMC Multiplexor

- Señales de selección: según corresponda en el diagrama, cuando están en alto, pasan a la salida la señal de entrada cuyo nombre termina en "_debug". Caso contrario, pasan a la salida la señal de entrada cuyo nombre termina en "_auto".
  - **debug_mode_signal**: indica que se encuentra activo el modo debug.
  - **custom_img_mode_signal**: indica que se le enviará una imagen desde el PS directo a la etapa de detección.
  - **i2c_mode_signal**: indica que se leerán y/o escribirán los registros.
- Señales de entrada:
  - **reset_sensor_auto**: señal de reset del sensor. Proveniente del AMC.
  - **reset_sensor_debug**: señal de el reset del sensor. Proveniente de una constante en uno.
  - **stby_sensor_auto**: señal del modo standby del sensor. Proveniente del AMC.
  - **stby_sensor_debug**: señal del modo standby del sensor. Proveniente de una constante en cero.
  - **pix_data_auto**: datos de los píxeles de entrada. Provenientes del AMC.
  - **pix_data_debug**: datos de los píxeles de entrada. Proveniente del PS a través de un puerto GPIO.
  - **pix_data_write_auto**: señal que indica que los píxeles de entrada son válidos. Proveniente del AMC.
  - **pix_data_write_auto**: señal que indica que los píxeles de entrada son válidos. Proveniente del PS a través de un puerto GPIO.
  - **ap_start_histogram_auto**: señal que comanda el inicio del bloque generador de histograma. Proveniente del AMC.
  - **ap_start_histogram_debug**: señal que comanda el inicio del bloque generador de histograma. Proveniente del PS a través de un puerto GPIO.

- **ap_reset_histogram_auto**: señal que comanda el reset del bloque generador de histograma. Proveniente del AMC.
- **ap_reset_histogram_debug**: señal que comanda el reset del bloque generador de histograma. Proveniente del PS a través de un puerto GPIO.
- **ap_start_detector_auto**: señal que comanda el inicio del bloque detector. Proveniente del AMC.
- **ap_start_detector_debug**: señal que comanda el inicio del bloque detector. Proveniente del PS a través de un puerto GPIO.
- **ap_reset_detector_auto**: señal que comanda el reset del bloque detector. Proveniente del AMC.
- **ap_reset_detector_debug**: señal que comanda el reset del bloque detector. Proveniente del PS a través de un puerto GPIO.
- **number_of_images_auto**: señal que indica el número de imágenes por captura. Proveniente de una constante seteada por el usuario.
- **number_of_images_debug**: señal que indica el número de imágenes por captura. Proveniente del PS a través de un puerto GPIO.
- **pix_threshold_auto**: señal que indica el umbral a partir del cual el valor del píxel de entrada es leído. Proviene de una constante seteada por el usuario.
- **pix_threshold_debug**: señal que indica el umbral a partir del cual el valor del píxel de entrada es leído. Proveniente del PS a través de un puerto GPIO.
- **fifo_reset_auto**: señal de reset a todas las FIFOs. Proveniente del AMC.
- **fifo_reset_debug**: señal de reset a todas las FIFOs. Proveniente del PS a través de un puerto GPIO.

- Señales de salida:
  - **reset_sensor**: señal de reset del sensor.
  - **stby_sensor**: señal de standby del sensor.
  - **pix_data**: datos de los píxeles del sensor.
  - **pix_data_write**: señal que indica que los datos de los píxeles del sensor son válidos.
  - **ap_start_histogram**: señal de start para el bloque generador de histograma.
  - **ap_reset_histogram**: señal de reset para el bloque generador de histograma.
  - **ap_start_detector**: señal de start para el bloque detector.
  - **ap_reset_detector**: señal de reset para el bloque detector.
  - **number_of_images**: señal que indica el número de imágenes por captura.
  - **pix_threshold**: señal que indica el umbral a partir del cual el valor del píxel de entrada es leído.
  - **fifo_reset**: señal de reset a todas las FIFOs.

D.5. Bloques de procesamiento

Tanto el bloque detector, como el bloque generador de histograma son programados en HLS. Por lo tanto, se comanda su inicio o reset a través de señales que corresponden al protocolo AP. Estas son:

- ap_start: si está en alto cuando ap_reset está en bajo, le indica al bloque que debe comenzar el procesamiento de datos.
- ap_reset: señal de reset activo alto.

Además, deben tener las interfaces configuradas de manera que puedan leer datos de FIFOs.

D.5.1. Bloque detector

Este bloque lee los valores de los píxeles de la FIFO de entrada, una vez que procesó el píxel actual. Como no se almacena la imagen completa, tan pronto como van llegando los píxeles se los categoriza en etiquetas, las cuales son valores numéricos y se les asocia tamaño e intensidad. Para ello, se utilizan 3 arrays de datos:

- eventFifo: almacena las etiquetas de los eventos, tiene 1283 posiciones (1 línea más 3 píxeles) con datos de 12 bits.
- eventSize: almacena el tamaño de cada evento, tiene 2048 posiciones con datos de 8 bits.
- eventIntensity: almacena la intensidad de cada evento, tiene 2048 posiciones con datos de 16 bits.

En los arrays *eventSize* y *eventIntensity* se utiliza como puntero el valor de la etiqueta.

El proceso para la detección de eventos se describe en el siguiente diagrama de flujo:

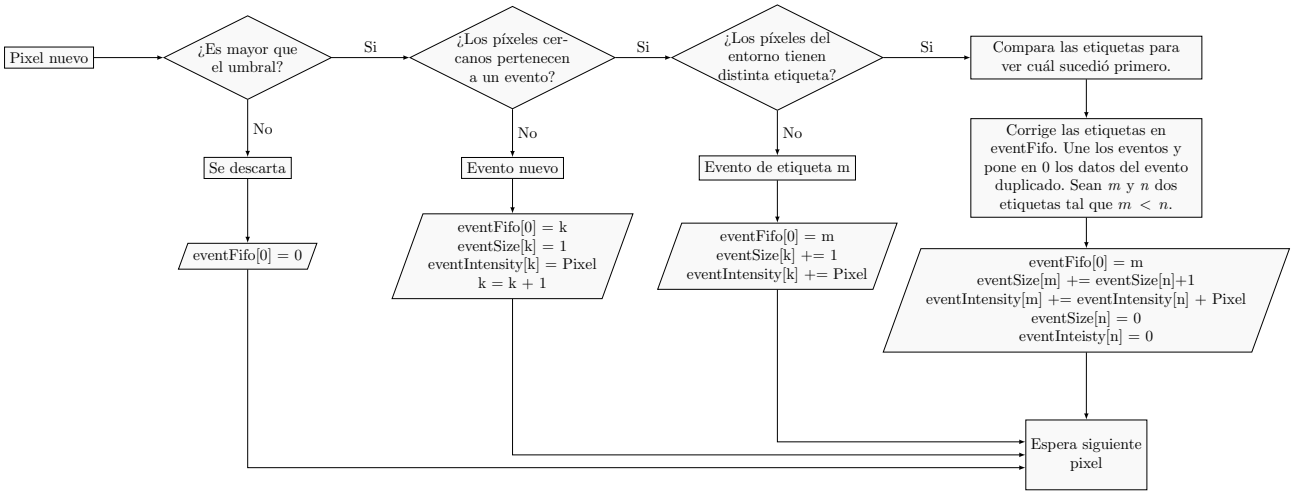


Figura D.13: Diagrama de flujo - Detector de eventos

Una vez que todos los píxeles fueron analizados y clasificados, se empiezan a transmitir los datos de salida, es decir *eventSize* y *eventIntensity*.

Señales de entrada:

- ap_detector_start
- ap_detector_reset
- umbralValue_V: valor de umbral de detección. Si el valor del dato leído está por debajo del umbral, se descarta.
- data_in_V_dout: datos de píxeles.
- data_in_empty_n: indica si la FIFO de entrada no está vacía.
- intensity_out_V_full_n: indica si la FIFO de intensidades no está llena.
- event_size_out_V_full_n: indica si la FIFO de tamaños no está llena.

Señales de salida:

- data_in_V_read: señal de lectura a la FIFO de entrada.
- intensity_out_V_din: datos de intensidad de salida.
- intensity_out_V_write: señal de escritura a la FIFO de intensidades.
- event_size_out_V_din: datos de tamaños de salida.
- event_size_out_V_write: señal de escritura a la FIFO de tamaños.

D.5.2. Bloque generador de histograma

Este bloque recibe los tamaños e intensidades de cada evento del bloque detector para producir un histograma de intensidades. Los valores para el eje de abscisas se encuentran entre 0 y 39999, el límite superior se fijó utilizando un 40 % por encima del valor máximo detectado en vídeos.

Debido a que se requiere que el ancho de los bines sea 1, entonces se utiliza un vector de 40000 posiciones en donde cada posición representa la intensidad asociada, mientras que el valor almacenado allí equivale a la cantidad de ocurrencias que tuvo.

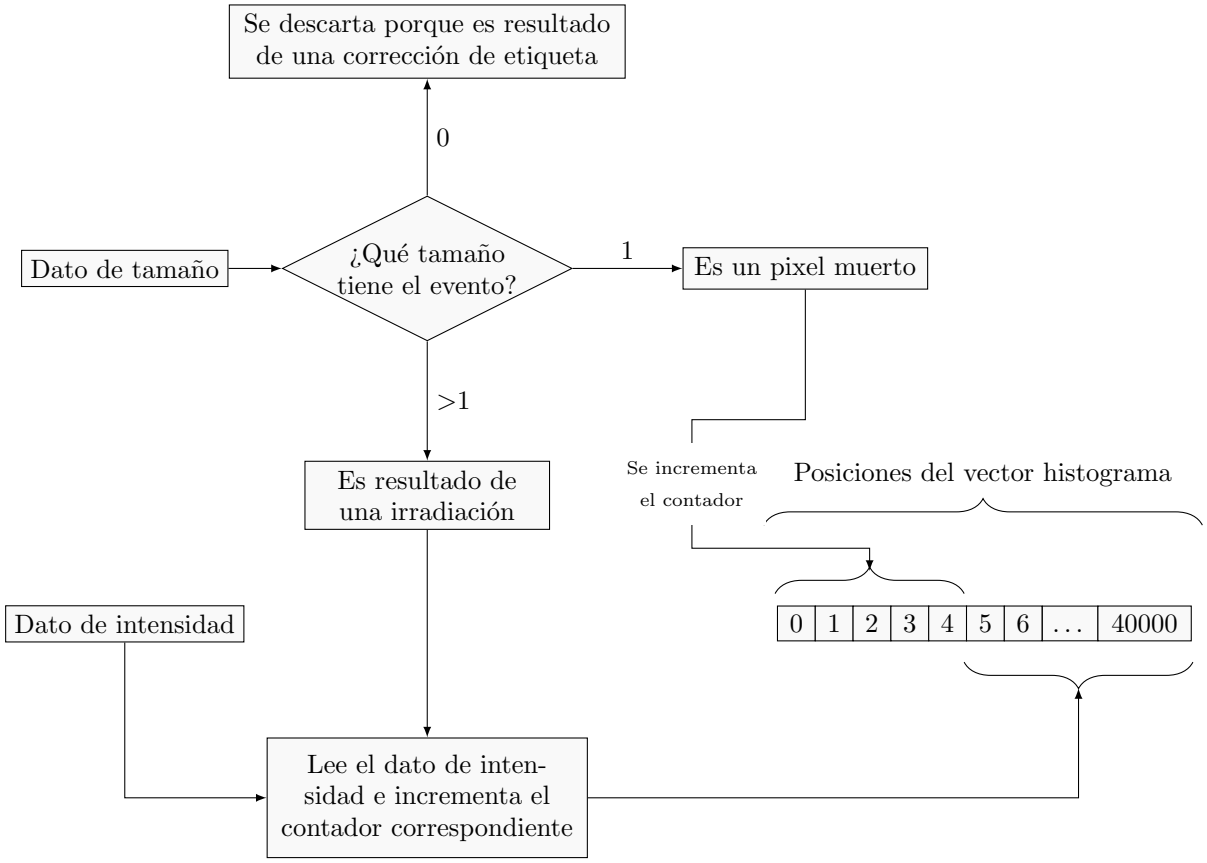


Figura D.14: Diagrama de flujo - Generador de histograma

El generador de histograma, además de los datos de tamaño e intensidad de cada evento, recibe el umbral y la cantidad de imágenes de la captura. El umbral se utiliza para colocar las primeras $2 \times umbral$ posiciones son dedicadas a almacenar la cantidad de píxeles muertos, esto es para mantener el tipo de dato como entero de 8 bits. La cantidad de imágenes es necesaria para saber que cantidad total de datos necesitará para la captura.

Señales de entrada:

- ap_histogram_start
- ap_histogram_reset
- umbralValue_V: valor de umbral de detección.
- imageNumber_V: número de imágenes por captura.
- eventSize_V_dout: datos de los tamaños de los eventos.
- eventSize_V_empty_n: indica si la FIFO de tamaños no está vacía.
- eventIntensity_V_dout: datos de las intensidades de los evento.
- eventIntensity_V_empty_n: indica si la FIFO de intensidades no está vacía.

Señales de salida:

- data_in_V_read: señal de lectura a la FIFO de entrada.

- intensity_out_V_din: datos de intensidad de salida.
- intensity_out_V_write: señal de escritura a la FIFO de intensidades.
- event_size_out_V_din: datos de tamaños de salida.
- event_size_out_V_write: señal de escritura a la FIFO de tamaños.
- eventSize_V_read: señal de lectura a la FIFO de tamaños.
- eventIntensity_V_read: señal de lectura a la FIFO de intensidades.
- histIntensity_V_we0: señal de escritura a la BRAM.
- histIntensity_V_address0: señal de dirección de escritura de la BRAM.
- histIntensity_V_d0: datos del histograma de intensidades.

D.6. Bloques de almacenamiento

D.6.1. FIFOs

Tienen como funcionalidad la prevención de la pérdida de datos entre bloques, en caso de que alguno demorase un más de lo esperado en procesarlos. No almacenarán una gran cantidad de datos ya que, tanto el algoritmo de detección como el generador de histograma trabajan en tiempo real, deberían estar la mayor parte del tiempo vacías.

De tipo FWFT, First Word Fall Through. A diferencia de las FIFOs estándares, que requieren un pulso de read para colocar el dato de entrada a la salida siempre, las FWFT pueden situar el primer dato de entrada directamente en la salida. Es decir, para el primer dato no necesitan la señal de read.

En las subsecciones subsecuentes se detallan cuestiones constructivas según la etapa en la que se encuentran.

D.6.1.1. FIFO de entrada

Se tomó como margen de seguridad casi una fila de píxeles completa. Sus parámetros son:

- Ancho de bits: 8.
- Profundidad: 1024.

D.6.1.2. FIFO de tamaño de eventos

Se tomó como margen de seguridad un cuarto del tamaño del array transmitido en esta etapa. Sus parámetros son:

- Ancho de bits: 8.
- Profundidad: 512.

D.6.1.3. FIFO de intensidad de eventos

Se tomó como margen de seguridad un cuarto del tamaño del array transmitido en esta etapa. Sus parámetros son:

- Ancho de bits: 16.
- Profundidad: 512.

D.6.2. BRAM

Su función es almacenar el histograma procesado hasta que el satélite esté listo para transmitir. Una vez que la transferencia de datos se haya completado, debe reiniciarse para asegurar que la próxima captura no contendrá ningún dato de la captura actual.

Sus parámetros son:

- Ancho de bits: 16.
- Cantidad de posiciones: 40000.
- Dual port.

D.7. Lógica de interfaz y comunicación

Para que la FPGA comprenda los comandos recibidos por UART, y poder enviarlos, se instancia el bloque "Processing System" en Vivado. Posteriormente, se programa su comportamiento utilizando la herramienta Vivado SDK. En lenguaje C, se implementa el siguiente diagrama de flujo:

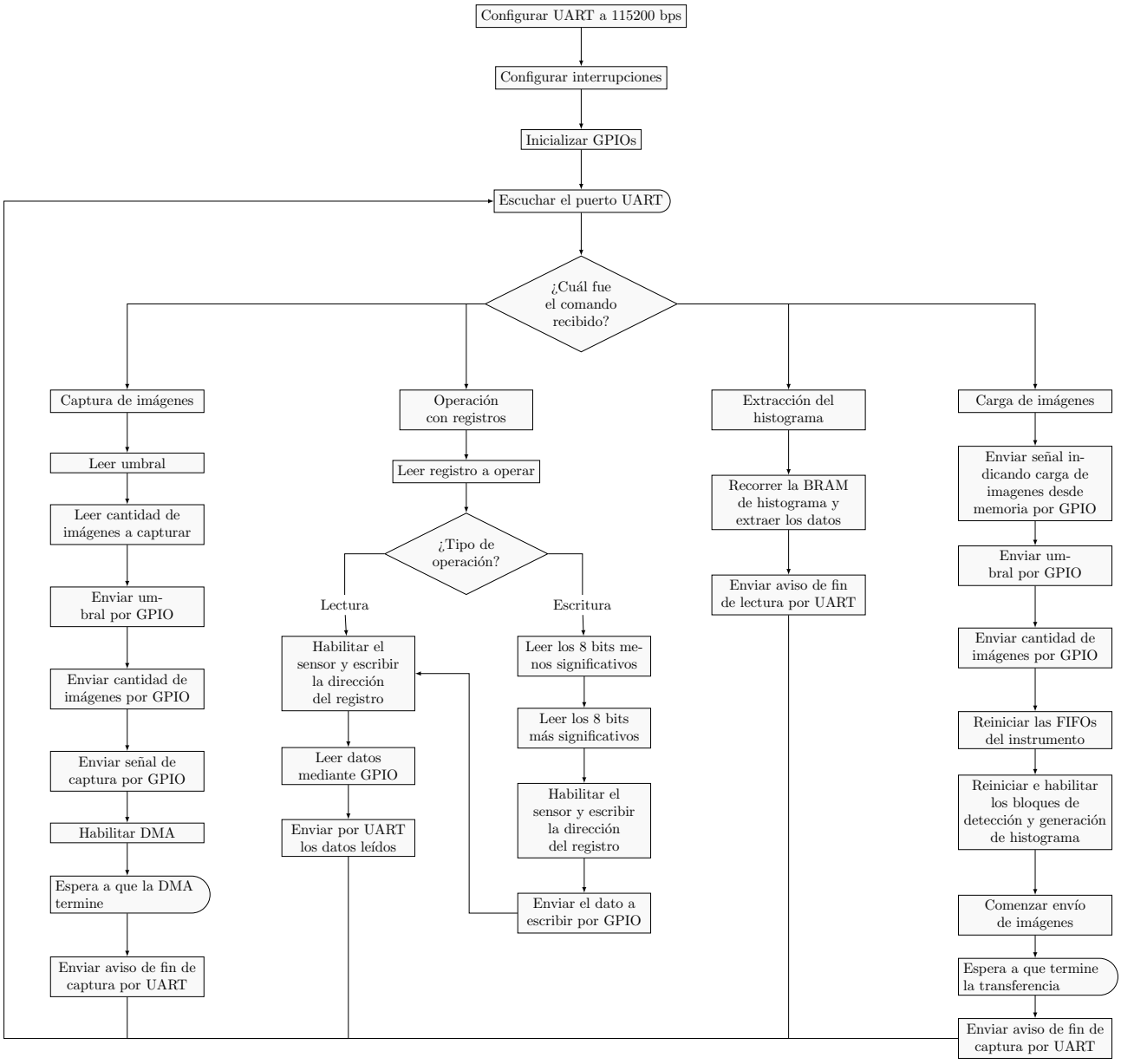


Figura D.15: Diagrama de flujo del PS

Los GPIO a utilizar son:

- Captura finalizada: XPAR_AXI_GPIO_AMC_DONE_DEVICE_ID
- Reinicio de FIFOs: XPAR_AXI_GPIO_FIFO_RESET_DEVICE_ID
- Reinicio de bloque de histograma: XPAR_AXI_GPIO_AP_RESET_HISTOGRAM_DEVICE_ID
- Habilitación de bloque de histograma: XPAR_AXI_GPIO_AP_START_HISTOGRAM_DEVICE_ID
- Reinicio de bloque de detección: XPAR_AXI_GPIO_AP_RESET_DETECTOR_DEVICE_ID
- Habilitación de bloque de detección: XPAR_AXI_GPIO_AP_START_DETECTOR_DEVICE_ID
- Carga de imagen desde memoria: XPAR_AXI_GPIO_CUSTOM_IMG_MODE_DEVICE_ID
- Activar modo i2c: XPAR_AXI_GPIO_I2C_MODE_SIGNAL_DEVICE_ID

- Habilitar bloque i2c: XPAR_AXI_GPIO_I2C_ENABLE_DEVICE_ID
- Señal de lectura/escritura: XPAR_AXI_GPIO_I2C_RW_DEVICE_ID
- Dirección del registro a operar: XPAR_AXI_GPIO_I2C_ADDRESS_DEVICE_ID
- 8 bits menos significativos del dato a escribir en el registro: XPAR_AXI_GPIO_I2C_DATA_WR0_DEVICE_ID
- 8 bits más significativos del dato a escribir en el registro: XPAR_AXI_GPIO_I2C_DATA_WR1_DEVICE_ID
- Datos leídos del registro: XPAR_AXI_GPIO_I2C_DATA_RD_DEVICE_ID
- Señal busy del bloque i2c: XPAR_AXI_GPIO_I2C_BUSY_N_DEVICE_ID
- Valor de píxel al enviar imágenes desde memoria: XPAR_AXI_GPIO_PIX_DATA_DEVICE_ID
- Píxel válido al enviar imágenes desde memoria: XPAR_AXI_GPIO_PIX_DATA_WRITE_DEVICE_ID
- Dato extraído del histograma: XPAR_AXI_GPIO_BRAM_HISTOGRAM_ADDR_DEVICE_ID
- Dirección de la BRAM del histograma: XPAR_AXI_GPIO_HISTOGRAM_DATA_DEVICE_ID
- Señal inicio de capture: XPAR_AXI_GPIO_USER_CAPTURE_SIGNAL_DEVICE_ID
- Cantidad de imágenes a capturar: XPAR_AXI_GPIO_USER_NUMBER_OF_IMAGES_DEVICE_ID
- Valor de umbral para la captura: XPAR_AXI_GPIO_USER_THRESHOLD_DEVICE_ID

Las definiciones mencionadas anteriormente se encuentran, igualmente, en el archivo “xgpio.h”, al que se puede acceder fácilmente desde la interfaz de Vivado SDK.

Los GPIO que utilizarán interrupciones son:

- Captura finalizada.
- Señal busy del bloque i2c.

Las direcciones de memoria se definen en base a la dirección de inicio de memoria (MEM_BASE_ADDR), usualmente es 0x01000000. Las posiciones a utilizar son:

- Extremo inferior del buffer de recepción: MEM_BASE_ADDR + 0x00300000.
- Extremo superior del buffer de recepción: MEM_BASE_ADDR + 0x025C0000.
- Posición inicial para los datos del histograma: MEM_BASE_ADDR + 0x035C0010.
- Posición inicial para los datos de píxeles al cargar imagen desde la memoria: MEM_BASE_ADDR + 0x035C9C60.

D.8. Configuración del sensor

Acorde a la hoja de datos del sensor ([2]), para la lectura y escritura se debe utilizar una interfaz serie de dos cables controlado por un *serial clock* (SCLK). El sensor funciona como *slave* en la interfaz, mientras que la en la FPGA se encuenta el bloque *master*, el cual gobernará el SCLK mencionado. Esta interfaz es similar a *i2c* pero con ciertas modificaciones.

La interfaz define varios códigos de transmisión diferentes, de la siguiente manera:

- Un bit de *start*.
- La dirección de 8 bits del dispositivo *slave*.
- Un bit de reconocimiento (ACK).
- El mensaje de 8 bits.
- Un bit de *stop*.

El proceso de escritura del registro es el siguiente, luego de cada paso el *master* espera un ACK de parte del sensor:

1. Se envía un bit de *start* y la dirección 0xBA.
2. Se envía la dirección del registro que se desea escribir.
3. Se envían los 8 bits más significativos del dato a escribir.
4. Se envían los 8 bits menos significativos del dato a escribir.
5. Se envía un bit de stop y no se espera respuesta del sensor.

Cuando se desea leer un registro del sensor, el proceso se modifica levemente:

1. Se envía un bit de *start* y la dirección 0xBA. El sensor responde con un ACK.
2. Se envía la dirección del registro que se desea leer. El sensor responde con un ACK.
3. Se envía la dirección 0xBB. El sensor responde con un ACK.
4. Se reciben los 8 bits más significativos del registro. El *master* responde con un ACK.
5. Se reciben los 8 bits menos significativos del registro. El *master* responde con un NACK.
6. Se envía un bit de stop y no se espera respuesta del sensor.

Este proceso se encuentra más en detalle en la página 14 de **[2]**. La máquina de estados, de tipo Mealy, a implementar es la siguiente:

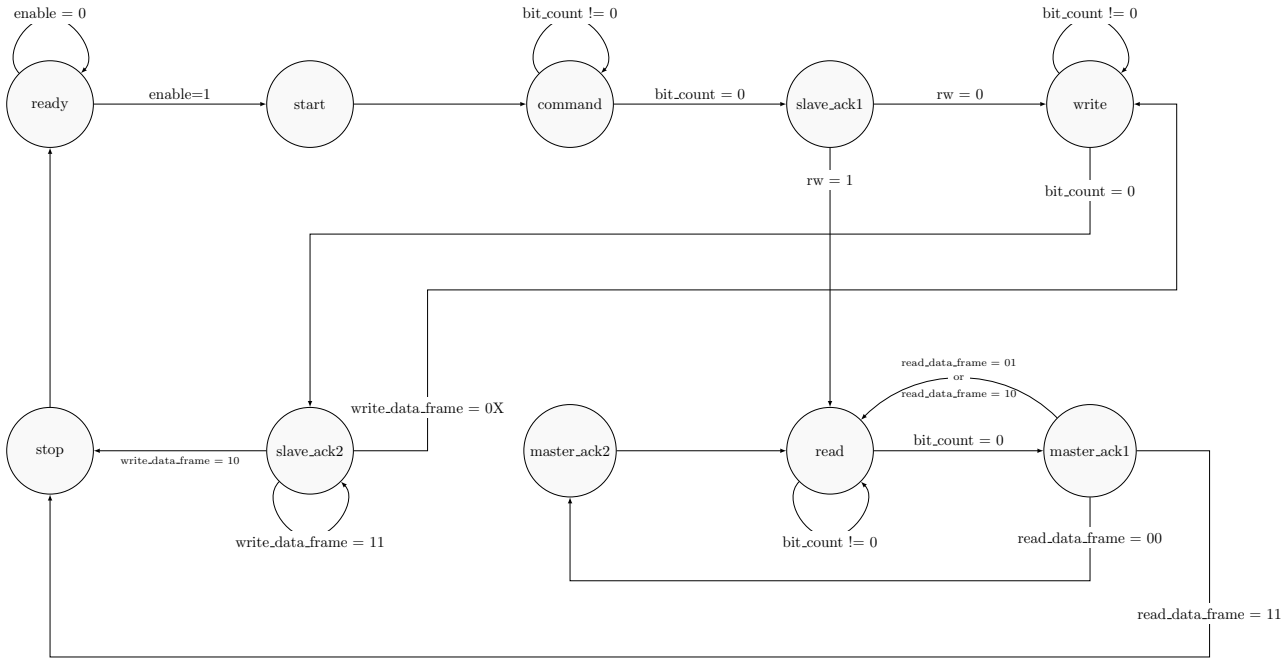


Figura D.16: Máquina de estados - Lectura/Escritura de registros

Señales de entrada:

- **Clock**
- **Reset**: Tipo activo bajo.
- **Enable**: Señal de habilitación.
- **rw**: Lectura/Escritura.
- **addr**: Address del registro a escribir/leer.
- **data_wr0** (8 bits): Datos a escribir en la parte menos significativa.
- **data_wr1** (8 bits): Datos a escribir en la parte más significativa.

Señales de salida:

- **busy**: Indica si existe algún proceso en ejecución.
- **data_rd (16 bits)**: Datos leídos del *slave*.

Señales inout:

- **sda**: Datos de salida serie.
- **scl**: Clock para los datos de salida.

Señales y variables internas para la transición de estados:

- **bit_cnt**: Contador que indica la cantidad de bits que faltan transmitir/leer.
- **read_data_frame** (2 bits)
  - 00: Registro a leer.
  - 01: Se transmite 0xBB, que indica una operación de lectura.
  - 10: Se leen los 8 bits más significativos.
  - 11: Se leen los 8 bits menos significativos.
- **write_data_frame** (2 bits)
  - 00: Registro a escribir.
  - 01: 8 bits menos significativos del dato a escribir.
  - 10: 8 bits más significativos del dato a escribir.
  - 11: Esta asignación no sucede nunca.

Se deben generar dos clocks:

- Data clock: Desfasado en $\pi/4$ del clock global
- scl clock: En fase al clock global

Comportamiento de los estados cuando se produce un flanco ascendente del data clock:

- **ready**:

```
if enable = 1
    Levantar flag busy
    Leer el address dado
    Leer los datos a escribir si corresponde
    Asignar estado start
else
    Bajar flag busy
    Asignar estado ready
```

- **start**:

```
Mantener flag busy en alto
Asignar, a una variable interna sda, el LSB de 0xBA
Asignar estado command
```

- **command**:

```
if bit_cnt = 0
    Poner en 1 la señal sda interna
    Asignar 7 a bit_cnt
    Asignar siguiente estado, slv_aclk1
else
    Decrementar bit_cnt en 1
    Asignar a la sda interna el bit bit_cnt-1 de 0xBA
    Asignar siguiente estado, command
```

- **slave_ack1**:

```
Asignar a sda interna el LSB del registro a leer/escribir
if rw = 0
    Siguiente estado write
else
    Siguiente estado read
```

- **slave_ack2**:

```
Cuando (write_data_frame) asignar:
    00: write_data_frame 01
        el bit bit_cnt de los 8 MSB de los datos a escribir
        Siguiente estado write
    01: write_data_frame 10
        el bit bit_cnt de los 8 LSB de los datos a escribir
        Siguiente estado write
    10: write_data_frame 00
        Siguiente estado stop
```

- **master_ack1**:

```
Según (read_data_frame) asignar:
    00: read_data_frame 01
        sda interna en 1
        Siguiente estado master_ack2
    01: read_data_frame 10
        sda interna en 1
        Siguiente estado read
    10: read_data_frame 11
        sda interna en 1
        Siguiente estado read
    11: read_data_frame 00
        Siguiente estado stop
```

- **master_ack2**:

```
Asignar a la sda interna el LSB de 0xBB
Siguiente estado read
```

- **read**:

```
if bit_cnt = 0
    sda interna en 1
    Asignar 7 a bit_cnt
    Siguiente estado master_ack1
    Cuando write_data_frame:
        10: Asignar los 8 bits más significativos a data_rd
        11: Asignar los 8 bits menos significativos a data_rd
else
    Decrementar en 1 bit_cnt
    Según (write_data_frame) asignar a la sda interna el bit:
        00: bit_cnt-1 del registro a leer
        01: bit_cnt-1 de 0xBB
    Siguiente estado read
```

- **write**:

```
Mantener en alto flag de busy
if bit_cnt = 0
    sda interna en 1
    Asignar 7 a bit_cnt
    Siguiente estado slave_ack2
else
    Decrementar en 1 bit_cnt
    Según (write_data_frame) asignar a sda interna el bit:
        00: bit_cnt-1 del registro a escribir
        01: bit_cnt-1 de los 8 bits menos significativos a escribir
        01: bit_cnt-1 de los 8 bits más significativos a escribir
    Siguiente estado write
```

- **stop**:

```
Bajar flag de busy
Siguiente estado ready
```

Comportamiento de los estados cuando se produce un flanco descendente del data clock:

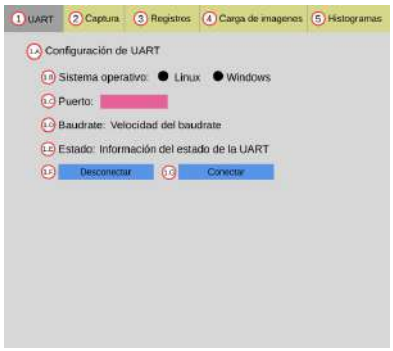
- start: si el serial clock está desactivado, activarlo
- read: si read_data_frame está en 01 o en 10, asignar a una variable interna lo que se reciba por *sda*
- stop: deshabilitar el serial clock.

Al resetear el dispositvo, se deben configurar los siguientes parámetros:

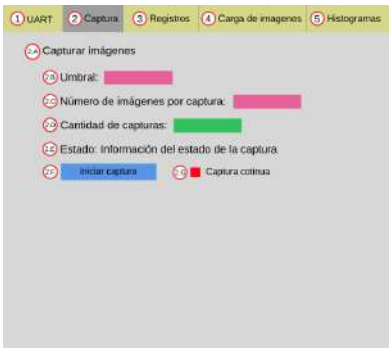
- Estado actual: **ready**.
- Señal *busy*: en alto.
- Limpiar el puerto de lectura.
- Deshabilitar el puerto *scl*.
- Colocar el puerto *sda* en alta impedancia.
- Asignar 7 a la variable interna *bit_cnt*.

D.9. Interfaz de usuario

La interfaz presentada en el documento **[3]** es:



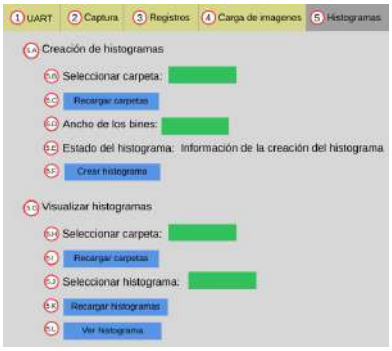
(a) Pestaña 1: Configuración UART



(b) Pestaña 2: Captura de imágenes

(c) Pestaña 3: Registros

(d) Pestaña 4: Carga de imágenes



(e) Pestaña 5: Histogramas

Figura D.17: Interfaz de usuario

Referencias:

- Campo para introducir datos.
- Seleccionar una opción.
- Botón.
- Pestaña seleccionada.
- Pestaña no seleccionada.
- Checkbox.
- Seleccionar de una lista.

D.9.1. Pestañas

D.9.1.1. UART

Al hacer clic en conectar, se deben recolectar los datos de sistema operativo y puerto de manera que:

```
if osystem = linux
    puertoTotal = '/dev/tty' + puerto
else
    puertoTotal = 'COM' + puerto
Intentar conectarse a la placa y si se logró:
Si no se pudo conectar al puerto, indicar que el puerto no es correcto.
Si falta indicar el puerto o el sistema, indicar que faltan datos.
```

Al desconectar la placa, se debe informar que se cerró correctamente y, si no fue utilizada, que se cerró sin usarse.

Además, los botones son excluyentes. Cuando se conecta la placa, se deshabilita el botón de "Conectar" y se habilita el botón "Desconectar", lo mismo sucede en el orden inverso.

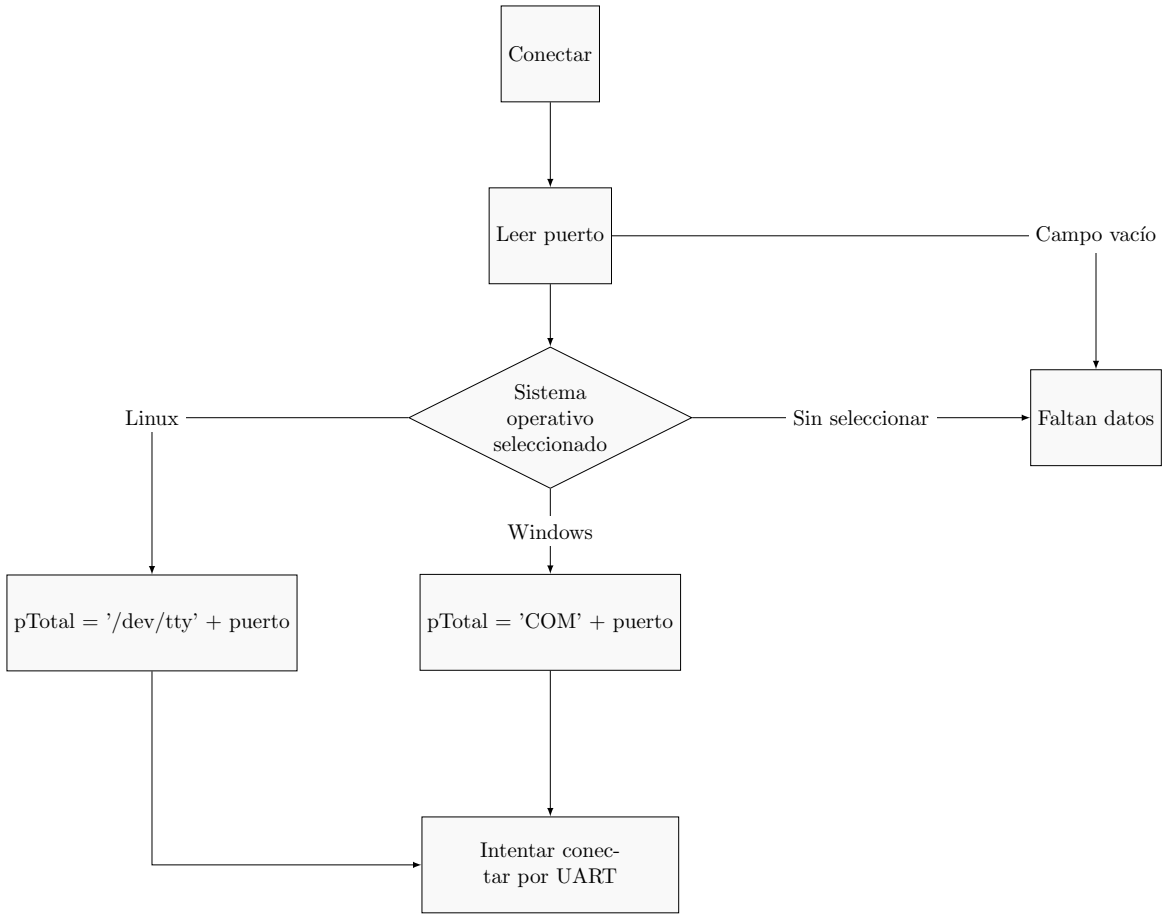


Figura D.18: Diagrama de flujo al presionar "Conectar"

D.9.1.2. Captura

Cuando se presione el botón de captura:

```
    capturaContinuaInicial = capturaContinua
    seguirCapturando = True
    Crear carpeta [fechaDeCaptura].run
    Crear archivo config.txt
    while (seguirCapturando)
        Enviar comando de captura
        Enviar parámetros (umbral y cantidad de imágenes)
        Si terminó la captura
            Crear directorioDeCaptura
            Enviar el comando TCL para extraer los datos de los pixeles en formato texto:
                set logfile [open "directorioDeCaptura/pix-data.txt" "w"]
puts $logfile [mrd -size b BRAMmemoryAddress nImagenes*1310720]
close $logfile

            Enviar los comandos TCL para extraer los datos del histograma en formato texto:
                set logfile [open "directorioDeCaptura/bramData.txt" "w"]
                puts $logfile [mrd -size b BRAMmemoryAddress 40000]
                close $logfile

            Mover directorioDeCaptura adentro de [fechaDeCaptura].run
```

```
if (capturaContinuaInicial)
    if (capturaContinua)
        seguirCapturando = True
    else
        seguirCapturando = False
else
    aumentar contador de capturasHechas
    seguirCapturando = (capturasHechas < cantidadDeCapturas)
```

El archivo config.txt tiene los siguientes campos:

- Umbral
- ImagenesPorCaptura
- RafagasDeImagenes
- ImagenesCreadas

Estos datos serán leidos a la hora de crear las imágenes y los histogramas correspondientes.

Respecto a los comandos TCL, la extracción de las imágenes se realiza en formato .txt y el parámetro *nImagenes* indica la cantidad de píxeles que debe extraer. El histograma, por su parte, también se extrae en formato .txt, el valor 40000 está fijo y se debe a que la cantidad de valores a extrar es constante.

En ambos casos, el "directorioDeCaptura" es una carpeta que varía, propia de cada captura y lleva una marca temporal. Las direcciones base de memoria (ImageMemoryAddress y BRAMmemoryAddress) deben ser las que se hayan definido en el PS de la FPGA.

Por último, el botón "Iniciar captura" estará deshabilitado mientras que la UART se encuentre desconectada.

D.9.1.3. Registros

Al presionar el botón "Ejecutar", el diagrama de flujo producido es el siguiente:

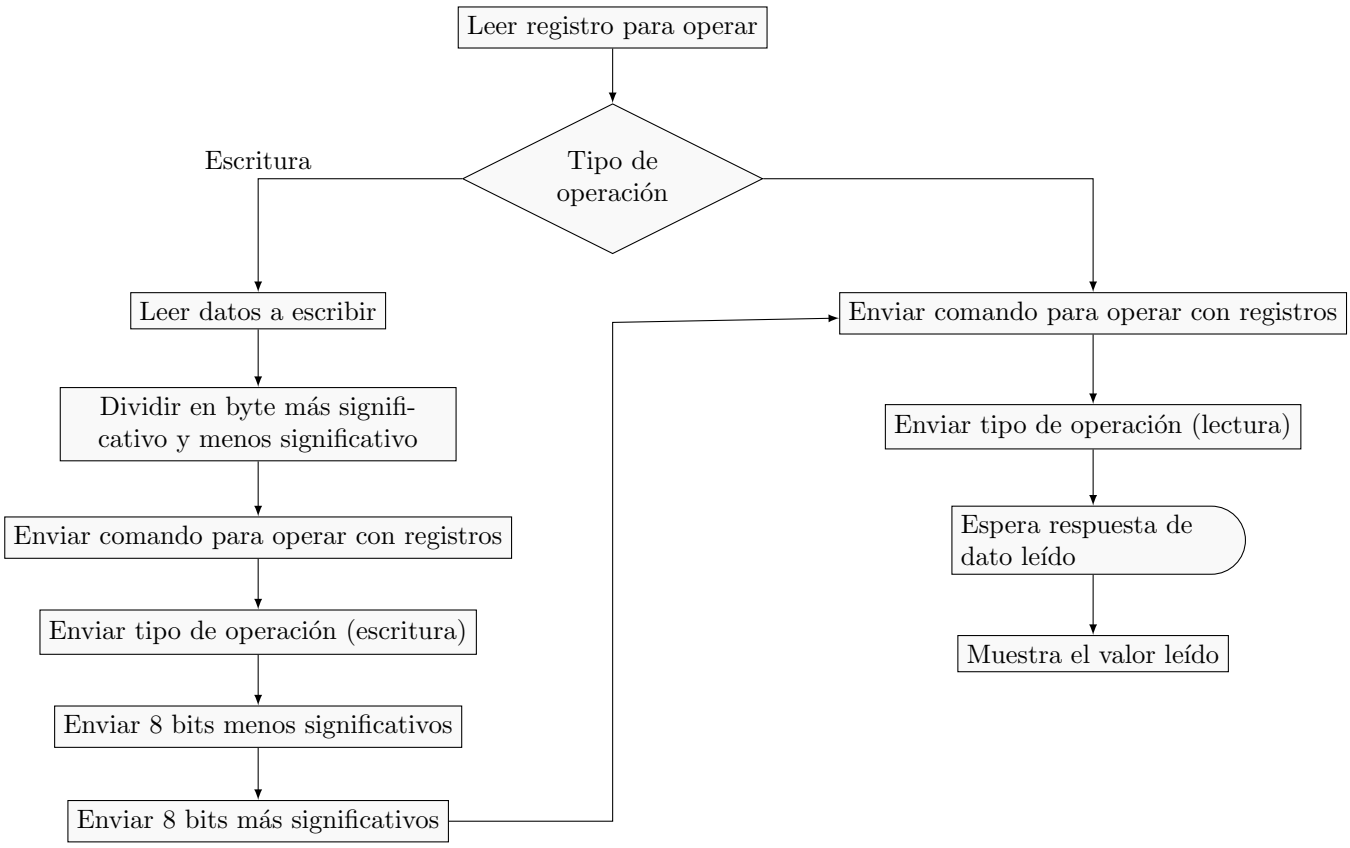


Figura D.19: Diagrama de flujo al ejecutar una lectura o escritura de registros

D.9.1.4. Carga de Imagenes

El comportamiento al presionar el botón “Iniciar envío” se presenta en la **Figura D.20**, ubicada en la hoja siguiente.

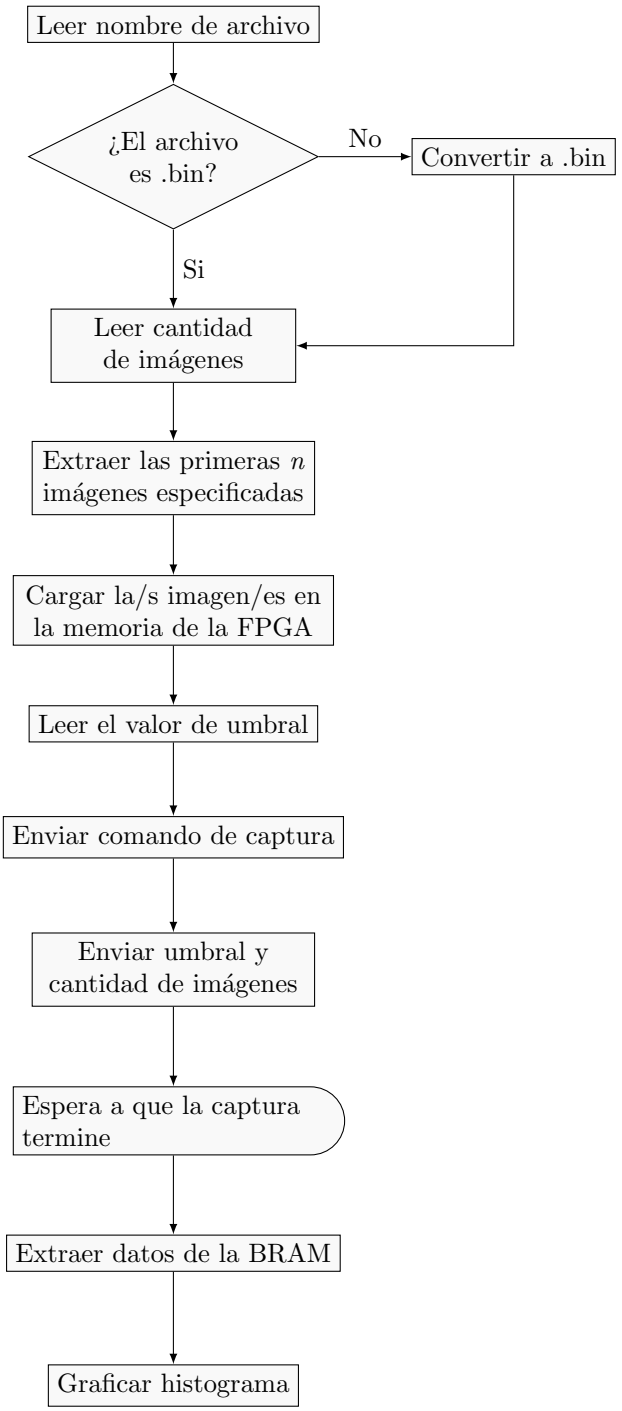


Figura D.20: Diagrama de flujo al presionar el botón “Iniciar envío”

Además, los botones “Iniciar envío” y “Ver histograma” estarán deshabilitados si la UART se encuentra desconectada. El primero se habilitará al conectarla, mientras que el segundo luego de realizar un envío de datos.

D.9.1.5. Histogramas

D.9.1.5.1 Creación de histogramas

Al presionar el botón “Recargar carpetas” se deben actualizar las carpetas en el listado:

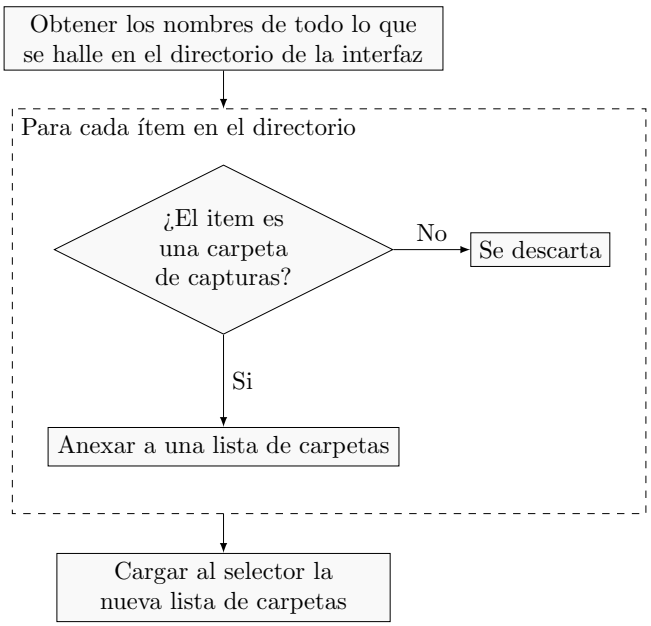


Figura D.21: Acción al presionar "Recargar carpetas"

Al presionar el botón "Crear histograma", se ejecuta:

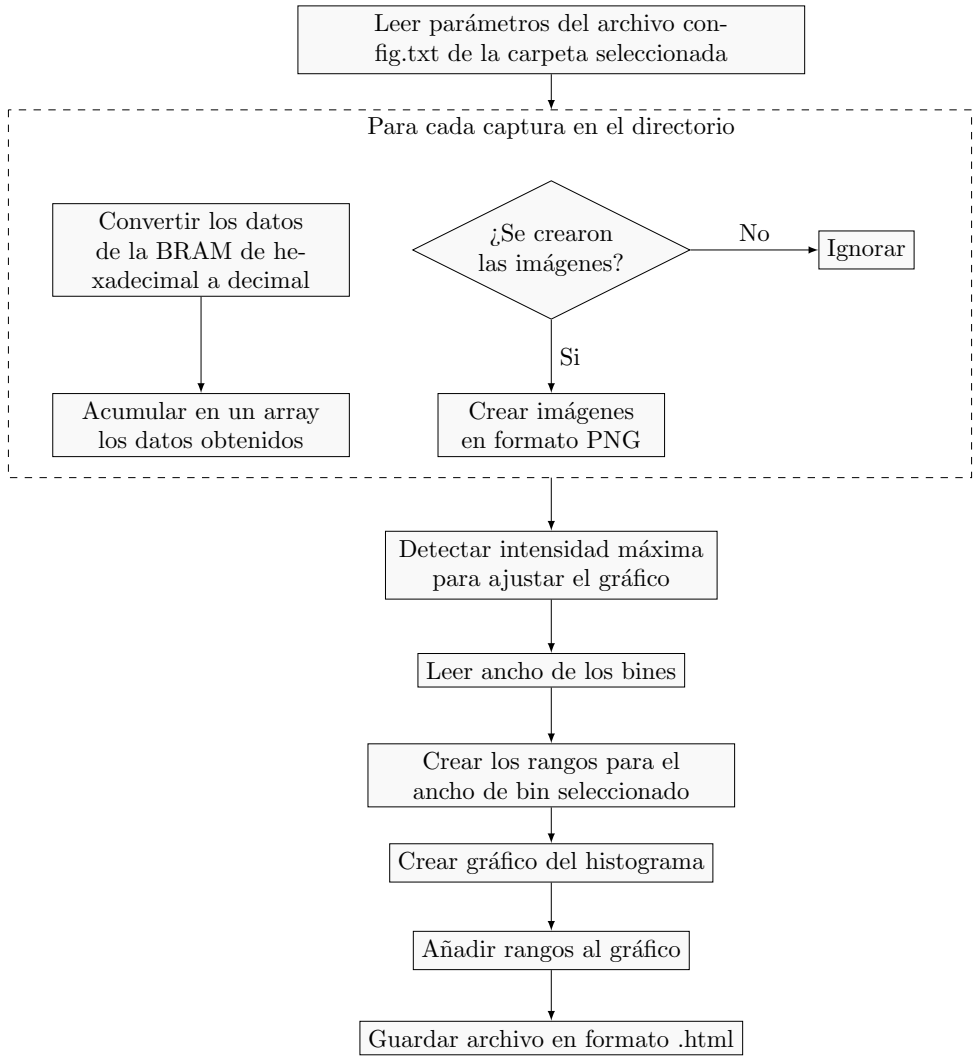


Figura D.22: Acción al presionar "Crear histograma"

D.9.1.5.2 Visualizar histogramas

El comportamiento del botón “Recargar carpetas” es idéntico al de la sección “Creación de histogramas”, pero se separa el selector para que sea menos confusa su utilización.

Al presionar recargar histogramas, se actualiza el listado de histogramas en la carpeta seleccionada, resulta similar a la función ejecutada al recargar carpetas:

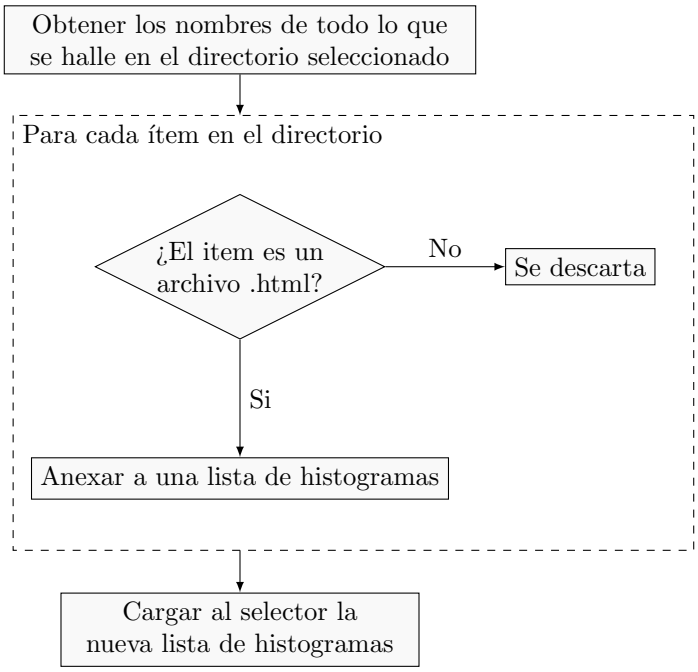


Figura D.23: Acción al presionar “Recargar histogramas”

Al presionar el botón “Ver histograma”, se abre el histograma en el navegador por default.

Apéndice E

Plan de pruebas

Universidad Nacional de Mar del Plata
Facultad de Ingeniería

Proyecto: SELFIE
Carrera: Ingeniería Electrónica

PLAN DE PRUEBAS

Alcance

El siguiente plan de pruebas abarca desde las pruebas individuales de los módulos, hasta las pruebas finales de aceptación del instrumento como un conjunto.

Ambientes de prueba

Los ambientes de prueba fueron tres:
- PC.
- PC + Sensor Aptina + FPGA.
- PC + Sensor Aptina + FPGA + Piedra de Americio 241.

Recursos, herramientas e instrumentos

- Placa de desarrollo Zedboard.
- PC.
- Sensor Aptina MTM9001.
- Piedra de Americio 241.
- Contenedor oscuro.
- Lente.

Políticas de trabajo

Las pruebas que se puedan realizar solo con el uso de la PC, se podrán realizar tanto en el Laboratorio de Sistemas Caóticos como en el hogar de los alumnos, comunicándose de por medio.
El resto de ellas serán desarrolladas en las instalaciones del Laboratorio, ya que requieren el uso de la FPGA y el sensor Aptina.
Por otro lado, para aquellas que requieran la manipulación de la piedra de Americio, solo uno de los alumnos manipulará la fuente de radiación y no tocará nada más, mientras que el otro se encargará del manejo de la PC. Cada vez que sea manipulada, el estudiante deberá lavarse las manos de manera exhaustiva sin tocar nada más, para ello el otro alumno abrirá las puertas y la canilla para el lavado. Una vez finalizado el uso de la piedra, deberá guardarse en el recipiente y caja correspondiente.

Estrategia de Comunicación

Los resultados de las pruebas se comunicarán en reuniones con los directores de tesis Maximiliano Antonelli y Claudio Marcelo González, y con los investigadores del Laboratorio de Bajas Temperaturas del Instituto Balseiro, Martín Pérez y José Lipovetzky.

Módulo	ID	Prueba	Tipo de Prueba	Procedimiento	Resultados esperados	Instrumental	Fecha de Prueba	Observaciones	Resultados obtenidos
i2cmaster	1	Escritura y lectura de registros	Unitaria	Leer registro 0x00. Escribir los valores de ganancia 0x01, 0x08 y 0x15 en el registro 0x35. Leer el registro de ganancia una vez que se escribió un valor. Poner en alto el bit 6 del registro 0x07 para que el sensor entregue la imagen de pruebas.	La lectura del registro 0x00 debe devolver el valor 0x8431. El valor escrito en el registro de ganancia debe coincidir con el valor leido. La imagen obtenida con el sensor en modo de pruebas debe estar formada por franjas negras y blancas verticales.	FPGA Vivado	15/02/20	-	Acorde a lo esperado
Interfaz de usuario	2	Comunicación con la FPGA y modos de funcionamiento	Unitaria	Conectarse por puerto UART. Escribir y leer registros. Capturar imagenes con un lente que serán transferidas por DMA a la computadora. Enviar imágenes por GPIO. Crear imágenes e histograma.	Conectarse y desconectarse correctamente de la UART. Realizar nuevamente los procedimientos de la prueba 1 y obtener las salidas indicadas. La imagen capturada coincide con el objeto colocado encima del sensor. La imagen enviada por GPIO y recibida es la misma. En la reconstrucción de imágenes no hay píxeles descorrelacionados, el histograma con bines unitarios coincide con el obtenido en Python y los datos son correctamente agrupados cuando se modifica el bineado.	FPGA Lente Vivado Vivado SDK	15/10/21	-	Acorde a lo esperado
Algoritmo de detección ideal	3	Verificación de detección de eventos	Unitaria	Generar imagenes arbitrarias de 5x5, 10x10 y 25x25. Luego, enviar imagenes de irradiaciones.	La cantidad de eventos y la intensidad de la imagen generada y lo reportado por el algoritmo coinciden.	IDE/Consola	10/07/20	-	Acorde a lo esperado
Algoritmo de detección real	4	Verificación de detección de eventos	Unitaria	Generar imagenes arbitrarias de 5x5, 10x10 y 25x25. Verificar manualmente si el resultado es correcto. Luego, enviar imagenes de irradiaciones.	La cantidad de eventos y la intensidad de la imagen generada y lo reportado por el algoritmo coinciden.	IDE/Consola	20/02/21	-	Acorde a lo esperado
Algoritmo de detección real	5	Verificación de píxeles muertos	Unitaria	Procesar las imágenes de irradiaciones y obtener los píxeles muertos tomando como píxeles muertos a aquellos píxeles que quedan encendidos comparando a la imagen anterior. Luego, procesar las imagenes de irradiaciones tomando como píxeles muertos a aquellos píxeles que quedan encendidos y no tienen otros píxeles encendidos a su alrededor.	La cantidad de píxeles muertos reportados por ambos métodos debe coincidir.	IDE/Consola	18/08/21	-	Acorde a lo esperado
Detector	6	Verificación de detección de eventos	Unitaria	Generar imagenes arbitrarias de 5x5, 10x10 y 25x25. Luego, enviar imagenes de irradiaciones, enviandolas en formato serie de 8 bits.	La cantidad de eventos y la intensidad de las imagenes reportadas por el bloque detector deben coincidir con lo reportado por el algoritmo de detección real.	FPGA Vivado	30/08/21	-	Problemas con el reset, las variables static no se reinician con el reset del bloque. Se solucionó con una instrucción pragma.
Detector	7	Tiempo de procesamiento	Unitaria	Enviar imágenes arbitrarias de 5x5, 10x10, 25x25 e imágenes de irradiaciones de 480x720 y 1280x1024. Anotar tiempos de descarte de píxeles por debajo del umbral, tiempos de procesamiento de eventos con varios píxeles y tiempo total. Probar leves modificaciones al HLS y volver a probar iterando.	Los tiempos de procesamiento deben bajar con cada iteración.	FPGA Vivado	24/5/2021 al 16/6/2021	Proceso iterativo.	La herramienta reportaba 20.44 W de consumo de potencia, resultado de un mal ruteo del clock. Una vez arreglado, los resultados fueron acordes a lo esperado.
Generador de histograma	8	Generación de histogramas	Unitaria	Generar arrays arbitrarios de tamaño e intensidad y enviarlos en formato serie de 8 bits.	El histograma enviado y el reportado por el bloque deben coincidir.	FPGA Vivado	31/08/21	-	Problemas con el reset, las variables static no se reinician con el reset del bloque. La solución hecha en el detector no es implementable porque ocupa 99% de LUTs, se utilizó un for loop para limpiar las posiciones.
Generador de histograma	9	Verificación de píxeles muertos	Unitaria	Enviar imágenes de irradiaciones.	En las primeras 2*umbral posiciones del histograma debe haber datos de los píxeles muertos.	FPGA Vivado	29/06/21		Acorde a lo esperado
Detector, generador de histograma y FIFOs	10	Detección de eventos y generación de histograma	Integral	Utilizar las mismas imagenes de las pruebas 4 y 5, enviar los datos de los pixeles en formato serie de 8 bits. Primero enviar de a una imagen, luego enviar de a dos. Enviar imágenes de: 5x5, 10x10, 25x25, 480x720 y 1280x1024	El histograma generado debe coincidir con lo calculado, para las pruebas de imágenes pequeñas, y con lo obtenido mediante el procesamiento en Python, para las imágenes más grandes.	FPGA Vivado	30/06/21	-	Acorde a lo esperado
Self Start (SS)	11	Transición de estados y generación de señales	Unitaria	Enviar las señales y pulsos correspondientes para los cambios de estado.	Los cambios de estado se producen cuando corresponde y las señales de salida tienen los valores correctos.	FPGA Vivado	11/08/21	-	Acorde a lo esperado
Sincronizador de datos (SYNC)	12	Sincronización de datos de entrada	Unitaria	Enviar señales de entrada.	2 ciclos de clocks después debe aparecer el cambio en la salida	FPGA Vivado	09/09/21	-	Acorde a lo esperado
Capture Input Controller (CIC)	13	Transición de estados y generación de señales	Unitaria	Enviar las señales y pulsos correspondientes para los cambios de estado. Verificar que estos cambios se producen y que las señales de salida tienen los valores correctos	Los cambios de estado del bloque se producen cuando corresponde y las señales de salida tienen los valores correctos.	FPGA Vivado	26/08/21	-	Acorde a lo esperado
Pix Pulse Valid Generator (PPVG)	14	Generación del pulso para escribir la FIFO	Unitaria	Ingresar con un clock a 100 MHz y una señal a 10 MHz.	Se genera un pulso de 10ns (1 clock) cada vez que llega un pulso de frecuencia 10 MHz	FPGA Vivado	25/08/21	-	Acorde a lo esperado
Auto Mode Logic Controller (AMLC)	15	Transición de estados y generación de señales	Unitaria	Enviar las señales y pulsos correspondientes para los cambios de estado.	Los cambios de estado del bloque se producen cuando corresponde y las señales de salida tienen los valores correctos.	FPGA Vivado	11/08/21	-	Acorde a lo esperado
Auto Mode Controller (AMC): AMLC, PPVG, CIC, SYNC	16	Transición de estados y generación de señales	Integral	Enviar las señales y pulsos correspondientes para los cambios de estado.	Los cambios de estado del bloque se producen cuando corresponde y las señales de salida tienen los valores correctos.	FPGA Vivado	15/10/21	-	Acorde a lo esperado
tvalid_fsm	17	Generación de las señales tvalid, tkeep, tlast y tdata para el módulo data_to_axi	Unitaria	Enviar 1310720 pulsos a 10 MHz con una frecuencia de clock de 100 MHz.	Verificar que tlast se genere en el último pulso.	FPGA Vivado	02/09/21	-	Acorde a lo esperado
Debug Mode Controller (DMC): tvalid_fsm, tvalid_counter, data_to_axi e i2cmaster	18	Lectura de datos y generación de señales	Unitaria	Enviar las señales y pulsos correspondientes para los cambios de estado.	Verificar que los cambios de estado se producen, que las señales de salida tienen los valores correctos y que se leen los datos enviados desde el Vivado SDK.	FPGA Vivado Vivado SDK	23/09/21	-	Acorde a lo esperado
SELFIE	19	Funcionamiento del detector, generador de histograma	Integral	Enviar imágenes desde la interfaz para verificar el correcto procesamiento de imágenes.	El histograma de la imagen enviada y el histograma reportado por el instrumento deben coincidir.	FPGA Vivado Vivado SDK Interfaz	15/10/21	-	Acorde a lo esperado
SELFIE	20	Captura y procesamiento de eventos	Homologación	1- Capturar 60 imágenes antes de irradiar, calcular el umbral en base a la gaussiana del ruido. 2- Capturar 60 imágenes con la piedra de americio sobre el sensor. 3- Capturar 60 imágenes después de irradiar para ver el estado del sensor. Realizar esto para ganancia x15, x8 y x1 en ese orden.	Observar que se forma una gaussiana y, a medida que la ganancia disminuye, se corre hacia la izquierda. Observar que la cantidad de píxeles muertos después de irradiar sea mayor a antes de irradiar.	FPGA Vivado Vivado SDK Interfaz Piedra de americio Contenedor oscuro	06/11/21	Para la generación de eventos se utilizó una piedra de Am241	Acorde a lo esperado
SELFIE	21	Captura y procesamiento de eventos para partículas gamma	Homologación	1- Cubrir el sensor con una servilleta y colocar la piedra de americio sobre ella. 2- Configurar la ganancia en x15. 3- Dejar capturando al instrumento por 24 horas y construir histograma.	Se debe obtener un histograma de intensidades con una concentración de eventos para intensidades bajas.	FPGA Vivado Vivado SDK Interfaz Piedra de americio Contenedor oscuro	13/11/21	Para la generación de eventos se utilizó una piedra de Am241	No se pudo finalizar. Los archivos del SDK de Xilinx se corrompen a las 3 horas aproximadamente.