



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería Electrónica y Eléctrica

Escuela Profesional de Ingeniería Electrónica

Diseño de un sistema de registro de imágenes de 5MP de bajo consumo para el monitoreo e inventario de animales grandes en zonas rurales del país

TESIS

Para optar el Título Profesional de Ingeniero Electrónico

AUTOR

Michel Ányelo ZARZOSA ROJAS

ASESOR

Flavio Nireo CARRILLO GOMERO

Lima, Perú

2023



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Zarzosa, M. (2023). *Diseño de un sistema de registro de imágenes de 5MP de bajo consumo para el monitoreo e inventario de animales grandes en zonas rurales del país*. [Tesis de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería Electrónica y Eléctrica, Escuela Profesional de Ingeniería Electrónica]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Michel Ányelo Zarzosa Rojas
Tipo de documento de identidad	DNI
Número de documento de identidad	45477442
URL de ORCID	https://orcid.org/0000-0002-4203-6947
Datos de asesor	
Nombres y apellidos	Flavio Nireo Carrillo Gomero
Tipo de documento de identidad	DNI
Número de documento de identidad	08624069
URL de ORCID	https://orcid.org/0000-0003-3043-2384
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Rafael Bustamante Alvarez
Tipo de documento	DNI
Número de documento de identidad	09377230
Miembro del jurado 1	
Nombres y apellidos	Luis Ernesto Cruzado Montañez
Tipo de documento	DNI
Número de documento de identidad	32920395
Miembro del jurado 2	
Nombres y apellidos	Rubén Gino Martínez Varela
Tipo de documento	DNI
Número de documento de identidad	09270432
Datos de investigación	
Línea de investigación	C.0.3.3. Desarrollo de modelos y aplicación de las tecnologías de la información y comunicaciones

Grupo de investigación	No aplica
Agencia de financiamiento	No aplica
Ubicación geográfica de la investigación	Edificio: Pontificia Universidad Católica del Perú País: Perú Departamento: Lima Provincia: Lima Distrito: San Miguel Latitud: -12.0475986 Longitud: -77.0325514
Año o rango de años en que se realizó la investigación	Agosto 2018 – Noviembre 2022
URL de disciplinas OCDE	Ingeniería eléctrica, Ingeniería electrónica https://purl.org/pe-repo/ocde/ford#2.02.01



UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
(Universidad del Perú, DECANA DE AMÉRICA)
FACULTAD DE INGENIERÍA ELECTRÓNICA Y ELÉCTRICA

ACTA DE SUSTENTACIÓN N° 003-EPIE-FIEE/2023

**TESIS N° 003-FIEE/2023 PARA OPTAR EL TÍTULO PROFESIONAL
DE INGENIERO ELECTRÓNICO**

Los suscritos Miembros de Jurado, nombrados por la Dirección de la Escuela Profesional de Ingeniería Electrónica, reunidos en la fecha bajo la Presidencia del **Dr. Rafael Bustamante Álvarez** e integrado por: **Mg. Rubén Gino Martínez Varela**, **Mg. Luis Ernesto Cruzado Montañez** y el **Ing. Flavio Carrillo Gomero** (Miembro -Asesor)

Después de escuchar la Sustentación de Tesis del Bachiller Michel Ányelo Zarzosa Rojas (09190145), para optar el Título Profesional de Ingeniero Electrónico por la modalidad de Titulación Ordinaria, quien expuso su tesis: **“DISEÑO DE UN SISTEMA DE REGISTRO DE IMÁGENES DE 5MP DE BAJO CONSUMO PARA EL MONITOREO E INVENTARIO DE ANIMALES GRANDES EN ZONAS RURALES DEL PAIS”** para la EPI Electrónica de la Facultad de Ingeniería Electrónica y Eléctrica de la Universidad Nacional Mayor de San Marcos.

Se acordó **APROBARLO** por **UNANIMIDAD**

Con la Nota de **_DIECISIETE_ (17)**

Ciudad Universitaria, 15 de julio de 2023

Mg. Rubén Gino Martínez Varela
Jurado Revisor

Mg. Luis Ernesto Cruzado Montañez
Jurado Revisor

Dr. Rafael Bustamante Álvarez
Jurado Revisor (Presidente)

Ing. Flavio Carrillo Gomero
Asesor



Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Vicerrectorado de Investigación y Posgrado



CERTIFICADO DE SIMILITUD

Yo Flavio Carrillo Gomero en mi condición de asesor acreditado con el Acta de Sustentación N°003-EPIE-FIEE/2023 de trabajo de tesis N°003-EPIE-FIEE/2023 del trabajo de tesis profesional cuyo título es: "DISEÑO DE UN SISTEMA DE REGISTRO DE IMÁGENES DE 5MP DE BAJO CONSUMO PARA EL MONITOREO E INVENTARIO DE ANIMALES GRANDES EN ZONAS RURALES DEL PAÍS" presentado por el bachiller Michel Anyelo Zarzosa Rojas para optar al título profesional de Ingeniero Electrónico. CERTIFICO que se ha cumplido con lo establecido en la Directiva de Originalidad y de Similitud de Trabajos Académicos, de Investigación y Producción Intelectual. Según la revisión, análisis y evaluación mediante el software de similitud textual, el documento evaluado cuenta con el porcentaje de 5% de similitud, nivel **PERMITIDO** para continuar con los trámites correspondientes y para su **publicación en el repositorio institucional**. Se emite el presente certificado en cumplimiento de lo establecido en las normas vigentes, como uno de los requisitos para la obtención del grado correspondiente.

Firma del Asesor

DNI: 08624069

Nombres y apellidos del asesor:

Ing. Flavio Carrillo Gomero

*Quisiera dedicar mi tesis a mi familia
y en especial a mi hija Bianca quien es mi
mayor motivo e inspiración.*

Michel Anyelo Zarzosa Rojas

AGRADECIMIENTOS

En primer lugar, quisiera agradecer a mi familia que siempre estuvo presente apoyándome en mi vida universitaria en especial a mi hija Bianca a quien atribuyo mi principal fuente de motivación.

Un agradecimiento especial para el Grupo de Telecomunicaciones Rurales quienes me acogieron en mi primera etapa laboral y donde aprendí día a día en el desarrollo de diversos proyectos de investigación.

También un especial agradecimiento a mi asesor el Ing. Flavio Carrillo, por su constante asesoría en el desarrollo del presente trabajo.

Por último, mi agradecimiento especial a mi Alma Mater, La Universidad Nacional Mayor de San Marcos donde me forme como profesional.

RESUMEN

Este trabajo de tesis consiste en el diseño e implementación de un módulo electrónico desarrollado para capturar imágenes de animales en su hábitat natural cuando son detectados por un sensor de movimiento que activa una cámara y procede a la captura de la imagen, para luego ser almacenada en una memoria SD. El módulo utiliza 3 circuitos integrados para la captura y almacenamiento de las imágenes en la memoria SD constituida por el dsPIC33 de la familia de Microchip, el sensor de luz OV5642 para capturar imágenes de hasta 5MP y una memoria FIFO de 512 KB de almacenamiento. El microcontrolador dsPIC33 es el dispositivo principal que se encarga de configurar la resolución, balance de blancos, autoenfoco, entre otras características de la imagen que se desea capturar e indicarle a la memoria FIFO el momento en el cual debe ejecutar el almacenamiento temporal de la imagen para ser leído y enviado a la memoria SD. Como resultado, se comprobó que en caso la capacidad de la memoria FIFO sea superada, el microcontrolador dsPIC configura un tiempo de espera pequeño hasta leer y enviar toda la información de la imagen a la memoria SD. Por otro lado, para evitar espantar al animal detectado fue conveniente el uso automatizado de un flash de luz amarilla para la iluminación nocturna y diferenciar las características propias del animal. Finalmente, se ha comprobado que este módulo ahorra consumo de energía eléctrica y ha permitido la mayor cantidad de captura de imágenes.

Palabras clave: Microchip, Microcontrolador dsPIC33, Sensor CMOS OV5642, Memoria FIFO, Sensor de movimiento.

ABSTRACT

This thesis work consists of the design and implementation of an electronic module developed to capture images of animals in their natural habitat when they are detected by a motion sensor that activates a camera and proceeds to capture the image, to later be stored in an SD memory. The module uses 3 integrated circuits to capture and store images in the SD memory consisting of the dsPIC33 from the Microchip family, the OV5642 light sensor to capture images up to 5MP and a FIFO memory of 512 KB of storage. The dsPIC33 microcontroller is the main device that is in charge of configuring the resolution, white balance, autofocus, among other characteristics of the image to be captured and indicating to the FIFO memory the moment in which it must execute the temporary storage of the image to be read and sent to the SD memory. As a result, it was found that in case the capacity of the FIFO memory is exceeded, the dsPIC microcontroller configures a short waiting time until reading and sending all the image information to the SD memory. On the other hand, to avoid frightening the detected animal, it was convenient to use an automated flash of yellow light for night lighting and to differentiate the animal's own characteristics. Finally, it has been proven that this module saves electricity consumption and has allowed the greatest amount of image capture.

Keywords: Microchip, dsPIC33 microcontroller, CMOS OV5642 sensor, FIFO memory, Sensor motion.

Tabla de contenido

CAPITULO 1. INTRODUCCIÓN.....	1
1.1 Planteamiento del problema	2
1.1.1 Problemas generales y específicos.....	3
1.2 Justificación	4
1.2.1 Objetivos	5
CAPITULO 2. MARCO TEÓRICO	6
2.1 Puesta en campo de cámara.....	8
2.2 Altura de la cámara.....	9
2.3 Angulo de la cámara.....	10
2.4 Distancia al sendero o trocha	11
CAPITULO 3. HIPOTESIS Y VARIABLES	14
3.1 Hipótesis general	14
3.2 Hipótesis específica	14
3.3 Variables.....	14
3.4 Operacionalización de las variables	15
3.5 Matriz de consistencia	15
CAPITULO 4. MATERIALES Y METODOS.....	16
4.1 Área de estudio.....	16
CAPITULO 5. COMPONENTES DEL MÓDULO CÁMARA.....	17
5.1 Procesador dsPIC33.....	18
5.2 Sensor OV5642	19
5.3 Memoria FIFO AL422B.....	19
5.4 Memoria SD.....	20
5.5 Sensor de Movimiento AMN41121 (PIR).....	20
5.6 Real Time Clock (RTC).....	20
5.7 Flash con luz amarilla	21
5.8 Súper capacitor.....	21

5.9 Fotorresistencia	21
5.10 Sensor de temperatura y humedad	22
5.11 Mosfet de potencia	22
CAPITULO 6. DISEÑO DEL HARDWARE	23
6.1 Diseño y control del prototipo del módulo cámara.....	25
6.2 Comportamiento de la memoria FIFO durante la lectura y escritura de datos.....	27
6.2.1 Comportamiento durante la escritura.....	27
6.2.2 Comportamiento durante la lectura.....	29
6.3 Acceso directo a memoria (Direct Memory Access - DMA)	30
6.4 Control de la información de la imagen.....	31
CAPITULO 7. CONTROL DE LA CAMARA.....	33
CAPITULO 8. CONFIGURACIÓN DE PERIFERICOS DEL dsPIC PARA UN ÓPTIMO COMPORTAMIENTO COMO REGISTRADOR DE IMÁGENES	36
8.1 Funciones necesarias para el correcto inicio de la cámara	36
• Configuración del oscilador interno.....	36
8.1.1 Temporizadores. Timers de 16 y 32 bits	37
8.1.2 Configurar clock de referencia para alimentar al sensor de la Cámara con una frecuencia de 7.5MHz	40
8.1.3 Configuración bus I2C.....	41
8.1.4 Configurar bus UART	42
8.1.5 Configurar bus SPI	43
8.1.6 Uso del acceso directo a memoria enlazado al SPI para el rápido envío de los de la memoria FIFO hacia la micro SD.	45
8.1.7 Inicialización de la memoria SD y saber si se encuentra presente o no. 47	
8.1.8 Definición de variables estáticas y variables necesarias para las interrupciones.....	48
8.2 Configuración de periféricos necesarios para el control de la cámara	49
• Función Void InicializaIO	49
• Camara_GetVersion	50
• Camara_Set_VGAprev_YUVmode	51
• Función Camara_Set_QSXGAapt_JPEGmode	51
• Función Camara_SetSignalsPolarity	53
• Camara_SetProcessorClock.....	53

8.3 Creación de un archivo	53
• Cámara_crearArchivo	53
8.4 Configuración de rtc para asignar la hora a los archivos	54
8.4.1 DS3231_Init	54
8.4.2 DS3231_DateHour	55
8.4.3 DS3231_EnableInt	55
8.4.4 DS3231_ClearInt.....	56
8.5 Asignar la hora y fecha a la que fue tomada la imagen en el archivo .jpg creado.....	56
8.6 Configuración para las interrupciones	57
8.6.1 _INT1Interrupt	57
8.6.2 _INT2Interrupt	59
8.6.3 _T1Interrupt.....	61
8.6.4 _T5Interrupt.....	61
8.6.5 PIR_IntConfig.....	61
8.7 Ahorro de energía e inicialización y configuración.....	62
8.8 Funciones camara_tomarimagen y camara_guardarimagen:.....	66
8.8.1 Camara_TomarImagen:	66
8.8.2 Camara_GuardarImagen	69
8.9 Control de medición de temperatura, humedad.....	75
8.9.1 Función HDC1000_ReadHum:.....	75
8.9.2 Función HDC1000_ReadTemp:.....	76
8.9.3 Función Temp_Init:.....	77
8.10 Control de memoria SD	78
8.11 Comportamiento como cámara trampa	78
CAPITULO 9. RESULTADOS	81
CONCLUSIONES	88
RECOMENDACIONES.....	89
BIBLIOGRAFÍA	90
ANEXOS	93

LISTA DE FIGURAS

Fig. 1.- En esta En esta fotografía se puede observar a una persona gateando delante de la cámara para corroborar una correcta captura del posible animal.	11
Fig. 2.- Captación de la presencia de un felino en donde se observa la orientación de la cámara hacia la trocha y la nitidez de la imagen del animal captado	12
Fig. 3.- Diagrama de conexiones entre la memoria FIFO, sensor OV5642 y el dsPIC.	23
Fig. 4.- Pulsos VSYNC y HREF captados por el osciloscopio.	27
Fig. 5.- Operación de escritura de datos en los registros (AVERLOGIC, 2016).	28
Fig. 6.- Operación de lectura de datos de los registros (AVERLOGIC, 2016).....	29
Fig. 7.- Diagrama de bloques del controlador DMA en el dsPIC33 (Microchip, 2011).....	31
Fig. 8.- Diagrama de conexiones del sistema diseñado.	35
Fig. 9.- Diagrama de flujos de la interrupción generada por el VSYNC	58
Fig. 10.- Diagrama de flujos de la interrupción generada por el ORDY	60
Fig. 11.- Diagrama de flujos de la interrupción generada por el PIR	62
Fig. 12.- Diagrama de flujos del comportamiento del módulo para un bajo consumo energético.....	65
Fig. 13.- Diagrama de flujo de la función Camara_TomarImagen()	67
Fig. 14.- Diagrama de flujos de la lectura de la memoria FIFO y la creación del archivo	71
Fig. 15.- Diagrama de flujos del comportamiento como cámara Trampa.	80
Fig. 16.- Imagen de 5mp capturada por el sensor OV5642	81

Fig. 17.- A la izquierda: Imagen capturada con el sensor CMOS de 5mp bajo sombra, a la derecha: imagen capturada con el sensor de 0.2 mp usada en el proyecto TAPIRNET.	82
Fig. 18.- A la izquierda: Imagen capturada con el sensor CMOS de 5mp bajo sombra, a la derecha: imagen capturada con el sensor de 0.2 mp usada en el proyecto TAPIRNET.	83
Fig. 19.- Imágenes capturadas con exposición directa del sol. A la izquierda: Sensor 0.2mp, a la derecha: Sensor 5mp.....	83
Fig. 20.- Imágenes capturadas con exposición directa del sol. A la izquierda: Sensor de 5mp, a la derecha: Sensor de 0.2mp.....	84
Fig. 21.- Imagen bajo sombra tomada con sensor 5mp.	84
Fig. 22.- Imagen bajo sombra tomada con sensor 5mp	85
Fig. 23.- Imagen bajo sombra tomada con sensor 5mp	85
Fig. 24.- Imagen bajo sombra tomada con sensor 5mp	86

LISTA DE TABLAS

Tabla I Diferencias entre diferentes tipos de cámaras trampa.	9
Tabla II Tiempos de escritura de las imágenes en la memoria SD.	86

LISTA DE ACRÓNIMOS

C

CMOS
metal oxide semiconductor · 4

D

DMA
direct memory access · 22
DPSRAM
Dual port SRAM · 33
DSC
Digital signal controller · 33

F

FIFO
first-in, first-out · 4
FPGA
field-programmable gate array · 21

I

I2C
Inter-Integrated Circuit · 21

K

KB
kilo bytes · 4

L

LED
light-emitting diode · 19

M

MB
Mega bytes · 23
MBaud
mega baudios · 23
MBps
Mega bits por segundo · 23
MHz
Megahercios · 23
MIPS
millones de instrucciones por segundo · 22
MP
megapíxeles · 4

P

PIC
peripheral interface controller · 23
PIP
Picture-in-picture · 24
PIR
pyroelectric Infrared Radial · 12
PLL
Phase-Locked Loop · 22

R

RTC
Real time clock · 25
RTSP
Run-Time Self-Programming · 33

S

SD
secure digital · 4
SPI
Serial Peripheral Interface · 20
SRAM
Static random access memory · 33

U

Universal Asynchronous Receiver-Transmitter ·

20

UART

CAPITULO 1. INTRODUCCIÓN

La preservación de animales y plantas se ha convertido en un tema importante, especialmente en la selva amazónica del Perú. Esta gran área geográfica ha sido recientemente afectada por actividades económicas legales e ilegales, por lo que se requieren medidas especiales de preservación para abordar la sostenibilidad de este frágil medio ambiente. El proyecto TAPIRnet (Camacho, Baquerizo, Palomino, & Gagliardi, 2016) fue una iniciativa desarrollada para el Instituto de Investigaciones de la Amazonia Peruana – IIAP, para contribuir con la preservación de los bosques mediante el uso de herramientas TIC. En esta primera experiencia, las redes inalámbricas de cámaras trampa se desplegaron en la Reserva Nacional Allpahuayo Mishana - Iquitos. El objetivo de aquel proyecto fue de apoyar a los biólogos en la creación de un inventario de mamíferos grandes. Las cámaras trampa de esta red, registraron imágenes de 0.3 MP y transmitieron datos a uno que actuaba como un nodo de sumidero central. A partir de esta experiencia, se obtuvieron ideas de mejorar dicho sistema.

Cuanto más relevante era actualizar las cámaras para una mejor resolución, más crítico era identificar con precisión los mamíferos en observación. Como consecuencia, un segundo proyecto llamado TAPIRchip (Zarzosa, Palomino, Chávez, & Camacho, 2016) fue ejecutado, cuyo objetivo principal fue desarrollar un módulo de cámara más sofisticada y con mejor resolución, capaz de registrar datos de parámetros Ambientales y enviarlos inalámbricamente a nodo central.

Para la red de sensores inalámbricos, el sensor visual ha adquirido especial importancia en aplicaciones de monitoreo ambiental, a fin de que los investigadores dispongan de información visual importante. En este contexto, tener una imagen de la escena no es suficiente, si no la de disponer de una imagen con calidad que muestre detalles, lo que implica mayor resolución en comparación con los obtenidos en los proyectos anteriores, cuantos más

píxeles tenga la imagen, más detalles se podrá visualizar. Para esta utilidad se espera por lo menos 2 MP de resolución y captura de 2 imágenes en 1 segundo o menos. En el presente trabajo de tesis se propone el desarrollo del hardware y software de la tarjeta de control de una cámara de 5MP que emplea un sensor CMOS fabricado por OmniVision, este es controlado por un módulo constituido por un microprocesador dsPIC, una memoria FIFO para almacenar los datos desde el sensor CMOS, una tarjeta SD para almacenar permanentemente las imágenes, logs generados y un sensor de movimiento (PIR).

1.1 Planteamiento del problema

De acuerdo con la Unión Internacional para la Conservación de la Naturaleza (UICN), las áreas naturales protegidas son espacios continentales o marines del territorio nacional reconocidos, establecidos y cautelados legalmente por el estado. En el caso del Perú están adscritas al Ministerio del Ambiente.

El Perú tiene una superficie de 1.285.215km² y con el objetivo de conservar la flora, fauna y los paisajes se instalaron varios mecanismos para proteger la biodiversidad, estas áreas protegidas se dividen en varias categorías Parques Nacionales, Reservas, Reservas Paisajísticas y Santuarios Históricos. En tal sentido la preservación de animales y plantas ha llegado a tener una relevancia especial en la selva de la amazonia peruana. Esta extensa área geográfica ha sido recientemente afectada por la caza ilegal y la tala de árboles, como es el caso Reserva Nacional Allpahuayo Mishana – Iquitos.

Debido a los cambios en la composición de la vegetación es que se pueden tener efectos sobre la abundancia relativa y la distribución de las especies de mamíferos medianos y grandes (Sáenz-Méndez et al. 2005; Lozano-Rodríguez 2010; Monroy-Vilchis et al. 2011; Aranda et al. 2012; Mesa-Zavala et al. 2012), siendo de gran importancia conocer el estado en el que se encuentran estas poblaciones. Por lo tanto, los estudios de abundancia

relativa son de gran utilidad para la detección de cambios en la dinámica de las poblaciones de la fauna silvestre y la toma de acciones necesarias de ser el caso para el manejo y conservación de una determinada población.

Actualmente un método usado por los biólogos para el monitoreo de animales salvajes en una determinada zona es utilizando la técnica de rastreo mediante huellas, observación directa, entre otros. Cabe destacar que la observación hecha por los biólogos también abarca las huellas de los cazadores, así como la vista de trochas temporales que estos realizan para abrirse camino dentro del bosque.

Como se puede observar este método de monitoreo por rastreo es muy tedioso y requiere que los biólogos realicen largas caminatas, siendo muchas veces la determinación de los rastros difícil debido a la similitud de las huellas de algunas especies. Debido a esta necesidad es necesario tomar medidas preventivas para la sostenibilidad de este medio ambiente haciendo uso de cámaras para reconfirmar o descartar la presencia de determinadas especies en el área de estudio.

1.1.1 Problemas generales y específicos

1.1.1.1 Problema general

¿En qué medida el diseño de un sistema de registro de imágenes de 5MP de bajo consumo mejorará el monitoreo e inventario de los animales grandes en zonas rurales del país?

1.1.1.2 Problemas específicos

- ¿De qué manera el diseño de dispositivos electrónicos para un sistema de registro de imágenes de 5MP permiten el monitoreo e inventario de los animales grandes en zonas rurales del país?
- ¿De qué manera el diseño de un sistema de registro de imágenes de 5MP permite la cuantificación de animales grandes captados en el monitoreo e inventario de los animales grandes en zonas rurales del país?

- ¿Cómo el diseño de un sistema de registro de imágenes de 5MP mejora la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país?

1.2 Justificación

La rápida pérdida de la biodiversidad y recursos naturales del Perú justifican la urgencia de apoyar nuevos métodos que ayuden a identificar, describir y caracterizar a la biodiversidad en la brevedad posible para que se puedan tomar las debidas medidas de conservación y mitigación. Enfocándose en la importancia del uso de técnicas modernas, siendo unas de ellas el inventario mediante imágenes haciendo uso de cámaras.

El uso de estas cámaras para el inventariado y monitoreo de animales en un determinado ambiente son efectuadas por las cámaras trampa, cámaras de alta resolución (mayores a 2 megapíxeles) que cuentan con un sensor de movimiento para la detección de la presencia del animal, que permite la captura de imágenes del animal en su hábitat natural, el almacenamiento en una memoria externa SD y la disponibilidad de un flash, usualmente infrarrojo, para la toma de imágenes nocturnas para de esta manera evitar que el animal huya y les genere temor la zona donde se encuentra la cámara. Estas cámaras trampa son instaladas en lugares predeterminados por un periodo de días retornando para el cambio de baterías o ser desplazadas a otro ambiente diferente donde se pretende realizar el estudio e inventario de animales.

El área de seguridad y la vigilancia también presenta un crecimiento potencial para el mercado de este tipo de sensores, soportando definiciones de hasta 4K y ofreciendo diversas características para una mejor visualización en ambientes de baja iluminación.

Por este motivo mediante este trabajo se contribuirá en la implementación de un producto de software de control para controlar un sensor CMOS de 5

megapíxeles, así como establecer un tiempo corto entre el disparo y el almacenamiento rápido en la memoria SD que dispone de la capacidad suficiente para el almacenamiento de múltiples imágenes. Así mismo, se incorporará un sistema de ahorro de energía eléctrica que permita una autonomía de operación de meses con respecto a las cámaras trampas convencionales.

1.2.1 Objetivos

1.2.1.1 Objetivo general.

Diseñar un sistema de registro de imágenes de 5MP de bajo consumo para el monitoreo e inventario de animales grandes en zonas rurales del país.

1.2.1.2 Objetivos específicos.

- Diseñar dispositivos electrónicos para un sistema de registro de imágenes de 5MP que permitan el monitoreo e inventario de los animales grandes en zonas rurales del país.
- Diseñar un sistema de registro de imágenes de 5MP que permita la cuantificación de mamíferos captados en el monitoreo e inventario de los animales grandes en zonas rurales del país.
- Diseñar un sistema de registro de imágenes de 5MP que permita la mejora de la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país.

CAPITULO 2. MARCO TEÓRICO

El monitoreo de la biodiversidad se puede realizar de diferentes maneras, dependiendo de los objetivos de los estudios. Sin embargo, existen diferentes enfoques, así como diferentes métodos para cada uno de estos enfoques. En consecuencia, para el estudio de las especies y la estimación de los índices de abundancia, entre otros índices, se utilizan diversos métodos para conocer la situación actual de cómo se encuentran las poblaciones de animales en un momento dado y a su vez poder realizar una evaluación del hábitat y los cambios producidos en ella (Gaines, Harrod, & Lehmkuhl, 1999).

Los métodos indirectos son utilizados con frecuencia para el monitoreo de mamíferos grandes y medianos a partir de indicios como son las huellas, el excremento, las madrigueras, entre otros; esto debido a la gran dificultad para poder realizar observaciones directas o capturas de estos animales (Lyra-Jorge, Ciocheti, Pivello, & Meirelles, 2008).

Existen diferentes métodos para monitorear la biodiversidad los cuales se seleccionan de acuerdo a diferentes criterios. Dos de los métodos más utilizados para el monitoreo de mamíferos grandes y medianos son el registro por huella y el uso de cámaras trampa, siendo ambos métodos no invasivos que permiten identificar que especies se encuentran en un área determinada y monitorear la abundancia de las especies, además las cámaras trampa se utilizan para el estudio de patrones de actividad de los animales (Cutler & Swann, 1999) (McCullough, Pei, & Wang, 2000). Esta última metodología presenta varias ventajas, entre las cuales se destacan, que una vez capturada la imagen del individuo este puede ser identificado fácilmente hasta el nivel de especie de esta manera, el equipo no tiene que ser monitoreado de forma permanente (Lyra-Jorge, Ciocheti, Pivello, & Meirelles, 2008).

La técnica de cámaras trampa es ampliamente utilizada para monitorear especies de hábitat nocturna y evasivos que generalmente se encuentran en bajas densidades. Estos equipos son efectivos para detectar especies de mamíferos terrestres de más de 1Kg; siendo eficientes para coleccionar datos que brinden información adicional de la distribución y uso del hábitat de las especies, estructuras poblacionales y comportamiento. Esta se aplica también para realizar inventario de especies, estimación de abundancia y evaluación de esfuerzos de conservación, adicionalmente los datos obtenidos a partir de cámaras trampa pueden ser utilizados para fijar y seguir metas de conservación de especies (Rowcliffe, Field, Turvey, & Carbone, 2008) (Treves, Mwima, Plumptre, & Isoke, 2010).

Existen dos tipos de cámaras trampa clasificadas como: activas y pasivas, las cuales se diferencian en el mecanismo de disparo. Las cámaras activas capturan la imagen de un animal u objeto cuando estos atraviesan por un rayo infrarrojo, mientras que las cámaras pasivas inician la secuencia de disparo cuando un objeto con una temperatura diferente a la ambiental se mueve por la zona de detección de la cámara.

La elección del tipo de cámara a usar si solo de captura de foto o también de video depende de lo que se requiera como objetivo principal siendo las cámaras de fotografía digitales y modernas de igual costo que las de video. La desventaja con las de video es que solo pueden ser usadas en situaciones donde es posible revisar las cámaras frecuentemente. La capacidad de imágenes que puede almacenar las cámaras digitales depende de la memoria SD que use, pero como mínimo debería poder almacenar 250 imágenes (Ancorenaz, Hearn, Ross, Sollmann, & Wilting, 2012). Los modelos digitales “duermen” para disminuir el consumo energético mientras no ocurra alguna detección y una vez que la cámara se dispone a tomar la imagen debe haber un retardo pequeño, mientras la cámara se “despierta”.

Todas las cámaras tienen un tiempo mínimo de proceso entre la toma de las imágenes, desde el encendido del flash y el almacenamiento de la imagen en

la memoria SD. Si el tiempo entre imágenes es muy grande, es posible que se pierdan imágenes si es que los animales viajan en grupos. Por este motivo, es necesario que el tiempo entre la toma de imágenes sea lo más mínimo posible. La única desventaja de reducir el tiempo entre las imágenes es que si un animal se encuentra frente a la cámara esta se disparará muchas veces pudiendo llenar la memoria. En consecuencia, se considera conveniente el uso de memorias SD de gran tamaño (Ancrenaz, Hearn, Ross, Sollmann, & Wilting, 2012).

Muchas de estas cámaras tienen la particularidad de disponer de un flash de luz blanca o infrarroja. La ventaja de la luz blanca es la claridad en el detalle de las imágenes cuando se toman por la noche mientras que las tomadas con la luz infrarroja a menudo son borrosas. Esto no debería ser un problema si el principal objetivo del estudio es documentar la presencia de mamíferos, sin embargo, algunos biólogos prefieren identificar rasgos específicos de algunos animales. Una desventaja de la luz blanca es que el brillo de la luz puede provocar que los animales se espanten y se alejen del alcance de la cámara intuitivamente reduciendo las probabilidades de futuras detecciones.

Otra característica de estas cámaras es su autonomía, pudiendo estar activa durante días incluso meses capturando imágenes. Para tener esta funcionalidad, es importante que las cámaras dispongan de un modo para ahorrar energía eléctrica cuando no detecte la presencia de animales a la vez de una rápida respuesta al momento de detectarlo. En la Tabla I se muestra las diferencias entre elegir una cámara con detección de video y otra con solo fotografía.

2.1 Puesta en campo de cámara

La configuración y elección de la cámara trampa es tan importante como la elección del lugar y el entorno donde será instalado para la toma de imágenes de las especies que deseamos.

2.2 Altura de la cámara

La altura para fijar la cámara depende del tipo de especie que se quiere captar. Las cámaras infrarrojas pasivas son las más usadas, pero dependen del calor corporal del animal cuando pasa por la cámara para su detección. Si lo que se desea es la captura de animales grandes es recomendable que la cámara se fije de 30 a 40 cm por encima del nivel cero para obtener buenos resultados (Ancrenaz, Hearn, Ross, Sollmann, & Wilting, 2012). Algunas veces es mejor ubicar las cámaras por encima de ese rango para evitar que se fotografíen animales de mayor tamaño; sin embargo, esto no deberá ser un problema si se dispone de baterías de larga duración y una memoria de gran capacidad.

Tabla I
Diferencias entre diferentes tipos de cámaras trampa.

Cámaras de video	Cámara fotográfica digital
Relativamente baratas	Relativamente caras
Se tiene que esperar el procesamiento	Resultados instantáneos (incluyendo imágenes de prueba)
Encendido rápido	Encendido lento
Permite un máximo de 36 imágenes.	Permite un máximo mayor a 250 imágenes.
Pocas imágenes por la poca duración de la batería.	Muchas imágenes por la larga duración de la batería.
Sensor infrarrojo activo	Sensor infrarrojo pasivo
Relativamente caro	Relativamente barato
Se puede ajustar para evitar la toma de imágenes de especies de menor tamaño.	Muy difícil de evitar especies no deseadas.
Propenso a activarse por efecto de la lluvia o insectos grandes.	Propenso a activarse por efecto de la radiación del sol sobre la vegetación o sombras moviéndose por el lugar.
La instalación es un poco difícil.	De fácil instalación.

2.3 Angulo de la cámara

Para fijar el ángulo de orientación de la cámara, es recomendable la elección de un terreno plano; sin embargo, en muchos bosques tropicales el terreno no es necesariamente plano, pudiendo ser inclinado con respecto al nivel cero de la cámara en zonas paralelas o perpendiculares. El ángulo de la cámara en relación con el nivel cero y el área donde ha sido instalada es un aspecto importante para la configuración de la misma debido a que su configuración e instalación podría ser dificultosa. Las cámaras infrarrojas pasivas tienen predominantemente un área horizontal de sensibilidad por tanto la cámara necesita estar inclinada de manera que esta área permanezca horizontal con relación a la pendiente del camino. Una forma de lograrlo es inclinando la cámara junto con el sensor de modo que las imágenes resultantes sean horizontales.

La mayoría de las cámaras incluyen un modo de prueba durante el cual es posible moverse delante de ella y ver como un LED nos señala que efectivamente se está capturando las imágenes. Es importante recordar que la mayoría de los animales son pequeños en comparación con las personas y por lo tanto es necesario gatear en lugar de caminar delante de la cámara durante las pruebas, tal y como se puede apreciar en la figura 1.



Fig. 1.- En esta En esta fotografía se puede observar a una persona gateando delante de la cámara para corroborar una correcta captura del posible animal (*Ancrenaz, Hearn, Ross, Sollmann, & Wilting, 2012*).

2.4 Distancia al sendero o trocha

La distancia de la cámara a la trocha depende del modelo de la cámara y en su medida al tipo de animal que se quiere captar. En todos los modelos de cámaras existe un pequeño retraso entre el disparo de la cámara por la presencia del animal y la imagen que se está tomando y almacenando en la memoria, aunque en modelos mucho más modernos este tiempo es insignificante. Si este tiempo es mayor que 1 segundo y está situado muy cerca de la trocha, el animal puede haber pasado por la cámara realizándose la captura de la imagen de la cola del animal. Los beneficios de colocar la cámara a cierta distancia de la trocha deben ser comparados con los problemas de identificación de las especies pequeñas ya que las imágenes serían pequeñas y no se podrán identificar los rasgos específicos y de interés del animal. Una clara imagen capturada por una cámara trampa es la que se puede apreciar en la figura 2.



Fig. 2.- Captación de la presencia de un felino en donde se observa la orientación de la cámara hacia la trocha y la nitidez de la imagen del animal captado (*Ancrenaz, Hearn, Ross, Sollmann, & Wilting, 2012*).

Existe una gran gama de módulos de cámaras CMOS disponibles en el mercado. Estos se pueden conectar fácilmente a una placa de prueba a través de puertos UART o SPI y a su vez los fabricantes generalmente proporcionan un programa de prueba. Estas cámaras tienen como resolución típica 0.3 MP, siendo este nivel de resolución un problema ya que las imágenes aparecen pixeladas cuando son ampliadas. Debido a ello se requiere una mejor calidad para extraer la información detallada de las imágenes siendo una resolución de 2 MP lo suficientemente bueno (Pitt, 2013). En el mercado es destacable un módulo de cámara CMOS llamado ArduCAM que puede ser usado con cualquier placa de Arduino. Existen tres características que resaltan de este módulo, siendo las más importantes las siguientes:

- Sensor CMOS OV2640 y OV5642 de 2 y 5 megapíxeles respectivamente.
- Procesador FPGA
- Memoria FIFO dentro del módulo.

El FPGA controla al sensor CMOS y usa el bus SPI para transferir los datos de la imagen a otro microprocesador. Adicionalmente, el sensor CMOS es directamente configurado a través del bus I2C. El comportamiento del FPGA es bueno, pero hay que tener en consideración como puntos claves la alimentación y el ahorro de energía eléctrica, que es una consideración importante a tomar en cuenta en el presente trabajo. Teniendo en cuenta estas ventajas y desventajas el módulo ArduCAM fue usado como referencia para el diseño del software y hardware de este trabajo de tesis.

CAPITULO 3. HIPOTESIS Y VARIABLES

3.1 Hipótesis general

El diseño de un sistema de registro de imágenes de 5MP de bajo consumo mejora el monitoreo e inventario de animales grandes en zonas rurales del país.

3.2 Hipótesis específica

- El diseño de dispositivos electrónicos para un sistema de registro de imágenes de 5MP permite el monitoreo e inventario de los animales grandes en zonas rurales del país.
- El diseño de un sistema de registro de imágenes de 5MP mejora la cuantificación de mamíferos captados en el monitoreo e inventario de animales grandes en zonas rurales del país.
- El diseño de un sistema de registro de imágenes de 5MP mejora la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país.

3.3 Variables

- Sistema de registro de imágenes de 5MP.
- Monitoreo e inventario de los animales grandes.

En el Anexo 1, se presenta la Matriz de consistencia en donde se detallan los indicadores relacionados para cada una de estas variables.

3.4 Operacionalización de las variables

En el Anexo 1, se detalla el cuadro de operacionalización de las variables, detallando sus indicadores.

3.5 Matriz de consistencia

En el Anexo 1, se detalla el problema, los objetivos y la hipótesis.

CAPITULO 4. MATERIALES Y METODOS

4.1 Área de estudio

El tipo de investigación del presente estudio es descriptivo - explicativo, puesto que describe el diseño de un módulo de cámara de 5MP para apoyar a los biólogos en el inventario de los mamíferos grandes. (Hernández-Sampieri & Mendoza Torres, 2018).

A su vez, la presente tesis es de corte Correlacional según Sampieri (2018) ya que se tiene como propósito medir el grado de relación que existe entre dos o más conceptos o variables, se miden cada una de ellas y después se cuantifican y analizan la vinculación. Estas correlaciones se sustentan en hipótesis sometidas a pruebas.

En cuanto a la población y muestra, tal como indica Sampieri (2018) queda a consideración del investigador la población y muestra de estudio, al ser del tipo descriptivo – explicativo.

CAPITULO 5. COMPONENTES DEL MÓDULO CÁMARA

Como se mencionó anteriormente en el módulo ArduCam, el sensor CMOS es controlado por un FPGA. Este sensor es directamente configurado a través del bus I2C. EL FPGA trabaja bien, pero el consumo de energía eléctrica es importante en este módulo por lo que la administración de la energía tiene que ser ejecutada por otro microcontrolador (por defecto una placa de Arduino), incrementando aún más su consumo. Para el diseño de un sistema registrador de imágenes que trabaje de forma autónoma por alrededor de 3 meses, el consumo de energía del módulo ArduCAM tiene una gran desventaja. Teniendo en cuenta estas ventajas y desventajas, se decide diseñar una placa embebida y usar el mismo sensor CMOS además de la inclusión de una memoria FIFO, pero descartando el uso del FPGA.

La empresa OMNIVISION se ha especializado en la fabricación de sensores de imagen con distintas resoluciones que varían desde 0.3MP hasta 16MP, pero el inconveniente es que no proporcionan ningún software para poder probar el sensor, esto fue resuelto con la información proporcionada en el manual del ArduCAM. Para capturar imágenes de 5 Megapíxeles OMNIVISION ofrece la serie OV564x. Dentro de este grupo se decidió utilizar el modelo OV5642 que ya viene incluido en un módulo fabricado por la empresa UCTRONICS (UCTRONICS, 2015) el cual se usó para el periodo de pruebas.

Para el control del módulo OV5642 se escogió un microcontrolador dsPIC de la familia Microchip y no un FPGA, para así evitar el incremento del consumo de energía eléctrica.

Una gran ventaja del procesador dsPIC es el multiplicador de frecuencia interno (PLL) que permite obtener velocidades de hasta 80 millones de instrucciones por segundo (MIPS). Además de un bus de acceso directo a memoria (DMA) que puede dar una velocidad extra al bus SPI e I2C y finalmente las instrucciones de registro de la MAC (Acumulador múltiple) que puede ayudar a realizar algunas operaciones de procesamiento de imágenes.

La experiencia previa con los procesadores Microchip dsPIC dio una razón más para utilizar este procesador principal. Se eligió el modelo dsPIC33EP256GP502 debido a la cantidad de pines disponibles (54) y el tamaño de la empaquetadura, lo que ayuda a desarrollar una placa de pequeñas dimensiones.

Cuando el sensor CMOS envía los datos de la imagen, esta es transmitida a su propia velocidad. La velocidad de lectura efectuada por el microcontrolador puede ser más lenta, ya que no hay un bus dedicado y los datos deben procesarse antes de una nueva operación de lectura. Para evitar la pérdida de datos, una buena alternativa es el uso de una memoria temporal que permita manejar la velocidad de escritura del sensor CMOS y la velocidad de lectura del dsPIC. En este contexto se pudo indagar que la empresa Averlogic ofrece memorias FIFO para transmisión de video y aplicaciones tipo cámara. Las imágenes de 5 MP pueden llegar a pesar alrededor de 1 MB. El modelo AL440B ofrece 512 KB de memoria que pueden almacenar la mitad de esta imagen aproximadamente, dándole al dsPIC el tiempo suficiente para leer los datos mientras la cámara sobrescribe los datos existentes.

Por otro lado, el dsPIC al no tener suficiente memoria como para almacenar las imágenes, obliga necesariamente a requerir de algún tipo de memoria externa del tipo SD. Existe una variedad de librerías para poder controlar las memorias SD muchas de ellas usan lenguaje C el cual es muy útil para programar microcontroladores.

5.1 Procesador dsPIC33

Microchip ha desarrollado desde hace años los conocidos procesadores PIC y posterior a ese modelo se desarrolló la serie dsPIC. Para este trabajo se usó el dsPIC33EP256GP502 que dispone de una memoria de 256 Kbyte, un oscilador interno de 7.37 MHz y una alimentación de 3.3 voltios. Este procesador tiene la ventaja de que posee un PLL interno que multiplica la frecuencia de trabajo hasta 70 MIPS, es decir mayor rapidez para ejecutar

tareas. Asimismo, posee instrucciones especiales que se encargan de ejecutar funciones de procesamiento digital de señales. A nivel de los protocolos de comunicación, el módulo UART puede llegar a trabajar hasta 17.5MBps, el SPI a 15MBps y el I2C hasta 1MBaud. La mayor ventaja es que estas funciones solo emplean un ciclo de reloj para ejecutar la operación mientras que en un procesador normal se deben realizar varias operaciones. De igual manera, otra ventaja es que el dsPIC posee registros de 16 Bits lo cual permite realizar operaciones con variables enteras.

5.2 Sensor OV5642

Los sensores CMOS de OMNIVISION son empleados en celulares debido a su pequeño tamaño y a la variedad de módulos que capturan imágenes a diferentes resoluciones. Como se explicó anteriormente, el modelo OV5642 ya es comercializado por ARDUCAM como el módulo “Arducam Mini de 5 Megapíxel”. Ellos ofrecen librerías en Arduino para controlar este sensor. Al adaptar estas librerías para el procesador dsPIC seleccionado se puede ahorrar tiempo en el desarrollo del funcionamiento del sensor. Asimismo, al usar un sensor pequeño se puede desarrollar una placa pequeña hecha a medida y obtener una cámara trampa de menor tamaño.

5.3 Memoria FIFO AL422B

La memoria fifo AL440B de 4Mbits (512K x 8 bits) provee puertos independientes de entrada y salida de 8 bits que pueden operar a una frecuencia máxima de 80Mhz. El direccionamiento y los punteros de control del circuito brindan una fácil interfaz de memoria que reduce el tiempo del diseño. Fabricado bajo la tecnología memoria embebida de alta densidad, el AL440B usa una tecnología de alta performance con funciones adicionales de control permitiendo la fácil lectura y escritura para visualización de video PIP, televisión digital, sistemas de seguridad y aplicaciones de video. Los flags

pueden usarse para indicar si la memoria FIFO se encuentra llena o vacía permitiendo además la configuración en cascada para incrementar la capacidad de almacenamiento.

5.4 Memoria SD

La memoria μ SD que se utilizó es la Samsung EVO (32Gb) clase 10, esta memoria puede alcanzar una velocidad de transferencia de hasta 48MB/s. La comunicación con el dsPIC33 es mediante el protocolo SPI en la que se tiene una restricción debido a que la máxima frecuencia de trabajo de estas memorias usando dicho protocolo es de 25Mhz siendo una característica del microcontrolador alcanzar hasta los 30Mhz como máximo por SPI.

5.5 Sensor de Movimiento AMN41121 (PIR)

El PIR (*passive infrared sensor*) es un sensor electrónico que mide la radiación infrarroja emitida por objetos en su campo de detección. Todos los objetos por encima del cero absoluto, emiten calor en forma de radiación la cual es imperceptible por el ojo humano pero que puede ser detectada por dispositivos electrónicos diseñados para este propósito. El término pasivo se debe a que este dispositivo se encarga de detectar la radiación infrarroja emitida o reflejada por un objeto. La peculiaridad de este sensor es el bajo consumo (46 μ A) con un alcance máximo de 5m.

5.6 Real Time Clock (RTC)

Al momento de capturar las imágenes es necesario registrar el momento en que se hizo la toma. Para ello se empleó un chip que puede almacenar el tiempo en formato digital. Para este trabajo se usó el dispositivo DS3231 que

se retrasa 9 segundos en un mes. La fecha y hora es entregada al microcontrolador la misma que es registrada en cada imagen que se toma.

5.7 Flash con luz amarilla

Para la captura de imágenes nocturnas es necesario contar con un sistema de iluminación. En base a las pruebas realizadas se determinó que el sensor CMOS solo puede detectar radiación de longitudes de onda de la región visible que es perceptible por el ojo humano. Por ello se buscó implementar un flash con luz blanca. Sin embargo, los componentes no trabajan con un nivel de voltaje de 3.3 voltios razón por la cual se buscó otro dispositivo de longitud de onda de operación de color similar. Para construir un flash con luz amarilla, se encontraron los componentes necesarios para que trabaje a 3.3 voltios y de esta forma se logra mantener el consumo de energía lo más bajo posible.

5.8 Súper capacitor

Para que el flash pueda iluminar adecuadamente el área de captura de imágenes es necesario usar una cantidad de mínima de 20 LEDS iluminando a su máxima potencia. Al usar esta configuración el consumo energético se eleva hasta 1A. Para evitar que los demás componentes de la cámara se vean afectados por este repentino consumo de energía, se colocó un súper capacitor para que se comporte como una pequeña pila. De esta forma cuando el flash está apagado el capacitor se carga y cuando se enciende, toda la energía proviene del capacitor. Aliviando de esta forma la descarga de la batería.

5.9 Fotorresistencia

Para que el microprocesador pueda determinar si es de día o de noche, se usó una resistencia que es sensible a longitudes de onda de la región visible.

Cuando es de día el valor de la resistencia se eleva y de noche su valor baja a casi 0Ω . Midiendo el valor de la resistencia el microprocesador puede saber cuándo esta oscuro en base a un valor umbral de referencia.

5.10 Sensor de temperatura y humedad

El módulo de la cámara posee También un pequeño sensor que mide la temperatura y la humedad. El sensor es el HDC1000 de Texas Instruments y se comunica con el microprocesador por el bus I2C.

5.11 Mosfet de potencia

Para lograr un máximo ahorro de energía se busca apagar totalmente los componentes de la cámara cuando el microprocesador entre en el modo de ahorro de energía. Un mosfet de potencia actúa como un interruptor y es controlado por voltaje. Cuando el microprocesador termina todos sus procesos, puede entrar en el modo de bajo consumo de energía (modo ahorro de energía), pero antes de realizar esto apaga todos los componentes del módulo cámara con el mosfet.

CAPITULO 6. DISEÑO DEL HARDWARE

En esta sección se explica los criterios técnicos utilizados para ensamblar la cámara. Lo primero que se estableció fue el voltaje; en este caso, el sensor CMOS, el microcontrolador dsPIC y la tarjeta μ SD funcionan a 3,3 voltios. El microcontrolador utiliza el bus I2C para configurar el sensor CMOS. En el caso de la salida de datos, el sensor CMOS utiliza entre 8 a 10 líneas digitales (DOUT2– DOUT9 o DOUT0 – DOUT9), los pines digitales se administran en puertos de ocho pines en el dsPIC.

Esta característica permite leer los datos de cada puerto en una sola instrucción. Además, la memoria FIFO (AVERLOGIC, 2016) lee los datos en ocho líneas digitales y envía los datos de la misma forma. Por lo tanto, en este diseño actual, el sensor CMOS escribirá en la memoria FIFO y el dsPIC leerá los datos de esta a través de ocho líneas digitales. Algunas líneas de control y conexiones se muestran en la Fig. 3.

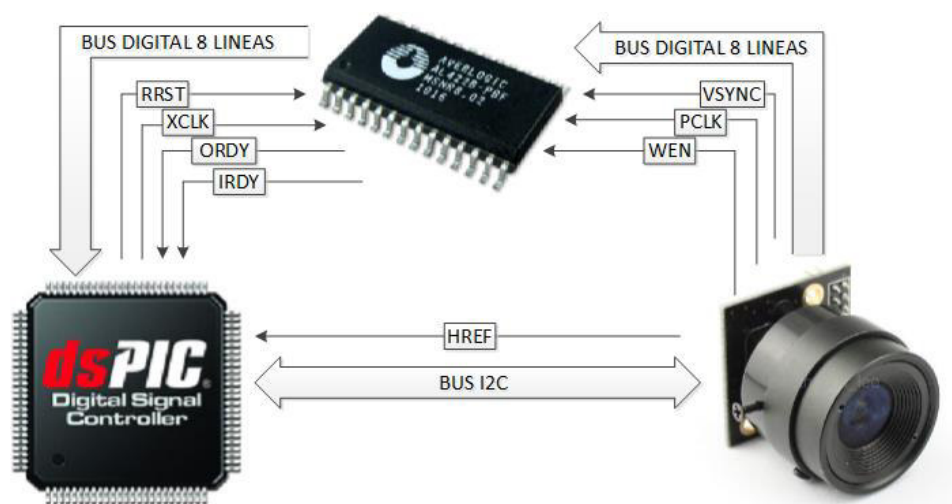


Fig. 3.- Diagrama de conexiones entre la memoria FIFO, sensor OV5642 y el dsPIC.

Los datos son extraídos con cada flanco de subida del pin PCLK. La frecuencia del sistema viene dada por un oscilador externo en el pin XCLK, esta frecuencia También se utiliza para generar la señal del reloj externo en el pin PCLK. Los pines HREF y VSYNC se utilizan para indicar el comienzo y el final de la transmisión de datos de la imagen. Finalmente, el pin PWDN es utilizado para apagar el sensor.

A continuación, se describen las funciones para el control de los pines:

1. La referencia horizontal (HREF) conectada directamente al dsPIC, indica que el sensor CMOS está enviando información.
2. La sincronización vertical (VSYNC) está conectada al reinicio de escritura (WRST) de la memoria FIFO y a un pin del dsPIC. El pin envía un pulso para restablecer el puntero del registro de escritura del FIFO.
3. La habilitación de escritura (WE) es una combinación de HREF y un pin digital dsPIC; estas señales pasan a través de una puerta lógica NAND. La memoria FIFO puede aceptar datos mientras este pin este habilitado.
4. El CMOS genera una señal de reloj para escribir datos en la memoria este es el reloj de los pixeles o pixel clock (PCLK) y está conectado al reloj de escritura o write clock (WCK).
5. El pin de restablecimiento de lectura (RRST) envía un pulso para restablecer el puntero del registro de lectura de la memoria FIFO.
6. El microprocesador genera una señal de reloj (RCK) para leer los datos de la memoria.
7. El pin (ORDY) de la memoria, se habilita cuando los datos se pueden leer desde la memoria FIFO después de 191 pulsos de escritura, cuando los datos están listos para leer.
8. El pin (IRDY) se habilita cuando la memoria necesita 191 pulsos de escritura para alcanzar su capacidad máxima.

Adicional a los pines descritos, la memoria FIFO tiene un pin de polaridad (PLRTY). De acuerdo con el estado de este pin, la lógica de los otros pines descritos puede cambiar para adaptarse a las necesidades del usuario. En el tablero desarrollado, PLRTY se configuró a 3.3 voltios con la finalidad de

hacerlo compatible con la lógica del sensor CMOS y de esta manera los pines se activen en un 0 lógico.

El CMOS requiere un oscilador externo (XCLK) que no se encuentra en el módulo propio de la cámara. El rango de valores de frecuencia aceptados va de 6 a 27 MHz. En la placa desarrollada se usó un pin del microcontrolador el cual nos permite generar 7.5MHz y de esta manera tener velocidades de lectura rápidas. El CMOS utiliza esta frecuencia para ajustar la frecuencia de salida de la velocidad de lectura (PCLK).

Finalmente, la tarjeta SD se conecta al dsPIC a través del bus SPI. La velocidad de lectura y escritura de las tarjetas SD varían según el proveedor (Matt, 2015). Se realizó un análisis comparativo con cinco tarjetas diferentes micro SD Clase 10 y se obtuvieron diferentes velocidades de lectura y escritura. Finalmente se seleccionó la tarjeta EVO de Samsung debido a los buenos comentarios de otros investigadores.

6.1 Diseño y control del prototipo del módulo cámara.

La cámara consta del sensor de imagen OV5642 de OmniVision y el microprocesador dsPIC33EP. La captura de imágenes se hace modificando los valores lógicos digitales ('0' o '1') presentes en la memoria del sensor. Para acceder a la memoria se utiliza un protocolo de comunicación serial llamado "I2C"; por medio del cual el microprocesador le indica al sensor en qué región de la memoria desea modificar sus valores. Una vez que estos valores son modificados el sensor se encargará de ejecutar la orden correspondiente según indica el fabricante. Con este proceso se especifica el formato de compresión de la imagen (RGB, YCC y JPEG), la resolución de la imagen (1MP, 2MP, 5MP, etc.) y configura otros parámetros que corresponden a los ajustes del sensor como el control de blancos, reducción de ruido, nivel de exposición a la luz, etc. para luego almacenarlo en una memoria μ SD.

Para obtener la data digital de una imagen es necesario seguir una serie de pasos sincronizados desde el sensor CMOS. Después de enviar la orden

desde el dsPIC, el sensor CMOS usará 2 líneas de control llamadas (VSYNC y HREF) para indicar que la información se está enviando. Dado que una imagen de 5 Megapíxeles pesa aproximadamente 1.5 Megabytes es necesario almacenar esta información en una memoria no volátil. El dsPIC solo tiene capacidad para almacenar 256 Kilobytes de este modo es necesario una memoria SD externa de 32 Gigabytes de capacidad. El sensor CMOS trabaja a una frecuencia de 7.5Mhz mientras que el dsPIC a 60Mhz. Sin embargo, para almacenar la imagen el dsPIC debe leer los datos del sensor y a la vez almacenarlos en la memoria SD. Esto puede generar que el dsPIC se retrase durante la lectura y generar pérdida de información. Para evitar esta pérdida se optó por usar una memoria volátil FIFO que almacene temporalmente los datos de la cámara y los entregue al dsPIC conforme los solicite. De esta forma se puede garantizar que el dsPIC almacene una imagen correctamente.

Cuando se envía satisfactoriamente los comandos, empieza el envío de información de la imagen. El pin VSYNC de la cámara envía un pulso para indicar que el envío de información ha comenzado, mientras el pin WEN este activo el sensor enviará datos a la memoria FIFO, del mismo modo el pin HREF envía pulsos de reloj para indicar el momento en que la información está disponible para su lectura. Finalmente, cuando se termina de enviar todos los datos de la imagen el pin VSYNC nuevamente envía otro pulso. Los pulsos del pin VSYNC y HREF se pueden ver en la figura 4.



Fig. 4.- Pulsos VSYNC y HREF captados por el osciloscopio. Fuente: elaboración propia

6.2 Comportamiento de la memoria FIFO durante la lectura y escritura de datos.

6.2.1 Comportamiento durante la escritura.

Durante la escritura la información recibida es almacenada en un registro “write-data-register” antes de escribirla en la memoria física.

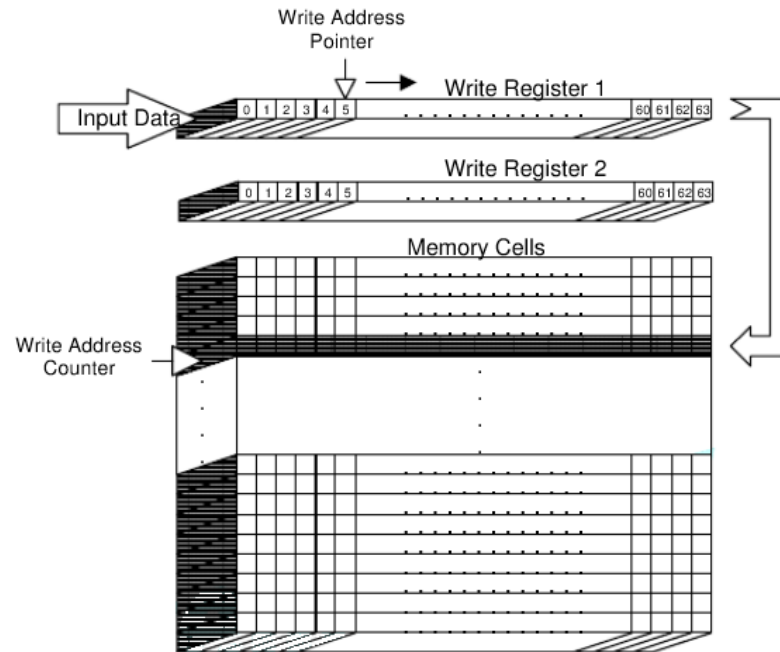


Fig. 5.- Operación de escritura de datos en los registros (AVERLOGIC, 2016).

EL FIFO AL440B tiene 2 registros “write-data-registers” que trabajan de forma alternada. Cada “write-data-register” puede almacenar hasta 64 bytes de información. Cuando uno de los “write-data-registers” se llena, estos 64 bytes de información son cargados rápidamente a la memoria física. A cada uno de los 64 “write-data-registers” se le asignan direcciones del 0 al 63. El puntero de escritura en los registros de escritura indica la posición donde la información será escrita. La figura 5 muestra que el punto de escritura está en la 5ta dirección, indicando que la información que será escrita después será guardada en la dirección 5 del registro de escritura. En la figura, las direcciones de 0 a 4 han sido llenadas con información. El puntero de escritura es incrementado en 1 cada vez que se escribe información es decir en cada ciclo del WCK. Cuando el registro de escritura llega a llenarse con nueva información, los 64 bytes de información son almacenados en la memoria física. En este momento, el contador de escritura es incrementado en 1 cada vez que los datos sean transferidos a la memoria física desde el registro de escritura. Cuando el registro de escritura 1 se llena y se encuentra en el proceso de escritura a la memoria, la siguiente información que ingresa será

almacenada en el registro de escritura 2. De esta manera los dos registros de escritura trabajan de forma alternada permitiendo que la información sea ingresada continuamente.

6.2.2 Comportamiento durante la lectura.

Durante la lectura, la información está en proceso de escritura, la información será almacenada en un buffer de lectura antes de la lectura del bus.

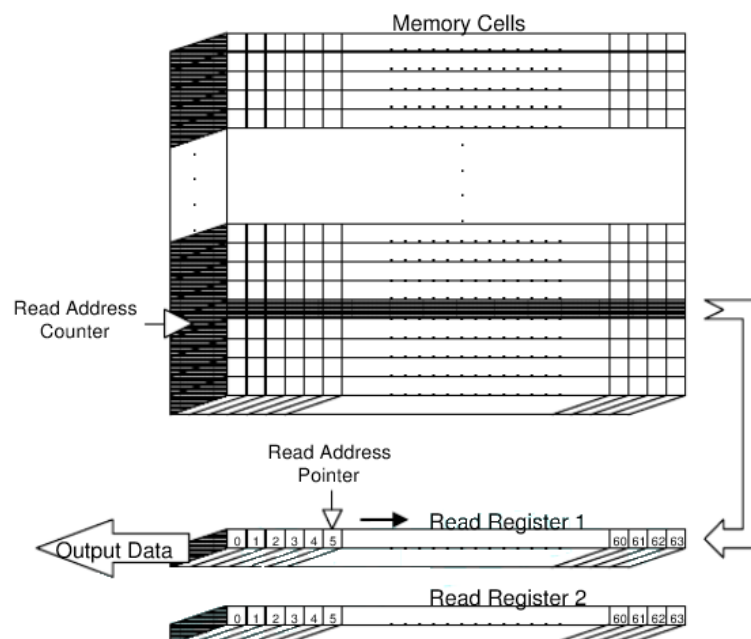


Fig. 6.- Operación de lectura de datos de los registros (AVERLOGIC, 2016).

AL igual que en la escritura el proceso de lectura tiene dos registros de lectura que trabajan de forma alternada. Cada registro de lectura puede almacenar hasta 64 bytes de información. La información de salida será transferida de la memoria al registro de lectura para la operación de lectura de la información. Cuando la operación de lectura está en proceso de lectura desde uno de los registros los próximos 64 bytes salientes son pre-cargados en el otro registro de lectura. El contador de lectura es incrementado por 1 cada vez que se envíe información desde la memoria a los registros es decir en cada flanco del RCLK. Cuando todos los 64 bytes son leídos y el registro se encuentre vacío,

la información es nuevamente transferida desde la memoria física hacia los registros. Cuando el registro 1 se vacía y hay información para ser transferida a uno de los registros esta se va al registro de lectura 2 de esta manera los 2 registros de lectura trabajan de forma alternada y la información puede ser leída continuamente.

6.3 Acceso directo a memoria (Direct Memory Access - DMA)

El control de acceso directo a la memoria (DMA) es un subsistema importante en la familia de controladores digital de señales (DSC) de alto rendimiento de 16 bits de la familia Microchip. Este subsistema facilita la transferencia de datos entre la CPU y un periférico sin asistencia de la CPU. El controlador DMA del dsPIC33 (Microchip, 2011) está optimizado para su uso en aplicaciones embebidas de alto rendimiento y en tiempo real, donde la latencia del sistema es un aspecto crítico a tener en cuenta. El controlador DMA transfiere datos entre los registros de datos de los periféricos y el espacio de datos SRAM. El subsistema DMA usa la DPSRAM y las estructuras de registro que permite que el DMA opere a través de sus propios buses de datos y direcciones independientes sin afectar el funcionamiento de la CPU. Esta arquitectura elimina la necesidad de robar ciclos, lo que detiene a la CPU cuando se solicita una transferencia DMA de mayor prioridad. Tanto la CPU como el controlador DMA pueden escribir y leer en las direcciones dentro del espacio de datos sin interferencia lo que resulta en un rendimiento máximo en tiempo real. Alternativamente, la operación de DMA y la transferencia de datos hacia/desde la memoria y los periféricos no se ven afectados por el procesamiento de la CPU. Por ejemplo, cuando se realiza una operación de programación automática en tiempo de ejecución (RTSP), la CPU no ejecuta ninguna instrucción hasta que finaliza RTSP. Sin embargo, esta condición no afecta la transferencia de datos a/o desde la memoria y los periféricos.

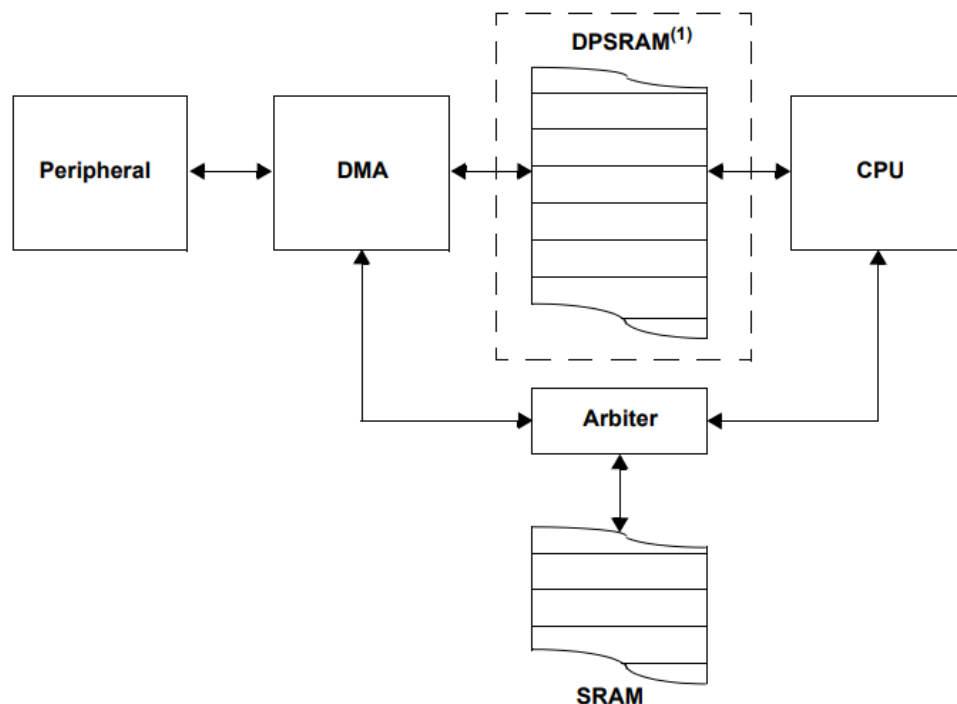


Fig. 7.- Diagrama de bloques del controlador DMA en el dsPIC33 (*Microchip, 2011*)

La figura 7 nos muestra el diagrama de bloques y cómo opera el DMA dentro del microcontrolador dsPIC.

6.4 Control de la información de la imagen.

El sensor CMOS tiene un bus de salida de datos de 8 líneas, es decir que en simultáneo el bus transmite 8 bits de información por cada ciclo de lectura. Como se explicó anteriormente, se optó por usar una memoria FIFO volátil para almacenar temporalmente los datos del sensor CMOS. La memoria FIFO posee una capacidad de almacenamiento de 512 KBytes y permite realizar la escritura y lectura de datos en simultáneo. Es decir, mientras la cámara escribe los datos en la memoria, el microprocesador puede leer los datos almacenados sin que afecte la otra operación. Internamente la memoria posee un registro de direccionamiento para la lectura y otro para la escritura, estos registros incrementan automáticamente su valor en cada lectura o escritura. Cuando se hace una nueva lectura se debe posicionar el registro de lectura y

escritura en la posición cero, por medio de los pines RST (read reset) y WRST (write reset). En el caso de alcanzar el máximo valor, los registros resetean su valor a cero y empiezan a reemplazar los valores anteriores. Para imágenes de más de 512 KBytes, la memoria FIFO vuelve a resetear su puntero de la dirección 512000 a 0 y empieza a escribir nuevamente en todos sus registros borrando la información antigua. Por esta razón el microcontrolador no debe retrasarse en la lectura de datos de la memoria FIFO o podría perder información de la imagen.

En el microcontrolador un método para que la velocidad de lectura sea lo más rápida posible, es aumentar la velocidad del almacenamiento en la memoria μ SD. Una mejora es leer los datos de la memoria FIFO y agruparlos en paquetes de 4K bytes para luego almacenarlos en una memoria μ SD a través del bus SPI. Una vez que se ha inicializado la memoria y creado el archivo, la escritura de datos no es directa. Cuando se ha creado el archivo se obtiene una referencia a la posición de memoria donde está el archivo y cada vez que se escribe un valor se debe acceder primero a la dirección de la memoria y después escribir los datos. Debido que cada acceso a la memoria SD toma tiempo se sugiere agrupar los datos en grupos grandes. Adicionalmente para reducir más el tiempo de escritura de datos en la memoria SD se va a usar la transferencia por DMA.

En un microprocesador cuando se desea transferir datos a un bus físico digital se deben emplear varias instrucciones hasta que se complete totalmente la transferencia. Sin embargo, al usar DMA se crea un canal de acceso exclusivo para el bus digital, de modo que el procesador queda libre para realizar otras tareas pendientes. De esta forma el tiempo de ejecución de un programa se puede reducir en algunos milisegundos. En este caso el bus DMA será usado para el bus SPI para de esta manera dejar libre el microprocesador y seguir leyendo los datos de la memoria FIFO.

CAPITULO 7. CONTROL DE LA CAMARA

La programación del dsPIC fue realizada haciendo uso del IDE de Microchip (MPLAB) en lenguaje C y para poder compilar los códigos en el dsPIC33, se usó el compilador XC16. La frecuencia a la cual trabaja el microcontrolador puede ser configurada a 70 o 60 MIPS modificando los valores en los registros CLVDIV y PLLFDB. En este caso para el desarrollo de esta tesis se configuró a 60 MIPS.

Para poder realizar el almacenamiento de las imágenes en la tarjeta μ SD se utilizó la librería FATFS (Cham, 2019), est

a librería solo posee instrucciones de lectura y escritura de los archivos creados en la memoria, las funciones para el control del hardware han sido escritos basados en los ejemplos propuestos por los desarrolladores de las librerías FATFS.

Primeramente, se configura el bus I2C para trabajar a 400KHz, el bus SPI es configurado a 20MHz y el ping RST de la memoria FIFO es puesto en alta (3.3V) y se inicializa la memoria μ SD.

El segundo paso es la toma de la imagen. En este proceso el dsPIC espera un flanco de bajada proveniente del sensor infrarrojo (PIR), la salida digital del sensor infrarrojo se encuentra conectado a un pin del microcontrolador, internamente este pin se comporta como una interrupción (INT0) de prioridad uno y por ende el microcontrolador sale de cualquier operación que se encuentre realizando o del modo ahorro energía. Posteriormente el dsPIC configura los registros del sensor CMOS a través del bus I2C, efectúa el proceso de alimentación de la cámara y la captura de la imagen. En el código del módulo ArduCAM 5MP (Arducam, 2016) que se tuvo como referencia, muestra una larga secuencia de registros específicos que deben ser escritos en un orden específico. Una vez realizado este proceso el sensor CMOS

tomara una imagen y empezara a enviar los datos a la memoria FIFO en cuadros de 8 bits.

Cada vez que el sensor CMOS envía los datos de la imagen, el pin HREF conmuta a alta (3.3 V). En la memoria FIFO el pin VSYNC conmuta a un estado bajo (0 V) por unos microsegundos y resetea el puntero del registro de escritura de la memoria FIFO a 0. Para comenzar a escribir en la memoria el pin WE debe estar en un estado "bajo". Una vez que el dsPIC lee la señal del pin HREF, automáticamente establecerá el pin WE a alta (3,3V). En la puerta NAND que se muestra en la figura 8, el estado del pin WE cambiará a "bajo" permitiendo la operación de escritura.

Por otra parte, el microcontrolador esperara a que el pin ORDY de la memoria FIFO cambie de estado a baja y luego esperara 100ms antes de leer la información que tenga disponible. Este pequeño retraso permite la correcta lectura de los datos, de otra manera la memoria podría entregarnos valores llenos de ceros (0x00).

En el caso del pin de reloj de lectura de la memoria FIFO (RCK) se podría generar usando una modulación de ancho de pulso, pero de esta manera el microcontrolador no podría controlar exactamente cuando la señal está en estado "alto" o "bajo". Para tener un mejor control, el dsPIC cambiará el estado del pin y durante los flancos de bajada, se leerán los datos del FIFO. Con este método la frecuencia de lectura obtenida es de 10 MHz.

Para escribir en la tarjeta SD, el microcontrolador acumulará 4 KB de datos. Se prefiere este método debido a que el acceso a la tarjeta SD lleva un tiempo innecesario y hacer esto por pequeñas cantidades puede ralentizar el proceso de lectura. Para detener la lectura de datos de la memoria FIFO, el dsPIC esperará otro pulso en el pin VSYNC, después de este pulso comparará los datos de lectura hasta que encuentre los bytes de "fin de imagen", en una imagen JPEG estos bytes tienen el valor 0xFF y 0xD9. El pin IRDY se usa para saber cuándo el tamaño de la imagen excede la capacidad de la memoria FIFO (512 KB); si esto sucede, el puntero de escritura se restablece a cero y comienza a sobrescribir los datos existentes.

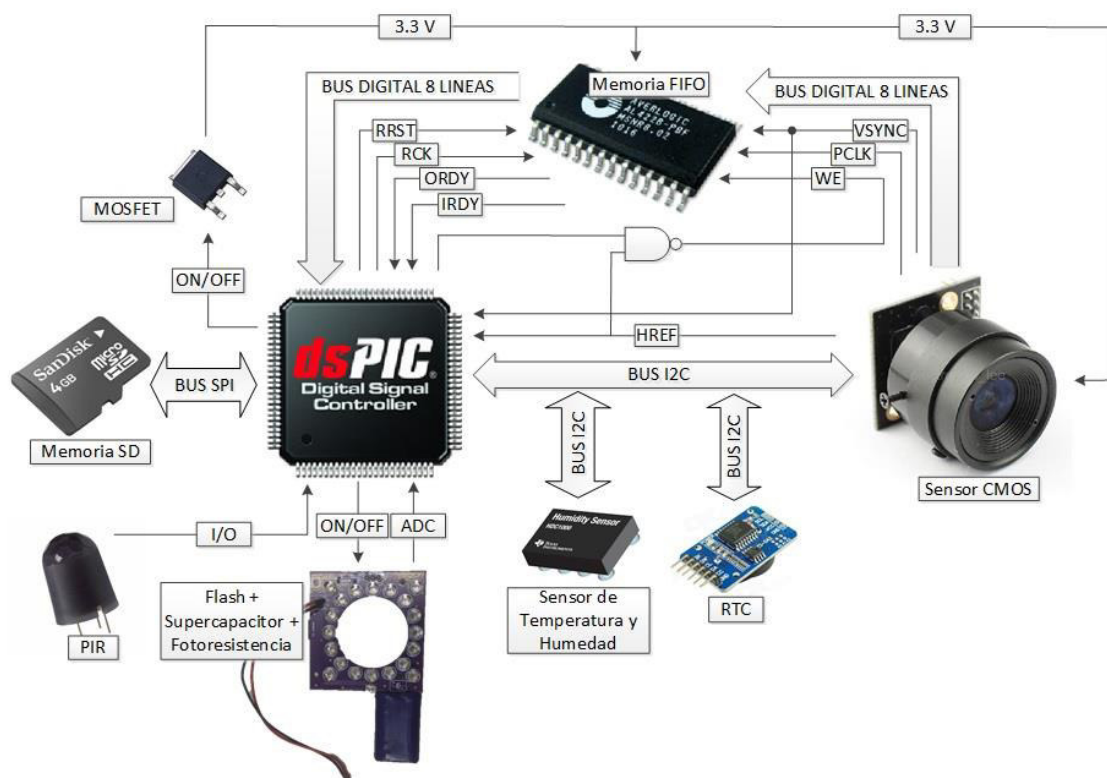


Fig. 8.- Diagrama de conexiones del sistema diseñado. Fuente: elaboración propia

CAPITULO 8. CONFIGURACIÓN DE PERIFERICOS DEL dsPIC PARA UN ÓPTIMO COMPORTAMIENTO COMO REGISTRADOR DE IMÁGENES

8.1 Funciones necesarias para el correcto inicio de la cámara

- ***Configuración del oscilador interno***

El sistema de oscilador dsPIC33E (Microchip, 2011) incluye las siguientes características:

- Cuatro opciones de oscilador externo e interno.
- Oscilador auxiliar que proporciona la fuente del reloj al módulo USB (no disponible en todos los dispositivos).
- Loop Phase-Locked Loop (PLL) en el chip para aumentar la frecuencia de operación interna con ciertas fuentes de osciladores internos y externos.
- Modo Doze para ahorro de energía del sistema.
- Monitor de reloj a prueba de fallas (FSCM) que detecta fallas en el reloj y permite la recuperación o el cierre seguro de la aplicación.
- Frecuencia de reloj de referencia escalable (REFCLKO)

Se configuró el oscilador y el PLL dentro la función CPU_Init para que trabaje a 120 Mhz. Según el datasheet podemos configurar los siguientes macros:

- PLLPRE<4:0>(CLKDIV<4:0>) = 00000 → N1 = 0
- PLLPOST<1:0>(CLKDIV<7:6>) = 00 → N2 = 0
- PLLDIV<8:0> = 0X03F → 63 en decimal

Calculando la frecuencia

$$F_{OSC} = F_{IN} \left(\frac{M}{N_1 \times N_2} \right) = F_{IN} \left(\frac{(PLLDIV_{<8:0>} + 2)}{((PLLPRE_{<4:0>} + 2) \times 2 (PLLPOST_{<1:0>} + 1))} \right)$$

$$= 7.37 \times \frac{65}{2 \times 2} = 119.7625 \text{ Mhz}$$

$$F_p = F_{cy} = \frac{F_{OSC}}{2} = 59.88125 \text{ Mhz}$$

Para mayores detalles ver el diagrama de bloques del Anexo 3.

```
void CPU_Init(void) {
    RCONbits.VREGS = 1;
    RCONbits.VREGSF = 1;

    // Sequence to set/clear IOLOCK for controlling Peripheral Pin Select
    (PPS))
    OSCCONL = 0x46;
    OSCCONL = 0x57;
    OSCCONbits.IOLOCK = 0; //PPS lock bit is not locked, writes to PPS are
    allowed

    //Configurando el system clock Fosc = 119762500Hz, Fcy = Fp = Fosc/2
    CLKDIVbits.DOZEN = 0; //Force Fcy divided by 1
    CLKDIVbits.FRCDIV = 0b000; //Internal Rast RC postcaler divided by 1
    CLKDIVbits.PLLPOST = 0; //N2 =
    CLKDIVbits.PLLPRE = 0; //N1 = 2
    PLLFDBits.PLLDIV = 63; //M = 65

    while(!OSCCONbits.LOCK);

    ANSELA &= ~(0x1813);
    ANSELB &= ~(0x010F);
    ANSELC &= ~(0x0807);
    TRISA |= 0x1F93;
    TRISB |= 0xFFFF;
    TRISC |= 0xBFFF;
    ANSELAbits.ANSA1 = 1;
}
```

El código anterior nos muestra el proceso de configuración de cada uno de los macros encargados de la configuración de la frecuencia de operación del microcontrolador.

8.1.1 Temporizadores. Timers de 16 y 32 bits

Existen 5 timers de 16 bits (Microchip, 2010) que pueden servir como contadores en tiempo real. Para el control de la cámara se utilizó el timer1 y timer2 como temporizadores de 16 bits, dicho temporizador (registro TMR1 y TMR2) se incrementará en cada ciclo de instrucción hasta el valor precargado

en el registro de periodo PR1 y PR2, una vez alcanzado estos valores los temporizadores vuelven a 0 y siguen contando. A su vez al coincidir estos valores (PRx y TMRx) se puede generar alguna interrupción activando el flag de interrupción TxIF del registro IFS0.

El TMR1 es el tiempo de 1ms que pide el fabricante para poder realizar una escritura tras otra en la memoria SD, para esto se configura el pre escalador en el registro TCKPS a 1, 8, 64 y 256 de acuerdo a las necesidades. Para poder hallar el periodo de interrupción del timer se tuvo en cuenta el siguiente proceso:

- El reloj interno de 60Mhz configurado anteriormente tal como se muestra en (1):

$$T_{cy} = \frac{1}{59.88125} \quad \dots (1)$$

- Con el valor calculado en (1), podemos calcular las temporizaciones aplicando la fórmula (2):

$$T_{interrupcion} = T_{cy} * Prescaler * (PR1) \quad \dots (2)$$

Reemplazando en la ecuación (2) para un tiempo de 1ms se tiene:

$$1ms = \frac{1}{59.88125} * 1 * (PR1)$$

$$PR1 = 59881$$

EL TMR2 es generado para poder contabilizar el tiempo que demora en escribir 4kb de los datos de la imagen en la memoria SD. Para esto se configuró hasta 15ms, tiempo suficiente para poder visualizar la escritura en la memoria. Para poder llegar a este valor se hizo uso de un pre escalador mayor TCKPS<2> = 64. Haciendo uso de la formula (2):

$$15ms = \frac{1}{59.88125} * 64 * (PR2)$$

$$PR2 = 14036$$

Los valores hallados de PR1 y PR2 son asignados en la función InicializaTIMER1() como se muestra continuación:

```
void InicializaTIMER1(void) {
    //----- SETUP TIMER 1 -----
    T1CON = 0b0000000000000000;
    TMR1 = 0;
    PR1 = 59881;                // 0.99999582mseg ~ 1mseg with ~60MIPS

    IPC0bits.T1IP = 0x01;      // Set Timer 1 Interrupt Priority Level
    IFS0bits.T1IF = 0;         // Clear Timer 1 Interrupt Flag
    IEC0bits.T1IE = 1;         // Enable Timer1 interrupt
    T1CONbits.TON = 1;         // Start Timer
}

void InicializaTIMER2(void) {

    T2CON = 0x0020;
    TMR2 = 0;
    PR2 = 14036;                // Pre-escaler set to 15mseg

    IFS0bits.T2IF = 0;
    IEC0bits.T2IE = 1;
}
```

El timer 4 y 5 de 16 bits se unen en uno solo para poder ser un solo timer de 32 bits, habilitando el este modo bits en el registro T4CONbits.T32. Este timer nos servirá como un time out de 1s para que el microcontrolador pueda detectar el flanco de bajada del VSYNC, de esta manera se evita que el microcontrolador se quede en este estado para siempre esperando algún reset para poder activarse al detectar la presencia de un animal.

Se tiene que tener en cuenta que en este caso el registro PR5 alojara el valor de 32 bits más significativo y el registro PR4 el menos significativo. Ambos registros de 16 bits se unen para formar uno de 32 bits. Para poder alcanzar este valor se hizo uso de un pre escalador de valor 8.

Utilizando la fórmula (2):

$$1s = \frac{1}{59.88125} * 8 * (PR4_5)$$

$$PR4_5 = 7485156$$

El valor de PR4_5 en hexadecimal es: 0x007236E4, donde los 4 bytes más significativos le corresponderían al registro PR5 y los 4 menos significativos al PR4. Estos valores son ingresados en los registros correspondientes tal y como se parecía en la función InicializaTimer4_5() mostrada a continuación:

```
void InicializaTIMER4_5(void) {
    T5CONbits.TON = 0;      // Stop any 16-bit Timer5 operation
    T4CONbits.TON = 0;      // Stop any 16/32-bit Timer5 operation
    T4CONbits.T32 = 1;      // Enable 32-bit Timer mode
    T4CONbits.TCS = 0;      // Select internal instruction cycle
clock
    T4CONbits.TGATE = 0;    // Disable Gated Timer mode
    T4CONbits.TCKPS = 0b01; // Select 1:1 Prescaler
    TMR5 = 0x00;           // Clear 32-bit Timer (msw)
    TMR4 = 0x00;           // Clear 32-bit Timer (lsw)
    PR5 = 0x0072;          // Load 32-bit period value (msw)
    PR4 = 0x36E4;          // Load 32-bit period value (lsw)

    IPC7bits.T5IP = 0x06;  // Set Timer5 Interrupt Priority Level
    IFS1bits.T5IF = 0;     // Clear Timer5 Interrupt Flag
    IEC1bits.T5IE = 1;     // Enable Timer 5 interrupt

    T4CONbits.TON = 1;     //Start 32-bit Timer
}
```

8.1.2 Configurar clock de referencia para alimentar al sensor de la Cámara con una frecuencia de 7.5MHz

Como se mencionó anteriormente para poder generar un clock necesario para que trabaje el sensor CMOS se configuró un pin digital (RP38) del microcontrolador mediante el re-mapeo del pin del clock de referencia (REFOCLK). Para poder dividir la frecuencia de los periféricos ($f_{cy} = 59.88125MHz$) primero se tiene que habilitar el registro REFOCONbits.ROSEL = 0 luego se configura el registro REFOCONbits.RODIV en el cual se puede dividir esta frecuencia para hallar un valor cercano a 7.5MHz. Para este motivo se tiene que dividir aproximadamente entre 8 por ello se configuro el registro REFOCONbits.RODIV<3:0> = 0011 en el cual se divide la frecuencia de los periféricos entre 8 [19]. Los pasos descritos anteriormente son configurados en la función InicializaREFOCLK() que se muestra a continuación:

```

void InicializaREFCLK(void) {

    //Configuring the output the clock reference through the pin RP38
    REFOCONbits.ROON = 0; //REFCLK off
    REFOCONbits.ROSEL = 0; //Fosc is used as reference clock output
    REFOCONbits.RODIV = 0b0011; //reference divided by 8: 60MHz/8 =
7.5MHz
    REFOCONbits.ROON = 1; //REFCLK on

    TRIS_CAM_XCLK = 0; //salida
    RPOR2bits.RP38R = 0b110001; //RP38 --> REFCLKO
}

```

Para un mayor detalle ver el diagrama de bloques en el Anexo 3.

8.1.3 Configuración bus I2C

El bus I^2C del dsPIC (Microchip, 2013) puede comportarse tanto como en modo esclavo como en modo maestro haciendo uso de una interfaz de 16bits. Toda conexión en este bus se realiza a través de dos líneas SDA (datos) y SCL(Reloj), permitiendo así la comunicación en ambas direcciones. En este protocolo se pueden especificar los diversos elementos conectados para ello se tiene direcciones de 7 o 10 bits.

Cuando se trabaja como master, para el cálculo de la tasa de baudios el módulo debe generar el clock para el I2C, generalmente este clock puede ser de 3 diferentes valores como 100 KHz, 400 KHz o 1 MHz. El detalle de la aplicación de esta fórmula se ve en (3).

$$I2CxBRG = ((\frac{1}{F_{SCL}} - Delay) \times FCY) - 2 \quad \dots (3)$$

Donde el valor típico para el retardo es de $130ns$. Reemplazando valores en (3):

$$I2CxBRG = ((\frac{1}{400Khz} - 130ns) \times 59.88125MHz) - 2$$

$$I2CxBRG = 142$$

Los pasos descritos anteriormente son configurados en la función InicializaI2C() que se configura los registros correspondientes para obtener los valores que se se requieren:

```

void InicializaI2C(void) {

    TRIS_CAM_SDA = 1; //RA8 input for SDA
    TRIS_CAM_SCL = 0; //RB4 output for SCL

    I2C2MSK = 0;
    I2C2BRG = 142; // Pre-escaler for 400KHz
    I2C2CON = 0x8000; //I2CEN = 1

}

```

8.1.4 Configurar bus UART

El bus UART del dsPIC (Microchip, 2011) es un protocolo diseñado para la comunicación serial asíncrona operando de manera bidireccional. La comunicación se lleva a cabo a través de dos líneas, una para la transmisión Tx y otra para la recepción Rx.

Las principales características del módulo UART son las siguientes:

- Modo dúplex con datos de 8 a 9 bits
- 1 o 2 bits de STOP
- Bit de paridad par o impar
- Frecuencia de baudios
- Es posible operar como interrupción
- Detección de error por paridad

Como se observa el Bit de selección de baudios de alta velocidad configurado en el registro U1MODE.BRGH tiene el valor de 1, por lo tanto, el generador de baudios (BRG) generara 4 clocks por bit (4 baudios por clock, modo de alta velocidad).

La fórmula para hallar el valor de U1BRG se muestra en (4):

$$U1BRG = \frac{F_p}{4 * Baudrate} - 1 \quad \dots (4)$$

Reemplazando valores en (4) para un baudrate de 115200:

$$U1BRG = \frac{59881250}{4 * 115200} - 1$$

$$U1BRG = 129.95 - 1 = 129$$

Cabe resaltar que para inicializar el protocolo UART tenemos que habilitarlo asignando el valor 1 al registro U1MODE.UARTEN. En el siguiente código mostrado se puede ver detalladamente la configuración de cada uno de los registros descritos antes:

```
void InicializaUART1(void) {
    //Configure pin RX y TX
    TRIS_TX_UART1 = 0; //TX
    TRIS_RX_UART1 = 1; //RX

    RPOR0bits.RP35R = 0b000001; //RP35 -> U1TX
    RPINR18bits.U1RXR = 37; //RP37 -> U1RX

    U1BRG = 129; //baudrate 115200
    U1MODE = 0x8008;
    U1STA = 0x0400;

    IFS0bits.U1RXIF = 0;
    IEC0bits.U1RXIE = 1; //Enable Interrupt
}
```

8.1.5 Configurar bus SPI

El bus SPI del dsPIC (Microchip, 11) es un protocolo de comunicación para la transferencia de información entre circuitos integrados en un sistema electrónico embebido, memorias EEPROM, conversores análogo-digital, etc. EL dsPIC que se usó en este trabajo consta de dos buses SPI, SPI1 y SPI2. En un inicio se tiene el bus SPI trabajando a una mínima frecuencia (pin SCK1), estos valores están entre 100 y 400Khz establecidos por la librería ELCHAN para el almacenado de las imágenes en la memoria externa. Los pre escaladores SPI1CON1bits.SPRE (1:8) y SPI1CON1Bits.PPRE(1:1/4/16/64) están configurados de modo que el SCK1 del SP1 opere a una frecuencia en el intervalo mencionado anteriormente.

El valor de SCK1 se obtiene usando la fórmula (5):

$$SCK1 = \frac{F_p}{SPRE * PPPE} \quad \dots (5)$$

Reemplazando en (5):

$$SCK1 = \frac{59881250}{8 * 64}$$

$$SCK1 = 117 \text{ KHz}$$

Este valor este en el rango para que pueda operar perfectamente la escritura en la memoria externa. A su vez en este apartado se habilita el Chip Select (CS) de la memoria micro SD.

En la función InicializaSPI1() que se muestra continuación se detalla los pasos para la correcta configuración del bus SPI desde los registros de estatus y control (SPI1STAT y SPI1CON2) hasta los registros para funciones específicas (DISSCK, DISSDO, MODE16), luego se configura el registro SCK1 con los valores de SPRE y PPRE definidos anteriormente. Por el ultimo se habilita el bus SPI mediante el registro SPIEN.

```
void InicializaSPI1(void) {

    /******
    // SPI module Configuration
    // Uncomment your selection for each variable
    // See device data sheet and PIC18 Peripheral Library docs for more
    details

    /******
    SPI1STAT = 0b0000000000000000; //enable SPI pins
    SPI1CON2 = 0b0000000000000000; //framed SPI support disabled
    //SPI1CON1 = 0b0000001000111101; //modo 8 bits, input data
    sampled at end,
    //bus mode 00, prescal 1:1, postcal 16:1
    //master mode
    SPI1CON1bits.DISSCK = 0; // Internal serial clock is enabled
    SPI1CON1bits.DISSDO = 0; // SDOx pin is controlled by the module
    SPI1CON1bits.MODE16 = 0; // Communication is word-wide (8 bits)
    SPI1CON1bits.MSTEN = 1; // Master mode enabled
    SPI1CON1bits.SMP = 1; // Input data is sampled at the end of data
    output time
    SPI1CON1bits.CKE = 0; // Serial output data changes on transition
    from
    SPI1CON1bits.CKP = 0; // Idle state for clock is a low level;
    //SPI BUS LOW SPEED (Min 100KHz, Max 400KHz)
    //(Min speed required for ACDM41 initalisation)
    SPI1CON1bits.SPARE = 0b000; //secondary prescal 8:1
    SPI1CON1bits.PPRE = 0b00; //primary prescal 64:1
    // active state is a high level
    SPI1STATbits.SPIEN = 1; // Enable SPI module

    TRIS_SDCARD_CS = 0;
    SDCARD_CS = 1;}
```

El detalle del funcionamiento se puede ver en el diagrama de bloques en el Anexo 4.

8.1.6 Uso del acceso directo a memoria enlazado al SPI para el rápido envío de los de la memoria FIFO hacia la micro SD.

Si se establece el bit DIR, los datos se leen desde la memoria DMA del DPSRAM (usando DMAxSTA o DMAxSTB) y el destino de la escritura se dirige al periférico (usando la dirección periférica, según lo proporciona DMAxPAD). Una vez configurado, cada canal es un conducto de datos unidireccional. Es decir, si un periférico requiere datos de lectura y escritura utilizando el módulo DMA, deben asignarse dos canales: uno para lectura y otro para escritura.

Se debe tener en cuenta que el microcontrolador hace lecturas de 4Kbytes (4096 registros que puedan almacenar 8bits). Estos 4kbytes (buff_CamCapture) se almacenan en la memoria interna del dsPIC de manera momentánea y se habilita el acceso directo a memoria (DMA) creando un pequeño canal entre esta y el bus SPI, de manera que toma menos tiempo al crear y escribir en el archivo de la imagen tomada.

- DMAxCON.DIR = 1 asignando un valor de 1 a este registro se configura la lectura desde el puerto dual de espacio de datos (DPSRAM) y escribimos lo leído en algún periférico en nuestro caso el bus SPI.
- DMAxPAD = SPI1BUF, se asigna la dirección del registro del buffer SPI
- DMAxCONbits.AMODE = 0, el modo de incremento de la dirección de registro indirecto se utiliza para mover bloques de datos al incrementar la dirección DPSRAM / RAM después de cada transferencia.
- DMAxCNT = XXXX, representa el número de solicitudes DMA que el canal debe atender antes de que la transferencia del bloque de datos se considere completa.

Ambas configuraciones del DMA con los valores mencionados anteriormente se muestran a continuación:

```
void DMA0_SetConfig(const BYTE * buff) {
```

```

//*****
//*****
// Setup DMA Channel 0 for:

//*****
DMA0CON = 0x6001;
/*
DMA0CONbits.CHEN = 0; //channel disabled
DMA0CONbits.SIZE = 1; //byte 1 - word 0
DMA0CONbits.DIR = 1; //register to peripheral
DMA0CONbits.HALF = 0; //full transfer interrupt
DMA0CONbits.NULLW = 0; //normal data
//DMA0CONbits.AMODE = 0b01; //register indirect without post-
increment
DMA0CONbits.AMODE = 0b00; //register indirect with post-increment
DMA0CONbits.MODE = 0b01; //one shot, no pingpong
*/

DMA0STAL = (unsigned int)buff;
DMA0STAH = 0x0000;

DMA0PAD = (volatile unsigned int)&SPI1BUF;

DMA0CNT = 512 - 1; //byte transfers
//DMA0CNT = 256 - 1; //byte transfers

DMA0REQ = 0x000A; //select SPI1 as DMA request source

IFS0bits.DMA0IF = 0; //clear DMA interrupt flag
IEC0bits.DMA0IE = 1; //enable DMA interrupt isr
}

void DMA1_SetConfig(void) {

//*****
//*****
// Setup DMA Channel 1 for:

//*****
DMA1CON = 0x4000;
/*
DMA1CONbits.CHEN = 0; //channel disabled
DMA1CONbits.SIZE = 0; //byte 1 - word 0
DMA1CONbits.DIR = 0; //peripheral to register
DMA1CONbits.HALF = 0; //full transfer interrupt
DMA1CONbits.NULLW = 0; //normal data
DMA1CONbits.AMODE = 0b00; //register indirect without post-
increment
DMA1CONbits.MODE = 0b00; //continuous, no pingpong
*/

DMA1STAL = (unsigned int)buff_DummyRx;
DMA1STAH = 0x0000;
DMA1PAD = (volatile unsigned int)&SPI1BUF;
DMA1CNT = 0; //byte transfers
DMA1REQ = 0x000A; //select SPI1 as DMA request source
IFS0bits.DMA1IF = 0; //clear DMA interrupt flag
IEC0bits.DMA1IE = 1; //enable DMA interrupt isr
}

```

En el caso de la función `DM0_SetConfig()`, se configura el acceso del bus SPI a través del DMA y como para un correcto envío de dicha información se hizo necesario la creación de la función `DMA1_SetConfig1()` en la cual se envía data ficticia al bus SPI. Esto garantiza el correcto envío de los datos del bus SPI a través de la DMA.

Para finalizar la función `InicializaDMA_SPI()` se encarga de cargar el buffer de 4kbytes creado en el microcontrolador y configurar las funciones mencionadas anteriormente.

```
void InicializaDMA_SPI(void) {
    DMA0_SetConfig(buff_CamCapture);
    DMA1_SetConfig();
}
```

8.1.7 Inicialización de la memoria SD y saber si se encuentra presente o no.

La función `Camara_SDCard()` se encarga de verificar si existe una memoria SD y que a su vez se encuentre funcionando de manera correcta para poder así almacenar los datos, de lo contrario dará error y la cámara nunca se activará, retornando así a su modo de ahorro de energía.

```
uint8_t Camara_SDCard(void) {
    DSTATUS estado;
    uint8_t aux;

    //Ensure card is still inserted
    ffs_10ms_timer = 10;

    //Wait for timer to expire
    while (ffs_10ms_timer);

    estado = disk_status(0);

    if (estado & STA_NODISK) {
        return 1; /* No card in the socket */
    }

    // INITIALIZE NEW MMC / SD CARD
    aux = 10;
    do {
        Result = f_mount(&FatFs, "", 1);
    } while ((Result != FR_OK) && (aux--));
```

```

        if(Result != FR_OK) {          /* Give a work area to the default
drive */
            return 1;
        }
        else{
            return 0;
        }
    }
}

```

8.1.8 Definición de variables estáticas y variables necesarias para las interrupciones.

```

FATFS FatFs;          /* FatFs work area needed for each volume */
FIL Fil;              /* File object needed for each open file */
FIL Fil2;             /* File object needed for each open file */
FIL Fil3;             /* File object needed for each open file */
FIL Fil4;
FRESULT Result;       /* Variable to store errors while using the SD
card*/
FRESULT Result1;
uint16_t tiemposMS[1024];
uint16_t tiemposMSIndex = 0;
uint32_t tiempoMSTotal = 0;
double tiempoMSRes = 0.00;

uint8_t salida[128];
char buff_name[5];
uint8_t flag_UartComando = 0;

uint8_t useDma = 0;

uint8_t buff_DummyRx[2];

// #define CAM_BUFF 512
// #define CAM_BUFF 1024
// #define CAM_BUFF 2048
#define CAM_BUFF 4096
uint8_t buff_CamCapture[CAM_BUFF]
__attribute__((far,section("samplebuffer"),address(0x2000)));

uint16_t index_CamCapture;
uint16_t bw;
uint16_t bw1;
uint16_t bw2;
uint16_t bw3;
uint8_t file_CamNombre[] = "N100000.jpg";
uint8_t file_sensor[] = "N1sensor.txt";
uint8_t file_metadata[] = "METADATA.dat";
/** @} */

/**
 * @brief Volatile variable for interruptions
 * @{
 */
volatile uint16_t ffs_10ms_timer;
volatile uint8_t flag_CaptureReady;
volatile uint8_t smCamara_Capture = 0;
volatile uint8_t Camara_FrameCount;
volatile uint8_t time_out = 0;

```

8.2 Configuración de periféricos necesarios para el control de la cámara

Se dispuso la creación de varias funciones necesarias para un óptimo control de la cámara, cada una de ellas necesario para una correcta toma de las imágenes proveniente del sensor CMOS.

- **Función Void InicializaIO**

Esta función se encarga de configurar los gpios del microcontrolador para comportarse según se haya establecido estos pueden funcionar como pines de entrada, salida o ambos a la vez.

También se configura los pines por los cuales se recibirá una señal de interrupción como es el caso de los pines VSYNC y ORDY. Por último, se asigna valores iniciales de “0” o “1” según lo solicitado por el fabricante en el caso de la memoria FIFO.

A continuación, se muestra la configuración detallada de los diversos pines digitales configurados para el propósito deseado:

```
void InicializaIO(void) {

//=====
//=====
//Test pin for Oscilloscope
//TRIS_PROBE_TEST = 0;
//PROBE_TEST = 0;

//LED and trigger buttons
TRIS_TEST_LED = 0; // output
TEST_LED = 0; // clear pin

TRIS_PIR = 1; // input
TRIS_FLASH = 0;
FLASH = 0;

TRIS_TEST_BUTTON = 1; //input
PU_TEST_BUTTON = 1; //enable pull up

//Camera Pins
TRIS_CAM_HREF = 1; //input
TRIS_CAM_VSYNC = 1; //input
TRIS_CAM_PWDN = 0; //output

//FIFO Memory pins B8-B15 are inputs
```

```

TRISB = TRISB | 0xFF00;
TRIS_MEM_RESET = 0; //output
TRIS_MEM_WEN = 0; //output
TRIS_MEM_RCLK = 0; //output
TRIS_MEM_RRST = 0; //output
TRIS_MEM_OE = 0; //output
TRIS_MEM_ORDY = 1; //input

//Assign VSYNC pin to INT1
RPINR0bits.INT1R = 57; //PPS INT1 --> RC9/RP57 para el VSYNC
IEC1bits.INT1IE = 0;
IFS1bits.INT1IF = 0;

//Assign ORDY pin to INT2
RPINR1bits.INT2R = 34; //PPS INT2 --> RB2/RPI34 para el ORDY
IEC1bits.INT2IE = 0;
IFS1bits.INT2IF = 0;

//Setup initial pin states for the Camera and FIFO memory
CAM_PWDN = 0; //power up

MEM_RESET = 0;
DelayMs(10);
MEM_RESET = 1;

MEM_RCLK = 1; //read port clock
MEM_RRST = 0; //read port reset state
MEM_OE = 1; //read port disable

MEM_WEN = 0; //no capture - verify that HREF actives high

// Enable Pins to communicate with ESP8266 Board
TRIS_TAKING_PHOTO = 0; //Output
}

```

- **Camara_GetVersion**

En esta función hacemos uso del bus I2C para verificar la versión de la cámara (0x5642) y de esta manera comprobar si está funcionando correctamente la comunicación I2C entre esta y el microcontrolador además de verificar si el modelo que se requiere es el que se ha adquirido.

```

static uint8_t Camara_GetVersion(void) {
    uint8_t aux1, aux2, res;

    res = HDByteReadI2C(CAM_ADDRESS, 0x300A, &aux1, 1);
    res += HDByteReadI2C(CAM_ADDRESS, 0x300B, &aux2, 1);
    if (res != 0) {
        return 1;
    }

    if (aux1 != 0x56) {
        return 1;
    }
    if (aux2 != 0x42) {
        return 1;
    }
    return 0;
}

```

```
}
```

- ***Camara_Set_VGAprev_YUVmode***

Esta función se encarga de configurar inicialmente la cámara, según la hoja de datos de aplicación esta función siempre tiene que estar presente antes de iniciar cualquier operación con la cámara. A su vez hace uso de la función `Camara_EnviaBytesI2C(Address,data)` en la cual haciendo uso del bus I2C se modifica los valores en la memoria interna de la cámara para que pueda comportarse como cámara, en esta etapa también se configura el tipo de imagen, el tamaño de los pixeles, reducción de ruido, etc.

```
void Camara_Set_VGAprev_YUVmode(void) {
    //13.1.1 VGA Preview
    //; for the setting , 24M Mlck input and 24M Plck output
    //;15fps YUV mode
    Camara_EnviaBytesI2C(0x3103, 0x93);
    Camara_EnviaBytesI2C(0x3008, 0x82);
    Camara_EnviaBytesI2C(0x3017, 0x7f);
    Camara_EnviaBytesI2C(0x3018, 0xfc);
    Camara_EnviaBytesI2C(0x3810, 0xc2);
    Camara_EnviaBytesI2C(0x3615, 0xf0);
    Camara_EnviaBytesI2C(0x3000, 0x00);
    Camara_EnviaBytesI2C(0x3001, 0x00);
    .
    .
    .
    .
}
```

El código completo se muestra en el Anexo 5.

- ***Función Camara_Set_QSXGAcapt_JPEGmode***

Permite la salida de la imagen en formato JPEG, requiere de haber ejecutado previamente la función `Camara_Set_VGAprev_YUVmode`. También se hace uso de la función `Camara_EnviaBytesI2C()`. Los registros configurados se muestran en el código a continuación:

```
void Camara_Set_QSXGAcapt_JPEGmode(void) {
    //13.1.4 QXGA Capture
    //; OV5642_QSXGA_YUV7.5 fps
    //; 24 MHz input clock, 24Mhz pclk
    //; jpeg mode 7.5fps
    Camara_EnviaBytesI2C(0x3503, 0x07);
    Camara_EnviaBytesI2C(0x3000, 0x00);
    Camara_EnviaBytesI2C(0x3001, 0x00);
    Camara_EnviaBytesI2C(0x3002, 0x00);
}
```

```

Camara_EnviaBytesI2C(0x3003, 0x00);
Camara_EnviaBytesI2C(0x3005, 0xff);
Camara_EnviaBytesI2C(0x3006, 0xff);
Camara_EnviaBytesI2C(0x3007, 0x3f);
Camara_EnviaBytesI2C(0x350c, 0x07);
Camara_EnviaBytesI2C(0x350d, 0xd0);
Camara_EnviaBytesI2C(0x3602, 0xe4);
Camara_EnviaBytesI2C(0x3612, 0xac);
Camara_EnviaBytesI2C(0x3613, 0x44);

Camara_EnviaBytesI2C(0x3621, 0x27);
Camara_EnviaBytesI2C(0x3622, 0x08);
Camara_EnviaBytesI2C(0x3623, 0x22);
Camara_EnviaBytesI2C(0x3604, 0x60);
Camara_EnviaBytesI2C(0x3705, 0xda);
Camara_EnviaBytesI2C(0x370a, 0x80);
Camara_EnviaBytesI2C(0x3801, 0x8a);
Camara_EnviaBytesI2C(0x3803, 0x0a);
Camara_EnviaBytesI2C(0x3804, 0x0a);
Camara_EnviaBytesI2C(0x3805, 0x20);
Camara_EnviaBytesI2C(0x3806, 0x07);
Camara_EnviaBytesI2C(0x3807, 0x98);
Camara_EnviaBytesI2C(0x3808, 0x0a);
Camara_EnviaBytesI2C(0x3809, 0x20);
Camara_EnviaBytesI2C(0x380a, 0x07);
Camara_EnviaBytesI2C(0x380b, 0x98);
Camara_EnviaBytesI2C(0x380c, 0x0c);
Camara_EnviaBytesI2C(0x380d, 0x80);
Camara_EnviaBytesI2C(0x380e, 0x07);
Camara_EnviaBytesI2C(0x380f, 0xd0);
Camara_EnviaBytesI2C(0x3810, 0xc2);
Camara_EnviaBytesI2C(0x3815, 0x44);
Camara_EnviaBytesI2C(0x3818, 0xc8);
Camara_EnviaBytesI2C(0x3824, 0x01);
Camara_EnviaBytesI2C(0x3827, 0x0a);
Camara_EnviaBytesI2C(0x3a00, 0x78);
Camara_EnviaBytesI2C(0x3a0d, 0x10);
Camara_EnviaBytesI2C(0x3a0e, 0x0d);
Camara_EnviaBytesI2C(0x3a10, 0x32);
Camara_EnviaBytesI2C(0x3a1b, 0x3c);
Camara_EnviaBytesI2C(0x3a1e, 0x32);
Camara_EnviaBytesI2C(0x3a11, 0x80);
Camara_EnviaBytesI2C(0x3a1f, 0x20);
Camara_EnviaBytesI2C(0x3a00, 0x78);
Camara_EnviaBytesI2C(0x460b, 0x35);
Camara_EnviaBytesI2C(0x471d, 0x00);
Camara_EnviaBytesI2C(0x4713, 0x02);
Camara_EnviaBytesI2C(0x471c, 0x50);
Camara_EnviaBytesI2C(0x5682, 0x0a);
Camara_EnviaBytesI2C(0x5683, 0x20);
Camara_EnviaBytesI2C(0x5686, 0x07);
Camara_EnviaBytesI2C(0x5687, 0x98);
Camara_EnviaBytesI2C(0x5001, 0x4f);
Camara_EnviaBytesI2C(0x589b, 0x00);
Camara_EnviaBytesI2C(0x589a, 0xc0);

//Low quality :
//write_i2c(0x4407, 0x08);
//Default quality:
//write_i2c(0x4407 ,0x04);
//High quality :
//Camara_EnviaBytesI2C(0x4407, 0x02);

//muy baja calidad
Camara_EnviaBytesI2C(0x4407, 0x04);
//

```

```

Camara_EnviaBytesI2C(0x589b, 0x00);
Camara_EnviaBytesI2C(0x589a, 0xc0);

Camara_EnviaBytesI2C(0x3002, 0x0c);
Camara_EnviaBytesI2C(0x3002, 0x00);
Camara_EnviaBytesI2C(0x3503, 0x00);

}

```

- ***Función Camara_SetSignalsPolarity***

Esta función establece si las señales vsync y clock se activaran en flaco de subida o de bajada. Las configuraciones de estas señales se muestran en el código a continuación:

```

static void Camara_SetSignalsPolarity(void) {

    uint8_t aux1;

    HDByteReadI2C(CAM_ADDRESS, 0x4740, &aux1, 1);
    aux1 = aux1 | 0x01; //active high VSYNC
    //aux1 = aux1 & ~(0x01); //active low VSYNC
    aux1 = aux1 | 0x020; //latch falling PCLK
    //aux1 = aux1 & ~(0x20); //latch rising PCLK
    HDByteWriteI2C(CAM_ADDRESS, 0x4740, aux1);
}

```

- ***Camara_SetProcessorClock***

Establece la frecuencia de salida de la cámara PCLK la cual está conectada al reloj de escritura de la memoria FIFO (WCKL). Como se puede observar esta frecuencia está dividida entre 1.5, sabiendo que el XCLK es de 7.5Mhz tenemos que el PCLK que a su vez está unida al clock de escritura en la memoria FIFO es de 5MHz.

```

static void Camara_SetProcessorClock(void) {

    //Don't use this setting because is too fast for the dsPIC
    //HDByteWriteI2C(CAM_ADDRESS, 0x3012, 0b000); //pclk = xclk / 1

    HDByteWriteI2C(CAM_ADDRESS, 0x3012, 0b001); //pclk = xclk / 1.5
}

```

8.3 Creación de un archivo

- ***Cámara_crearArchivo***

Esta función se encarga de crear un archivo en la memoria SD y le asigna el nombre pre-establecido anteriormente. Cabe mencionar que estos archivos son creados con la extensión jpg. Estas configuraciones se muestran en el código a continuación:

```
static uint8_t Camara_CrearArchivo(void) {
    //----- CREATE NEW FILE FOR WRITING -----
    Result = f_open(&Fil, (const TCHAR*)file_CamNombre, FA_WRITE |
FA_CREATE_ALWAYS);

    if (Result == FR_OK) {
        //----- FILE WAS SUCESSFULLY CREATED -----
#ifdef DEBUG
        printf("File created correctly\r\n");
#endif
        return 0;
    } else {
        //----- ERROR - THE FILE COULD NOT BE CREATED -----
#ifdef DEBUG
        printf("File could not be created\r\n");
#endif
        TEST_LED = 1;
        return 1;
    }
}
```

8.4 Configuración de rtc para asignar la hora a los archivos

Se usa el RTC DS3231 el cual opera usando el bus I2C para leer la información de la hora, minuto y segundo, así como el año, mes y día.

8.4.1 DS3231_Init

Inicializa el módulo RTC y verifica que se pueda leer la hora correctamente.

```
void DS3231_Init(void) {
    uint8_t ctReg = 0;
    static uint8_t hrReg = 0;

    ctReg = 0b00011100;
    if(LDByteWriteI2C2(DS3231_ADDRESS, DS3231_CONTROL_REG, ctReg))
        printf("Error while writing I2C bus\r\n");

    __delay_ms(10);

    if(LDByteReadI2C2(DS3231_ADDRESS, DS3231_HOUR_REG, &hrReg, 1))
        printf("Error while reading I2C bus\r\n");

    hrReg &= 0b10111111;
    __delay_ms(100);

    if(LDByteWriteI2C2(DS3231_ADDRESS, DS3231_HOUR_REG, hrReg))
```

```

        printf("Error while writing I2C bus\r\n");
    printf("RTC initialized correctly\r\n");
}

```

8.4.2 DS3231_DateHour

Lectura de la hora del RTC.

```

void DS3231_DateHour(uint8_t *regs) {
    if(LDByteReadI2C2(DS3231_ADDRESS, DS3231_SEC_REG, regs, 8))
        printf("Error while reading I2C bus\r\n");

    regs[0] = bcd2bin(regs[0]); // Seconds
    regs[1] = bcd2bin(regs[1]); // Minutes
    regs[2] = bcd2bin(regs[2] & ~0b11000000); // Hours
    regs[4] = bcd2bin(regs[4]); // Days
    regs[5] = bcd2bin(regs[5]); // Month
    regs[6] = bcd2bin(regs[6]); // Year
}

```

8.4.3 DS3231_EnableInt

Esta función se encarga de configurar el pin de interrupción del DS3231. Se indica la hora de la periodicidad de la interrupción, cada minuto, cada segundo o cada hora.

```

void DS3231_EnableInt(uint8_t periodicity) {
    uint8_t ctReg = 0;

    ctReg = 0b00011101;
    if(LDByteWriteI2C2(DS3231_ADDRESS, DS3231_CONTROL_REG, ctReg))
        printf("Error while writing I2C bus\r\n");

    switch(periodicity) {
        case EverySecond:
            LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1SEC_REG,
                0b10000000);
            LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1MIN_REG,
                0b10000000);
            LDByteWriteI2C2(DS3231_ADDRESS,
                DS3231_AL1HOUR_REG, 0b10000000);
            LDByteWriteI2C2(DS3231_ADDRESS,
                DS3231_AL1WDAY_REG, 0b10000000);
            break;
        case EveryMinute:
            LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1SEC_REG,
                0b00000000);
            LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1MIN_REG,
                0b10000000);
            LDByteWriteI2C2(DS3231_ADDRESS,
                DS3231_AL1HOUR_REG, 0b10000000);
            LDByteWriteI2C2(DS3231_ADDRESS,
                DS3231_AL1WDAY_REG, 0b10000000);
            break;
        case EveryHour:

```

```

        LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1SEC_REG,
0b00000000);
        LDByteWriteI2C2(DS3231_ADDRESS, DS3231_AL1MIN_REG,
0b00000000);
        LDByteWriteI2C2(DS3231_ADDRESS,
DS3231_AL1HOUR_REG,0b10000000);
        LDByteWriteI2C2(DS3231_ADDRESS,
DS3231_AL1WDAY_REG,0b10000000);
        break;
    }
}

```

8.4.4 DS3231_ClearInt

Limpiamos la interrupción después de usada para que el pin cambie su estado.

```

void DS3231_ClearInt(void) {
    uint8_t statusReg = 0;

    if(LDByteReadI2C2(DS3231_ADDRESS, DS3231_STATUS_REG, &statusReg,
1))
        printf("Error while reading I2C bus\r\n");
        statusReg &= 0b11111110;

    if(LDByteWriteI2C2(DS3231_ADDRESS, DS3231_STATUS_REG, statusReg))
        printf("Error while writing I2C bus\r\n");
}

```

8.5 Asignar la hora y fecha a la que fue tomada la imagen en el archivo .jpg creado

La librería de ELMCHAN usada en esta tesis, nos permite poder asignar el valor de la hora a la que se crean los archivos usando la función `get_fattime()` asignándole el valor de 0 a `_FS_NORTC`. Esta función debe retornar un valor valido del tiempo. Ya que disponemos del RTC DS3231 leeremos estos valores para retornarlos en esta función. Todos estos pasos se aprecian en el siguiente código:

```

DWORD get_fattime (void)
{

```

```

static uint8_t regs[8] = {0};
DS3231_DateHour(regs);

return ((DWORD)(regs[6] + 20) << 25)
       | ((DWORD)regs[5] << 21)
       | ((DWORD)regs[4] << 16)
       | ((DWORD)regs[2] << 11)
       | ((DWORD)regs[1] << 5)
       | ((DWORD)regs[0]>> 1);
}

```

8.6 Configuración para las interrupciones

8.6.1 *_INT1Interrupt*

Interrupcion_1 generada por el ping unido al VSYNC. Una vez configurado todos los periféricos y entradas y salidas digitales, al momento de activar la cámara esta esperara el flanco de bajada en el VSYNC para realizar el procedimiento de la captura de la imagen, por ello se asignó una interrupción al cambio de estado de esta y proceder inmediatamente con el guardado de los datos proveniente del sensor CMOS. En el siguiente diagrama de flujos se muestra el comportamiento del microcontrolador una vez detectada el flanco en el pin VSYNC.

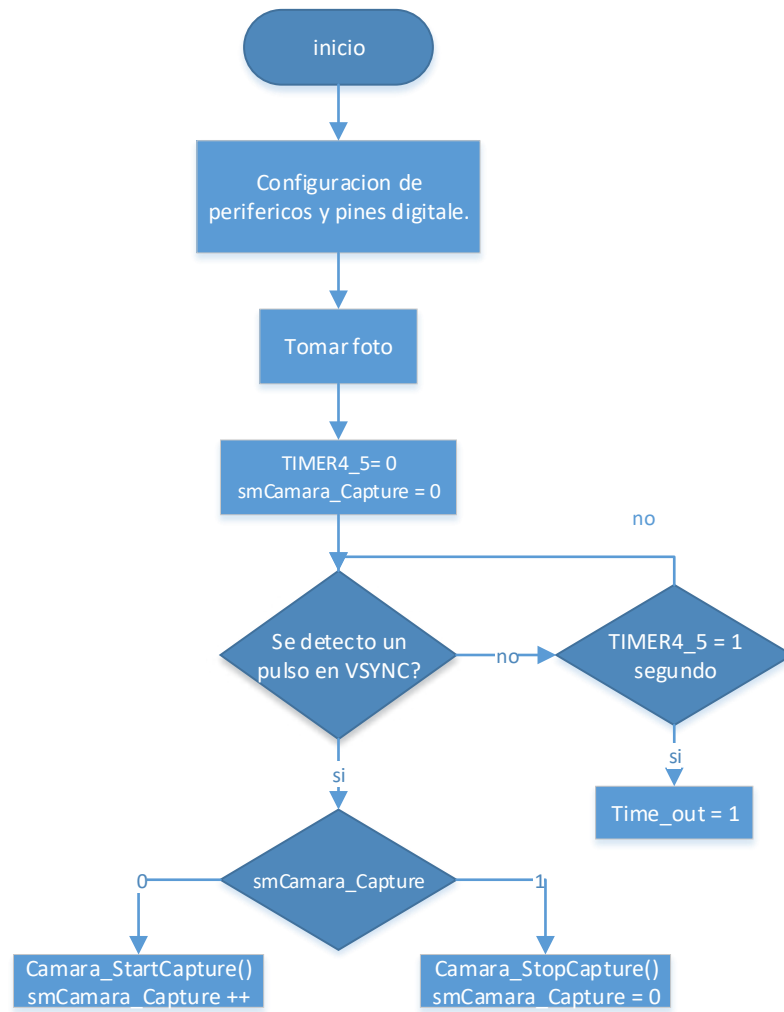


Fig. 9.- Diagrama de flujos de la interrupción generada por el VSYNC

Según lo mostrado en el diagrama de flujos el código se vería de la siguiente manera:

```

void __attribute__((__interrupt__, no_auto_psv)) _INT1Interrupt(void)
{
    IFS1bits.INT1IF = 0;
    //////////////////////////////////////
    //Reject Timeout of 5seg
    //    IFS1bits.T5IF = 0;
    //    IEC1bits.T5IE = 0; // Enable Timer5 interrupt
    //    T4CONbits.TON = 0; // Start 32-bit Timer
    //////////////////////////////////////

    switch (smCamara_Capture) {
        case 0:
            if(Camara_FrameCount-- == 0){
                Camara_StartCapture();
                flag_UartAck = 1;
                smCamara_Capture++;
            }
    }
}

```

```

        break;
    case 1:
        Camara_StopCapture();
        smCamara_Capture = 0;
        //camara_flagCaptureReady =
1;////////////////////////////////////
        break;
    default:
        smCamara_Capture = 0;
        break;
    }
}

```

8.6.2 *_INT2Interrupt*

Interrupción generada por el ORDY la cual nos avisara si hay información disponible para ser leída de la FIFO, así como si hay espacio disponible para seguir escribiendo en ella. Para evitar un bucle infinito durante este proceso se hace uso del timer de 32 bits configurado a 1 segundo de esta manera se sale de la rutina de esperar al ORDY. Una vez alcanzado este segundo de espera se habilita la interrupción en el timer y se habilita otro flag (Time_out) a 1. Este flag se encargará posteriormente de cancelar la captura de la foto. Estos pasos descritos se pueden apreciar en el siguiente diagrama de flujos.

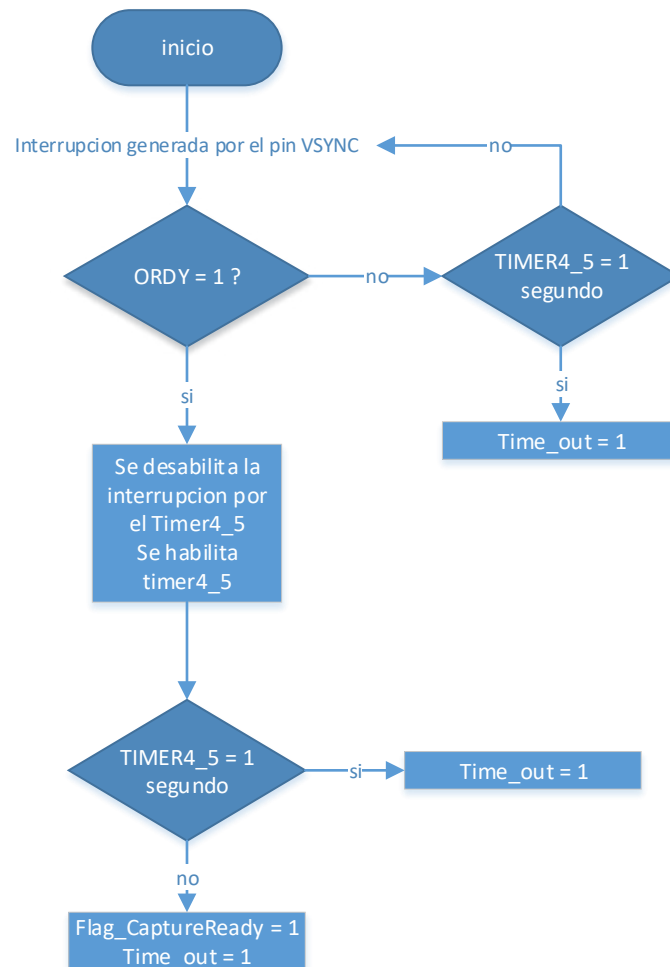


Fig. 10.- Diagrama de flujos de la interrupción generada por el ORDY

Una vez generada la interrupción por el ORDY se procederá a limpiar el flag de la interrupción, se finalizará el timer de 1 segundo y se habilitará el flag flag_CaptureReady que indicará que la memoria FIFO se encuentra lista para ser leída por el microcontrolador como se muestra en el código siguiente:

```

void __attribute__((__interrupt__, no_auto_psv)) _INT2Interrupt(void)
{
    IFS1bits.INT2IF = 0;

    IFS1bits.T5IF = 0;
    IEC1bits.T5IE = 0; // Disable Timer5 interrupt
    T4CONbits.TON = 0; // N-Start 32-bit Timer

    IEC1bits.INT2IE = 0;
    flag_CaptureReady = 1;
    time_out = 1;
}

```

8.6.3 *_T1Interrupt*

Interrupción generada por el TIMER 1 la cual la usamos para establecer un pequeño tiempo por cada escritura en la SD.

```
void __attribute__((__interrupt__, no_auto_psv)) _T1Interrupt(void) {
    IFS0bits.T1IF = 0; //Clear Timer1 interrupt flag

    //----- FAT FILING SYSTEM DRIVER TIMER -----
    disk_timerproc();
    //----- FAT FILING SYSTEM DRIVER TIMER -----
    if (ffs_10ms_timer) {
        ffs_10ms_timer--;
    }
}
```

8.6.4 *_T5Interrupt*

Interrupción generada por el TIMER4_5 de 32 bits que usamos como time_out y así evitar que se quede en un ciclo infinito.

```
void __attribute__((__interrupt__, no_auto_psv))
_T5Interrupt(void)
{
    /* Interrupt Service Routine code goes here */
    IFS1bits.T5IF = 0; //Clear Timer5 interrupt flag
    printf("Se reinicia el timer \r\n");
    time_out = 1;
}
```

8.6.5 *PIR_IntConfig*

Interrupción generada por el sensor de movimiento el cual tiene la función de despertar al dsPIC de la función de ahorro de energía para así tomar la imagen del animal que paso por el lugar.

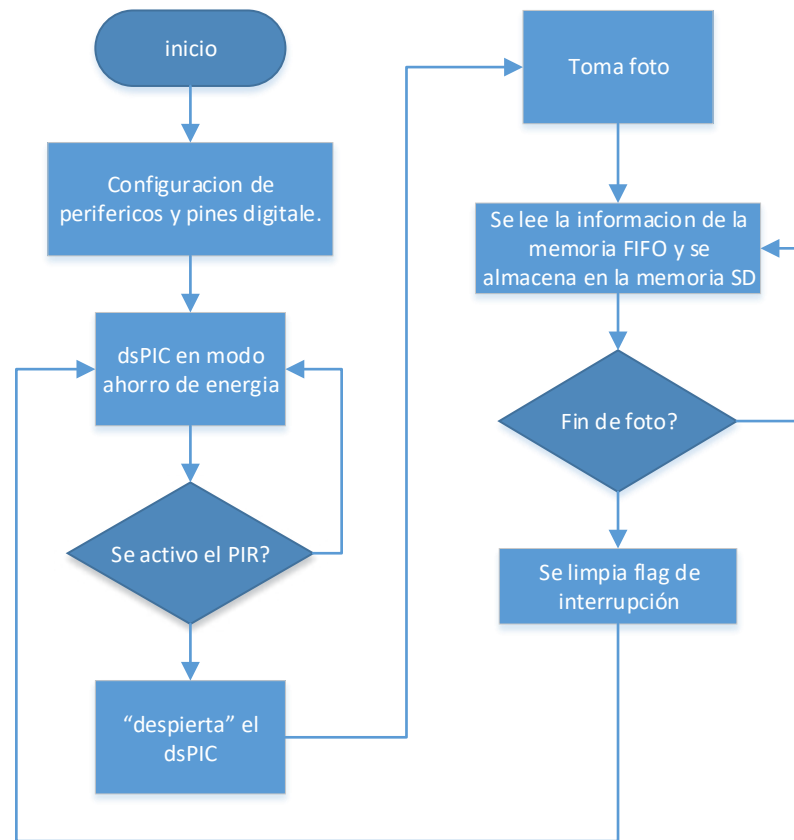


Fig. 11.- Diagrama de flujos de la interrupción generada por el PIR

```

void PIR_IntConfig(void)
{
    INTCON2bits.INT0EP = 0; // 0: interrupt on positive edge, 1:
    // interrupt on negative edge.
    IPC0bits.INT0IP = 7; // priority
    IFS0bits.INT0IF = 0; // reset INT0 interrupt flag
    IEC0bits.INT0IE = 1; // enable INT0 interrupt service routine
}

```

8.7 Ahorro de energía e inicialización y configuración

El dsPIC tiene la posibilidad de usar un modo de máximo ahorro energético en el cual se desactiva la frecuencia del sistema. En este modo todos los periféricos que requieren de un reloj para su funcionamiento dejarán de funcionar. Esto permite que mientras no haya ningún animal que pueda activar el sensor de movimiento el sistema ingrese a un modo donde su consumo sea lo más mínimo posible. Para esto no solo se debe manipular el macro ya definido para lograr ese bajo consumo energético sino También se tiene que

configurar todos los pines digitales del microcontrolador a 1 lógico haciendo uso de un pull-up interno del microcontrolador. Otro registro a modificar es el PMDX el cual deshabilitará los periféricos. Estos pasos descritos se pueden apreciar en el siguiente código:

```
void AhorroEnergia(void)
{
    TRISA = 0xFFFF;
    TRISB = 0xFFD7;
    TRISC = 0xFFFE;
    //TRISCbits.TRISC5 = 0;
    CNPDAbits.CNPDA0 = 1;
    CNPDAbits.CNPDA1 = 1;
    CNPDAbits.CNPDA2 = 1;
    CNPDAbits.CNPDA3 = 1;
    CNPDAbits.CNPDA4 = 1;
    CNPDAbits.CNPDA7 = 1;
    CNPDAbits.CNPDA8 = 1;
    CNPDAbits.CNPDA9 = 1;
    CNPDAbits.CNPDA10 = 1;
    CNPDBbits.CNPDB0 = 1;
    CNPDBbits.CNPDB2 = 1;
    CNPDBbits.CNPDB3 = 1;
    CNPDBbits.CNPDB4 = 1;
    CNPDBbits.CNPDB5 = 1;
    CNPDBbits.CNPDB6 = 1;
    //CNPDBbits.CNPDB7 = 1;
    CNPDBbits.CNPDB8 = 1;
    CNPDBbits.CNPDB10 = 1;
    CNPDBbits.CNPDB11 = 1;
    CNPDBbits.CNPDB12 = 1;
    CNPDBbits.CNPDB13 = 1;
    CNPDBbits.CNPDB14 = 1;
    CNPDBbits.CNPDB15 = 1;
    //    CNPDCbits.CNPDC0 = 1;
    CNPDCbits.CNPDC1 = 1;
    CNPDCbits.CNPDC2 = 1;
    CNPDCbits.CNPDC3 = 1;
    CNPDCbits.CNPDC4 = 1;
    CNPDCbits.CNPDC5 = 1;
    CNPDCbits.CNPDC6 = 1;
    CNPDCbits.CNPDC7 = 1;
    CNPDCbits.CNPDC8 = 1;
    CNPDCbits.CNPDC9 = 1;
    PMD1 = 0xFEFB; // Disables all Peripheral Interfaces
    PMD2 = 0xF0F;
    PMD3bits.CMPMD = 1;
    PMD3bits.CRCMD = 1;
    PMD3bits.I2C2MD = 1;
    PMD4bits.CTMUMD = 1;
    PMD4bits.REFOMD = 1;
    PMD7 = 1;
    CAM_PWDN = 1;
}
```

Una vez el animal pase en frente de la cámara se tiene que deshabilitar este modo deshabilitando el pull-up interno de todos los pines digitales y modificando los valores del registro PMD a 0.

```

void NoAhorroEnergia(void)
{
    CNPDAbits.CNPDA0 = 0;
    CNPDAbits.CNPDA1 = 0;
    CNPDAbits.CNPDA2 = 0;
    CNPDAbits.CNPDA3 = 0;
    CNPDAbits.CNPDA4 = 0;
    CNPDAbits.CNPDA7 = 0;
    CNPDAbits.CNPDA8 = 0;
    CNPDAbits.CNPDA9 = 0;
    CNPDAbits.CNPDA10 = 0;
    CNPDBbits.CNPDB0 = 0;
    CNPDBbits.CNPDB2 = 0;
    CNPDBbits.CNPDB3 = 0;
    CNPDBbits.CNPDB4 = 0;
    CNPDBbits.CNPDB5 = 0;
    CNPDBbits.CNPDB6 = 0;
    //CNPDBbits.CNPDB7 = 0;
    CNPDBbits.CNPDB8 = 0;
    CNPDBbits.CNPDB10 = 0;
    CNPDBbits.CNPDB11 = 0;
    CNPDBbits.CNPDB12 = 0;
    CNPDBbits.CNPDB13 = 0;
    CNPDBbits.CNPDB14 = 0;
    CNPDBbits.CNPDB15 = 0;
    CNPDCbits.CNPDC0 = 0;
    CNPDCbits.CNPDC1 = 0;
    CNPDCbits.CNPDC2 = 0;
    CNPDCbits.CNPDC3 = 0;
    CNPDCbits.CNPDC4 = 0;
    CNPDCbits.CNPDC5 = 0;
    CNPDCbits.CNPDC6 = 0;
    CNPDCbits.CNPDC7 = 0;
    CNPDCbits.CNPDC8 = 0;
    CNPDCbits.CNPDC9 = 0;
    PMD1 = 0x0000;
    PMD2 = 0x0000;
    PMD3bits.CMPMD = 0;
    PMD3bits.CRCMD = 0;
    PMD3bits.I2C2MD = 0;
    PMD4bits.CTMUMD = 0;
    PMD4bits.REFOMD = 0;
    PMD7 = 0;
    TRIS_TAKING_PHOTO = 0;
}

```

Todo el proceso del control de la captura de la imagen y el modo de ahorro de energía se puede apreciar de mejor manera en el siguiente diagrama de bloques.

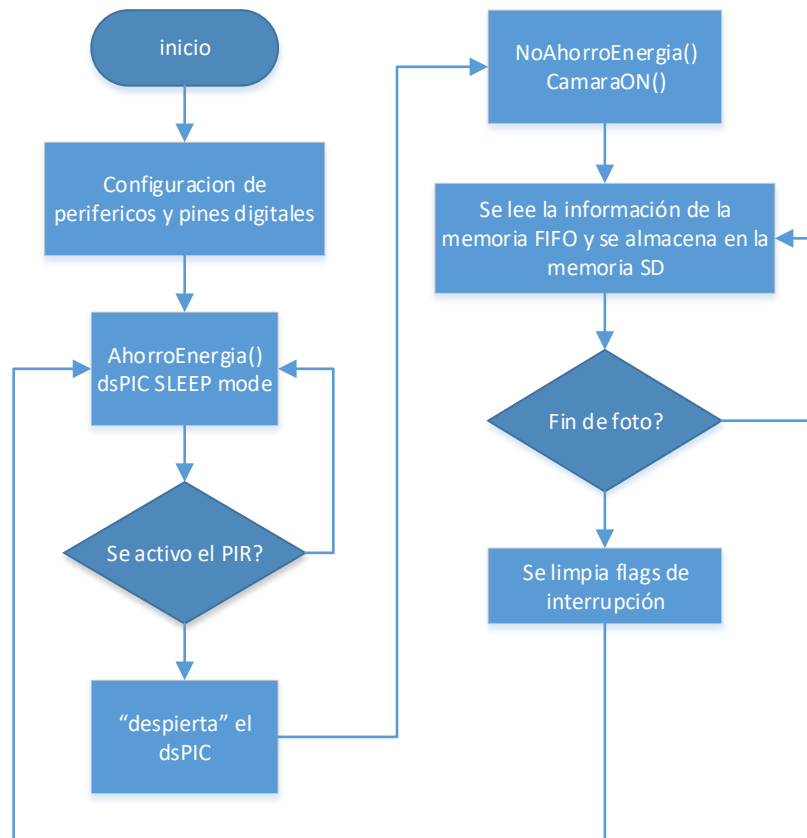


Fig. 12.- Diagrama de flujos del comportamiento del módulo para un bajo consumo energético.

Para que se pueda tomar una imagen se utiliza la función `CamaraON()`; esta función retornara 1 si se superó el tiempo de espera de la SD y 0 de no ser el caso, procediendo a la ingresar a la función `Camara_TomarImagen()` como se puede apreciar en el código siguiente:

```

uint8_t CamaraON()
{
    int tiempo_espera = 5000;
    Camara_Reset();
    InicializaREFOCLK();
    InicializaTIMER1();
    InicializaTIMER2();
    InicializaSPI1();
    InicializaDMA_SPI();
    InicializaI2C();
    InicializaUART1();
    InicializaIO();

    DelayMs(100);

    INTCON2bits.GIE = 1; //enable all interrupts

#ifdef DEBUG
    printf("File: %s LINE: %d Entrando al While\r\n", __FILE__,
    __LINE__);

```

```

#endif
    while (!Camara_GetVersion() && (tiempo_espera > 0)) {
        Camara_Reset();
        tiempo_espera--;
    }
#ifdef DEBUG
    printf("File: %s LINE: %d Saliendo del While\r\n", __FILE__,
__LINE__);
#endif

    if (tiempo_espera == 0){
#ifdef DEBUG
        printf("Camera not found\r\n");
#endif
        return 1;
    }
    Camara_SetConfig();

    tiempo_espera = 3;
#ifdef DEBUG
    printf("File: %s LINE: %d Entrando al While\r\n", __FILE__,
__LINE__);
#endif
    while (Camara_SDCard() && (tiempo_espera > 0)){
        tiempo_espera--;
    }
#ifdef DEBUG
    printf("File: %s LINE: %d Saliendo del While\r\n", __FILE__,
__LINE__);
#endif

    if (tiempo_espera == 0){
#ifdef DEBUG
        printf("SD card not found\r\n");
#endif
        return 1;
    }
    return 0;
}

```

8.8 Funciones camara_tomarimagen y camara_guardarimagen:

Sirven para generar la rutina de toma de imagen y el proceso de guardar la imagen.

8.8.1 Camara_TomarImagen:

Esta función se encarga de conmutar un PIN (TRISC6) digital del dsPIC el cual está unido al pin RCLK de la memoria FIFO. Una vez se genere el pulso en el pin VSYNC este generará una interrupción que se encargara de empezar la transferencia de información de la imagen hacia la memoria FIFO. Una vez haya suficientes datos en la memoria el pin ORDY cambiará de estado para

indicarnos que ya se puede realizar la lectura. En este proceso habilitamos las interrupciones generadas por el VSYNC y ORDY y el timer de 32 bits que sirve como timeout hasta encontrar un pulso en el VSYNC de lo contrario saldrá de esta rutina y volverá al modo ahorro energético.

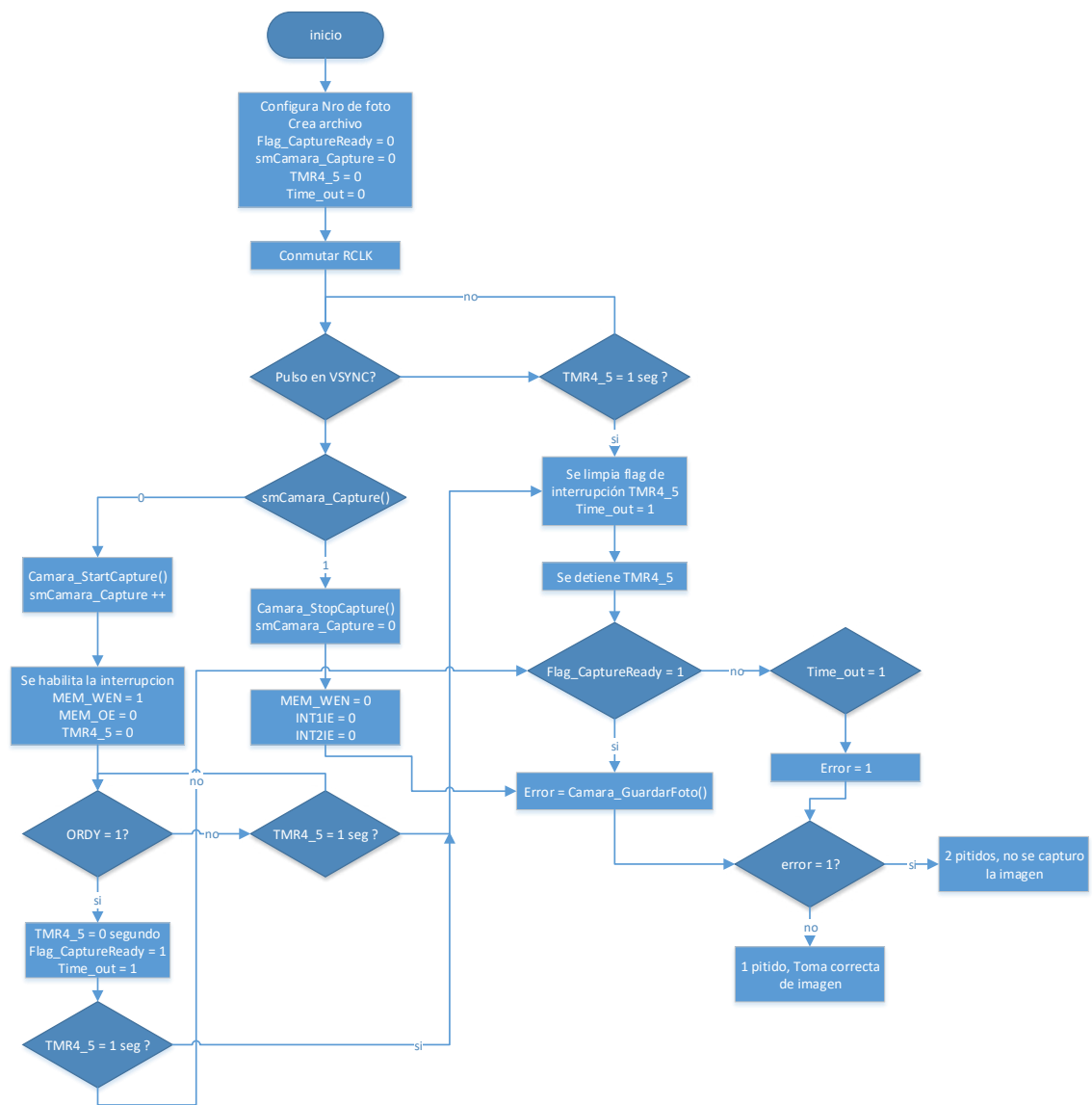


Fig. 13.- Diagrama de flujo de la función Camara_TomarImagen()

El código empleado para realizar lo mostrado en el diagrama de flujos es el siguiente:

```

uint8_t Camara_TomarImagen(void) {
    uint8_t error;

```

```

    Camara_SetFileName();
    error = Camara_CrearArchivo();
    FLASH = 0;
    if (error == 0) {
#ifdef DEBUG
        Camara_EnviaStringUART1((uint8_t *)"@01,-----
        -----!\r\n");
        sprintf((char *)salida,"@01,Taking Photo: %s!\r\n",
file_CamNombre);
        Camara_EnviaStringUART1(salida);
#endif

        TEST_LED = 1;
        DelayMs(50);
        TEST_LED = 0;

        smCamara_Capture = 0;
        flag_CaptureReady = 0;
        time_out = 0;
        //CamaraCount = 3;//wait 3 frame
        //CamaraCount = 2;//wait 2 frame
        //CamaraCount = 1;//wait 1 frame
        Camara_FrameCount = 0;//inmediatly

        //Interrupt for the VSYNC
        INTCON2bits.INT1EP = 1;//flanco de bajada
        IFS1bits.INT1IF = 0;
y        IEC1bits.INT1IE = 1;

        //Timeout de VSYNC por 1 segundo
        InicializaTIMER4_5();

#ifdef DEBUG
        printf("File: %s Line: %d Toggleando\r\n", __FILE__,
__LINE__);
#endif

        do {
            asm("BTG LATC,#6");
        } while (time_out == 0);

        IEC1bits.T5IE = 0; // Disable Timer5 interrupt
        T4CONbits.TON = 0; // Stop 32-bit Timer

#ifdef DEBUG
        printf("File: %s Line: %d Time Out Acabado\r\n", __FILE__,
__LINE__);
#endif

        if (flag_CaptureReady) {
            error = Camara_GuardarImagen();

        } else if (time_out == 1) {
#ifdef DEBUG
            printf("no detecte el vsync \r\n");
#endif
            error = 1;
        }
    }

    //Send a beep if picture reading was succesfull
    if (error == 0) {
        TEST_LED = 1;
        DelayMs(500);
        TEST_LED = 0;
        //Send two tones if error while reading
    } else {

```

```

        TEST_LED = 1;
        DelayMs(250);
        TEST_LED = 0;
        DelayMs(250);
        TEST_LED = 1;
        DelayMs(250);
        TEST_LED = 0;
    }

    T2CONbits.TON = 0;
    TMR2 = 0;
    T2CONbits.TON = 1;
    MetaData();
#ifdef DEBUG
    printf("File: %s LINE: %d tiempo de creacion de metadata
%d\r\n",__FILE__, __LINE__, TMR2);
#endif
    TMR2 = 0;
    T2CONbits.TON = 0;

    T4CONbits.T32 = 1;
    Temp_Init();
#ifdef DEBUG
    printf("TMR4 %x\r\n",TMR4);
    printf("TMR5 %x\r\n",TMR5);
#endif
    T4CONbits.T32 = 0;
    Camara_ResetApp();
    return error;//1: hubo error, 0: exito
}

```

8.8.2 Camara_GuardarImagen

Esta función se encarga de leer el bus paralelo de 8 bits del dsPIC, estos son almacenados temporalmente en un buffer de 4KBytes. Gracias al acceso directo a memoria la transferencia del buffer hacia el SPI se hace por medio del acceso directo a memoria lo cual nos ahorra mucho tiempo. Para obtener una mayor rapidez a la hora de escribir los datos en la memoria SD y por ende un rápido guardado de la toda la información de la imagen en la memoria, se decide leer los 8 bits (bus paralelo) y almacenarlos periódicamente en un buffer de 4KB. Una vez alcanzado esta capacidad se efectúa la escritura en la memoria SD de los primeros 4KB de la imagen, como se recordará en el caso de imágenes de 5mp el tamaño puede superar el mega byte. Debido a la configuración del DMA (acceso directo a memoria) en la que se creó un canal de comunicación directo entre esta y el bus SPI la escritura en la memoria SD se hizo de una manera más rápido.

La librería del EL-CHAN contiene los parámetros de configuración necesarios para la correcta creación y tratamiento de los archivos, al momento de escribir los datos en la memoria SD usa el modo FCLK_FAST la cual configura el bus SPI para trabajar a su máxima velocidad mediante la siguiente definición:

```
#define      SPI_BUS_SET_TO_FULL_SPEED_SD      SPI1STATbits.SPIEN=0;
SPI1CON1bits.PPRE1=1;      SPI1CON1bits.PPRE0=1;      SPI1CON1bits.SPRE2=1;
SPI1CON1bits.SPRE1=0; SPI1CON1bits.SPRE0=1; SPI1STATbits.SPIEN=1;
```

Donde:

SPRE<2:0> = 5, Pre escalador secundario 3:1

PPRE<1:0> = 3, Pre escalador primario 1:1

Usando la fórmula (5) para poder conocer la frecuencia a la que trabaja el clock del bus SPI, tenemos:

$$SCK1 = \frac{59881250}{3 * 1}$$

$$SCK1 = 19.96 \text{ MHz}$$

El fabricante de la librería, Elm-chan, menciona que para el caso de lecturas y escrituras en una memoria SD usando el bus SPI, el máximo valor de frecuencia que puede tomar el clock es de 25Mhz, tomando esto en cuenta en la configuración del clock del SPI usando los pre escaladores tenemos un valor de aproximadamente 20Mhz, de esta manera estamos en el rango establecido por el fabricante.

Todo el proceso del guardado de la imagen se muestra en el siguiente diagrama de flujos:

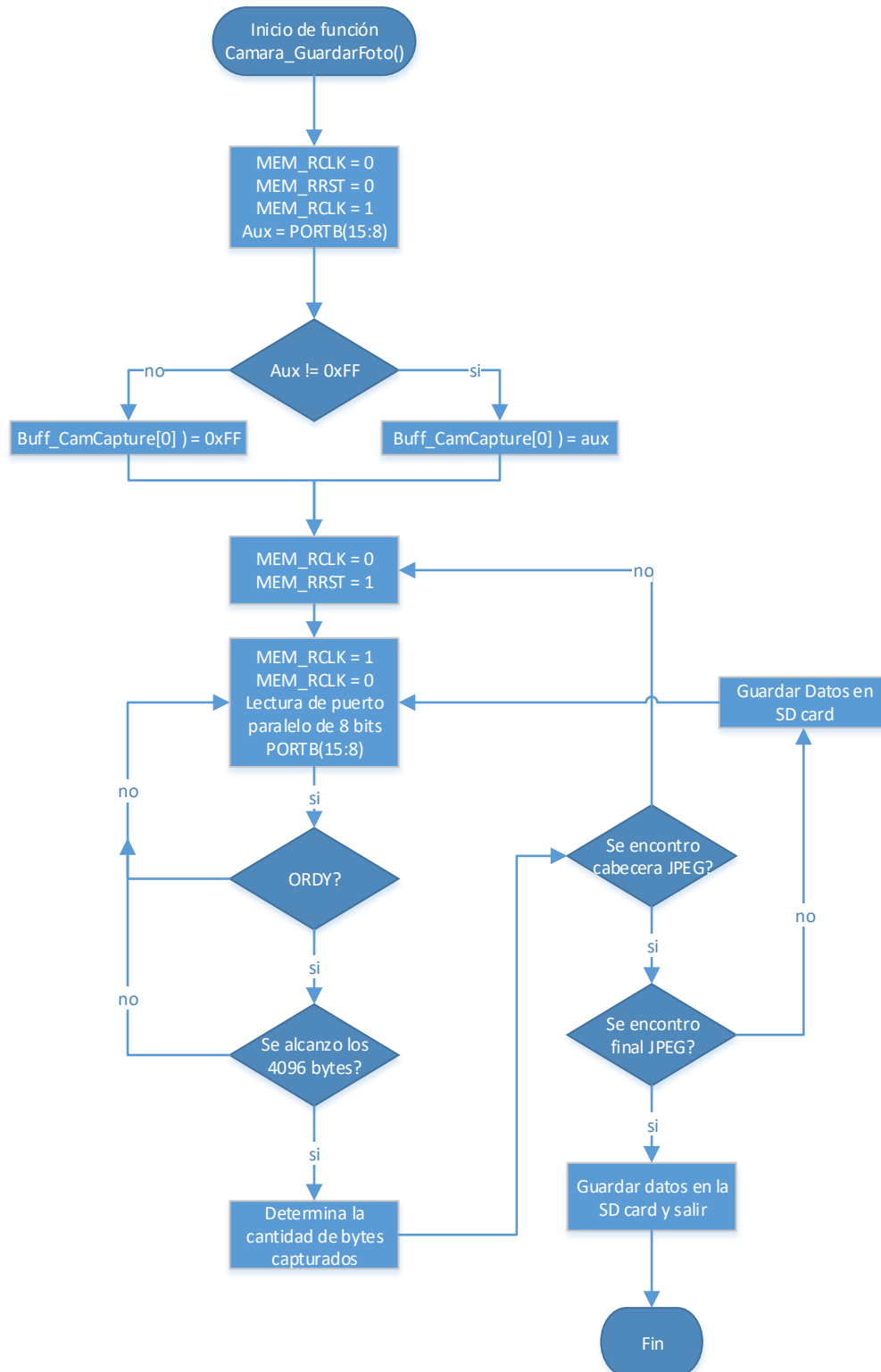


Fig. 14.- Diagrama de flujos de la lectura de la memoria FIFO y la creación del archivo

Lo descrito en el diagrama de flujos se muestra en el siguiente código:

```
inline static uint8_t Camara_GuardarImagen(void) {

    uint8_t aux;
    uint16_t bw;
    uint8_t inicio_CamCapture;
    uint8_t fin_CamCapture;
    uint8_t firstCapture;
    uint16_t cantCapture;
    uint8_t timeoutinicioCapture = 0;
    uint16_t timeoutfinCapture = 0;
    uint32_t totalCapture = 0;

#ifdef DEBUG
    printf("File: %s Line: %d @01,JPEG started!\r\n", __FILE__, __LINE__);
#endif
    T1CONbits.TON = 0; // Stop Timer
    tiempoMSTotal = 0;
    tiemposMSIndex = 0;
    int p=0;
    for (p=0; p < (sizeof(tiemposMS)/sizeof(tiemposMS[0])); p++) {
        tiemposMS[p] = 0;
    }
    T2CONbits.TON = 0;
    TMR2 = 0;
    T2CONbits.TON = 1;
    asm("RESET_CAMARA:");

    MEM_RCLK = 0;
    MEM_RRST = 0;
    MEM_RCLK = 1;
    DelayMs(10);
    aux = (uint8_t)(PORTB >> 8);

    if(aux != 0xFF) {
        buff_CamCapture[0] = 0xFF;
    } else {
        buff_CamCapture[0] = aux;
    }
};

MEM_RCLK = 0;
MEM_RRST = 1;

inicio_CamCapture = 0;
fin_CamCapture = 0;
firstCapture = 1;
bw = 0;
useDma = 0;
cantCapture = 0;

asm("MOV #0x2001, W7");
asm("GOTO INICIO");

asm("ESPERAR:");
DelayMs(5);
#ifdef DEBUG
    printf("File: %s LINE: %d Entrando al While\r\n", __FILE__, __LINE__);
#endif
asm("GOTO CHECKLISTO");

do{
    asm("INICIO: MOV PORTB, W1");
    asm("LSR W1, #8, W1");
    asm("MOV.B W1, [W7++]");

    MEM_RCLK = 1;
```

```

MEM_RCLK = 0;

asm("BTSC PORTB, #2");//ORDY
asm("GOTO ESPERAR");
asm("CHECKLISTO: BTSS W7,#12");//para buff size 4096
//asm("BTSS W7,#11");//para buff size 2048
//asm("BTSS W7,#10");//para buff size 1024
asm("GOTO INICIO");

//=====
//SECCION DONDE GUARDAMOS LOS DATOS CAPTURADOS EN LA SD CARD
//asm("GUARDAR:");
T2CONbits.TON = 0;
tiemposMS[tiemposMSIndex++] = TMR2;
TMR2 = 0;
T2CONbits.TON = 1;

//=====
//DETERMINA LA CANTIDAD DE BYTES A GUARDAR
asm("MOV #0x2000, W1");
asm("SUBR W1, W7, W1");
asm("MOV W1, [W14+12]");

//=====
//VERIFICAMOS CABECERA DEL JPEG
if (firstCapture) {
    firstCapture = 0;
    if(++timeoutinicioCapture >= 200) { //50 milisegundos
        break;
    }
    //asm("CHECKHEADER: MOV #0xE0FF,W0");
    asm("MOV #0xE0FF,W0");
    asm("MOV #0x2002,W1");
    asm("SUBR W0, [W1],W0");
    asm("BRA NZ, RESET_CAMARA");
    inicio_CamCapture = 1;
}

//=====
//VERIFICAMOS FINAL DEL JPEG
asm("MOV #0x2000, W2"); // w2 tiene la dirección de memoria 0x2000
asm("MOV [W14+12], W0"); // Movemos a W0 los bytes escritos en el
buffer
asm("ADD W2, W0, W0"); // W0 tiene la direccion de memoria final

asm("CHECKFINAL: CLR W1"); // limpiamos w1
asm("MOV.B [W2++], W1"); // w1 = 0x2001
asm("SUBR W2, W0, W3"); // w3 = w0 - w2
asm("BRA Z, FIN_CHECKFINAL"); // si la resta es 0 salta a
FIN_CHECKFINAL

asm("MOV.B [W2], W3"); // w3 = contenido del buffer en la direccion
w2
asm("SL W3, #8, W3"); //
asm("IOR W3, W1, W1");

asm("MOV #0xD9FF, W3");
asm("SUBR W1, W3, W3");
asm("BRA NZ, CHECKFINAL");

fin_CamCapture = 1;

asm("FIN_CHECKFINAL: MOV #0x2000, W0");
asm("SUBR W0, W2, W1");
asm("MOV W1, [W14+14]");

```

```

index_CamCapture = bw + 1;

//ESCRIBIR EN LA SD CARD
T1CONbits.TON = 1; // Start Timer 1

if (fin_CamCapture) {
    /* Write data to the file */
    useDma = 1;
    f_write(&Fil, buff_CamCapture, index_CamCapture, &bw);
    useDma = 0;
    if ((Result != FR_OK) || (bw != index_CamCapture)) {
        break;
    }
    totalCapture += cantCapture;
    break;
} else {
    /* Write data to the file */
    useDma = 1;
    Result = f_write(&Fil, buff_CamCapture, cantCapture, &bw);
    useDma = 0;
    if ((Result != FR_OK) || (bw != cantCapture)) {
        inicio_CamCapture = 0;
        break;
    }
    totalCapture += cantCapture;
}

if(++timeoutfinCapture >= 2000) { // segundos
    fin_CamCapture = 0;
    break;
}

T1CONbits.TON = 0; // Stop Timer
T2CONbits.TON = 0;
tiemposMS[tiemposMSIndex++] = TMR2;
TMR2 = 0;
T2CONbits.TON = 1;

asm("MOV #0x2000, W7");

} while(1);
#ifdef DEBUG
    printf("File: %s LINE: %d Saliendo del While\r\n", __FILE__, __LINE__);
#endif

//asm("FINALIZAR:");
T1CONbits.TON = 0; // Stop Timer
T2CONbits.TON = 0;
tiemposMS[tiemposMSIndex++] = TMR2;
//Se concluyo con toda la carga del FIFO en la SD card
// PROBE_TEST = 0;
//Deshabilitamos el puerto de lectura del FIFO
MEM_OE = 1;

//----- CLOSE THE FILE -----
Result = f_close(&Fil);

if ((Result == FR_OK) && inicio_CamCapture && fin_CamCapture) {
    //Camara_EnviaStringUART1((uint8_t *)"@01,JPEG finished OK!\r\n");
#ifdef DEBUG
    printf("@01,JPEG finished OK!\r\n");
#endif

    for (bw = 0; bw < tiemposMSIndex; bw++) {
        tiempoMSTotal = tiempoMSTotal + (uint32_t)tiemposMS[bw];
    }
}

```

```

        tiempoMSRes = (float)tiempoMSTotal * 0.00106878;
#ifdef DEBUG
        printf("@01,JPEG time elapsed: %.4f mseg!\r\n", tiempoMSRes);
#endif
        //sprintf((char *)salida, "@01,JPEG time elapsed: %.4f mseg!\r\n",
        tiempoMSRes);

        //Camara_EnviaStringUART1(salida);
        //Camara_EnviaStringUART1((uint8_t *)"@01,-----
        -----!\r\n");
#ifdef DEBUG
        printf("@01,-----
        -----!\r\n");
#endif

        return 0;//no error
    }
    Camara_Reset();
#ifdef DEBUG
    //Camara_EnviaStringUART1((uint8_t *)"@01,JPEG finished ERROR!\r\n");
    printf("@01,JPEG finished ERROR!\r\n");
    //Camara_EnviaStringUART1((uint8_t *)"@01,JPEG time elapsed: !\r\n");
    printf("@01,JPEG time elapsed: !\r\n");
    //Camara_EnviaStringUART1((uint8_t *)"@01,-----
    -----!\r\n");
    printf("@01,-----
    -----!\r\n");
#endif

    return 1;//theres is an error
}

```

8.9 Control de medición de temperatura, humedad

Se usó el integrado HDC1000 para la medición de temperatura y humedad, la cual opera bajo el protocolo I2C para poder configurar y establecer el tipo de información que uno requiere, pudiendo solo leer temperatura o solo leer humedad, en este caso dicha información será leída y guardada en un archivo de texto posterior a la toma de la imagen.

8.9.1 Función *HDC1000_ReadHum*:

Esta función se encarga de retornar el valor de la humedad, aquí podemos observar el uso del bus I2C para la recepción de la información. Este valor que nos retorna tiene que ser convertido usando una fórmula para una correcta lectura.

A continuación, se muestra el código correspondiente para efectuar la lectura de la humedad.

```
float HDC1000_ReadHum() {
    static uint8_t confReg[2] = {0}, Reg_Value[2] = {0};
    uint16_t Hum = 0;

    //Set Temperature or Humidity Acquisition and High Resolution
    confReg[1] = 0; //Must be '0' according to datasheet
    confReg[0] = (0 <<
HDC1000_MODE_BIT) | (0 << HDC1000_TEMP_RES_BIT) | (0 << HDC1000_HUM_RES_BIT);

    if(LDPageWriteI2C2(HDC1000_ADDRESS, HDC1000_CONFIGURATION_REG, confReg,
2))
        printf("Error while writing I2C bus\r\n");

    if(LDByteReadI2C2(HDC1000_ADDRESS, HDC1000_HUMIDITY_REG, Reg_Value, 2))
        printf("Error while reading I2C bus\r\n");

    Hum = ( Reg_Value[0] << 8 ) | Reg_Value[1];
    return ( (float)Hum / 65536 ) * 100;
}
```

8.9.2 Función *HDC1000_ReadTemp*:

Esta función se encarga de retornar el valor de la temperatura, aquí podemos observar el uso del bus I2C para la recepción de la información. Este valor que nos retorna tiene que ser convertido usando una fórmula para una correcta lectura.

A continuación, se muestra el código correspondiente para efectuar la lectura de la temperatura.

```
float HDC1000_ReadTemp(){
    static uint8_t confReg[2] = {0}, Reg_Value[2] = {0};
    uint16_t Temp = 0;

    //Set Temperature or Humidity Acquisition and High Resolution
    confReg[1] = 0; //Must be '0' according to datasheet
    confReg[0] = (0 <<
HDC1000_MODE_BIT) | (0 << HDC1000_TEMP_RES_BIT) | (0 << HDC1000_HUM_RES_BIT);

    if(LDPageWriteI2C2(HDC1000_ADDRESS, HDC1000_CONFIGURATION_REG, confReg,
2))
        printf("Error while writing I2C bus\r\n");

    if(LDByteReadI2C2(HDC1000_ADDRESS, HDC1000_TEMPERATURE_REG, Reg_Value,
2))
        printf("Error while reading I2C bus\r\n");

    Temp = ( Reg_Value[0] << 8 ) | Reg_Value[1];
    return ( (float)Temp / 65536 ) * 165 - 40;
}
```

8.9.3 Función Temp_Init:

Esta función se encarga de recopilar los dos valores anteriores junto con el nivel de batería, el tiempo de escritura y el nombre de la última imagen capturada, para almacenarlos en un archivo de texto para su posterior revisión y estudio. Este proceso se muestra en el siguiente código:

```
void Temp_Init(void) {

    static uint8_t regs[8] = {0};
    char data[125]={0};
    double Temperature = 0, Humidity = 0;
    double bateria = 0;
    double voltaje = 0;
    int len = 0;

    T1CONbits.TON = 0;
    T2CONbits.TON = 0;
    TMR2 = 0;
    T2CONbits.TON = 1;
    Temperature = HDC1000_ReadTemp();
    Humidity = HDC1000_ReadHum();

#ifdef DEBUG
    printf("Tiempo de uso del sensor T/H %d\r\n",TMR2);
#endif
    TMR2 = 0;
    T2CONbits.TON = 0;

    //Read the Time from de RTC
    DS3231_DateHour(regs);

    voltaje = MAX17043_ReadVoltage();
    bateria = MAX17043_ReadSoC();

#ifdef DEBUG
    printf("Temperture Read: %.2f C\r\n",Temperature);
    printf("Humidity Read: %.2f RH\r\n",Humidity);
    printf(" Voltaje %.2f",voltaje);
#endif
    len = sprintf(data,"%u/%u/%d %d:%d:%d Temperatura: %.2f Humedad: %.2f Bateria: %.2f Voltaje: %.2f %s Time Elapsed %.2fms\r\n",
        regs[4], regs[5], regs[6]+2000, regs[2], regs[1], regs[0],
        Temperature, Humidity, bateria, voltaje, file_CamNombre, tiempoMSRes);
    Result = f_open(&Fil2, (const TCHAR*)file_sensor, FA_WRITE |
FA_OPEN_APPEND);
#ifdef DEBUG
    if (Result == FR_OK) {
        //----- FILE WAS SUCESSFULLY CREATED -----
        printf("Archivo sensor.txt creado \r\n");
    }else{
        printf("No se creo sensor.txt \r\n");
    }
#endif

    T1CONbits.TON = 1;
    Result = f_write(&Fil2, data, len, &bw1);
    T1CONbits.TON = 0;
    Result = f_close(&Fil2);
}
```

8.10 Control de memoria SD

Para evitar conflictos de la SD cuando la cámara toma una imagen y cuando la radio empiece a transmitir se habilitó un pin (TX_IN_PROGRESS) digital en el módulo de la radio y otro pin digital de lectura en el dsPIC para saber si en el instante en el que la cámara efectuara la imagen, la SD está siendo usada o no.

El proceso para efectuar este control se muestra en el siguiente código:

```

if(TX_IN_PROGRESS && PIR_Detect){
    NoAhorroEnergia();
    if(!CamaraON()){
        DelayMs(2700);
//        FLASH = 1;
        if(Read_voltage() == 0){
            error = Camara_TomarImagen();
            DelayMs(100);
            error = Camara_TomarImagen();
        }
//        FLASH = 0;
    }
    TAKING_PHOTO = 0;
    AhorroEnergia();
    while(PIR);
    PIR_Detect=0;
}

```

8.11 Comportamiento como cámara trampa

Mencionado ya todo lo anterior podemos saber cómo se comporta el código final, cabe mencionar que hay un retraso de casi 3 segundos desde que el PIR se activa hasta que la cámara tome una imagen, esto es debido a que la cámara necesita un tiempo para poder estabilizarse y adaptarse al Ambiente. Al intentar tomas imágenes sin este retraso todas salieron en negro.

Todo este proceso de control se detalla en el siguiente código:

```

int main(void) {

    uint8_t error;
    CPU_Init();
    Camara_Init();
    MAX17043_Init();    //Initializes the batter voltage tester

    PIR_IntConfig();

```

```

CNPDBbits.CNPDB7 = 1;
while (PIR);
PIR_Detect=0;
DelayMs(100);
AhorroEnergia();

while(1)
{
    Sleep();
    if(TX_IN_PROGRESS && PIR_Detect){
        NoAhorroEnergia();
        if(!CamaraON()){
            DelayMs(2700);
            // FLASH = 1;
            if(Read_voltage() == 0){
                error = Camara_TomarImagen();
                DelayMs(100);
                error = Camara_TomarImagen();
            }
            // FLASH = 0;
        }
        TAKING_PHOTO = 0;
        AhorroEnergia();
        while(PIR);
        PIR_Detect=0;
    }
}

```

En el diagrama de flujos mostrado en la figura 15 se puede apreciar el comportamiento final del módulo para comportarse como una cámara trampa.

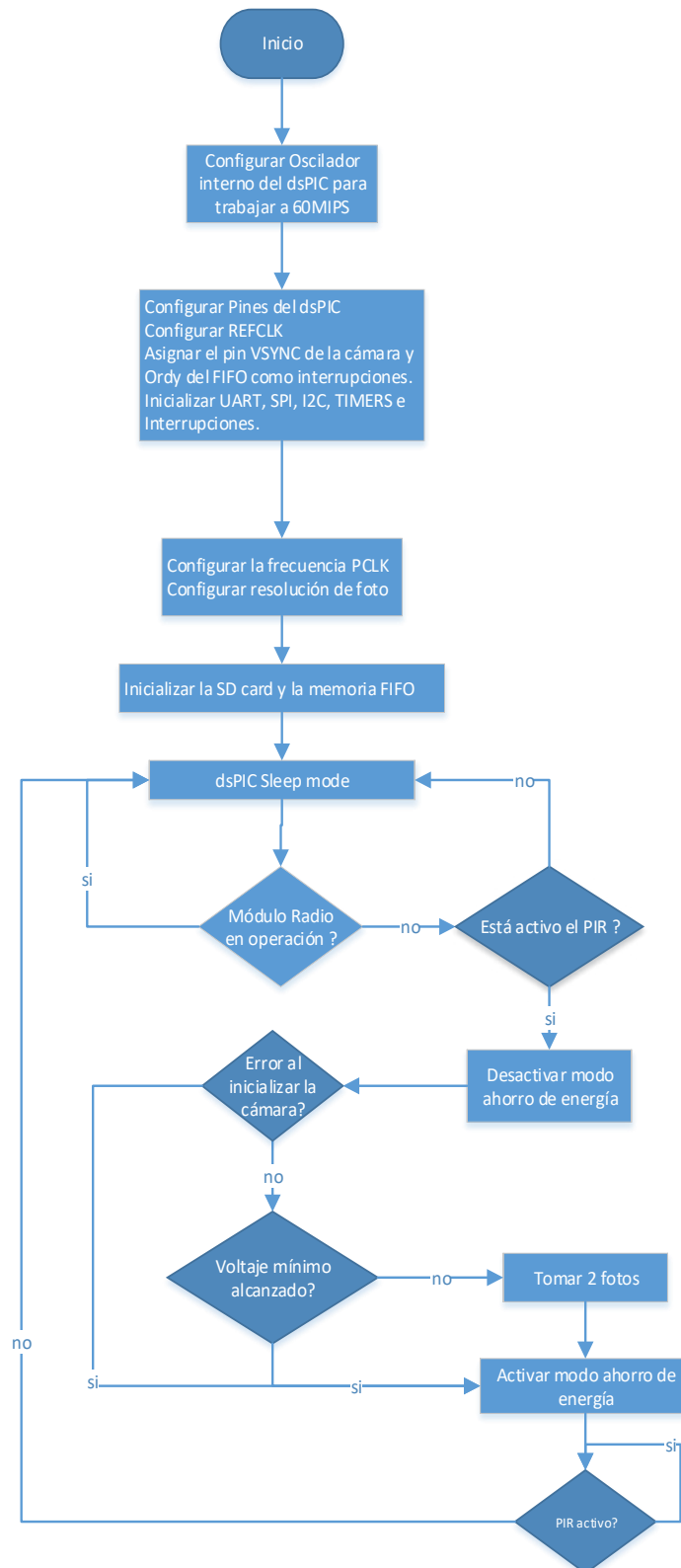


Fig. 15.- Diagrama de flujos del comportamiento como cámara Trampa.

CAPITULO 9. RESULTADOS

Los resultados obtenidos después de un exhaustivo control del sensor CMOS son presentados en la figura 16 la imagen que se muestra fue tomada con la máxima resolución. Para unos 5 MP el tamaño de la imagen fue de 2592x1944 pixeles.



Fig. 16.- Imagen de 5mp capturada por el sensor OV5642

Otro resultado de una serie de capturas con el módulo diseñado en este trabajo y permitió realizar la comparación con la cámara diseñada en el

proyecto TAPIRNET. De esta comparación de imágenes, se comprobó que la calidad y por ende una mayor cantidad de megapíxeles del sensor son importantes para poder determinar características físicas adicionales de los animales.

A continuación, se muestra las imágenes capturadas en las mañanas con ambas cámaras, en los casos de la Fig 17 y la Fig 18 las cámaras se posicionaron debajo de un árbol, bajo la sombra de las ramas, lo cual simula un escenario real en la selva donde la densidad de los arboles es muy espesa y no permite el ingreso directo de los rayos solares.



Fig. 17.- A la izquierda: Imagen capturada con el sensor CMOS de 5mp bajo sombra, a la derecha: imagen capturada con el sensor de 0.2 mp usada en el proyecto TAPIRNET.

En el caso de la cámara de 0.2MP se aprecia que las imágenes tienen una tonalidad rosa, tal como se muestra en la Fig 17. Esto se debe a que esta cámara no posee un filtro contra la luz infrarroja permitiendo el paso de este color directamente al sensor y produciendo este efecto en las imágenes capturadas. Esta carencia de filtro IR permite el uso de un flash infrarrojo para la captura de imágenes nocturnas sin afectar el comportamiento normal del animal en observación que se quiere capturar.

En la imagen tomada por el sensor de 5mp, tal como se muestra en la Fig 18 también se puede apreciar una muy pequeña tonalidad rosa en las esquinas donde se percibe la luz solar.



Fig. 18.- A la izquierda: Imagen capturada con el sensor CMOS de 5mp bajo sombra, a la derecha: imagen capturada con el sensor de 0.2 mp usada en el proyecto TAPIRNET.

En el caso de las imágenes que se tomaron bajo una exposición directa al sol (figura 19 y figura 20) de las cámaras (12pm aprox.) las imágenes mejoran considerablemente su nitidez y calidad, pero en esta situación se puede apreciar aún más que el sensor de 5mp tiene una muy buena calidad, pudiendo diferenciar de forma muy clara diversos rasgos propios del animal en este caso se puede apreciar las manchas que tiene los collares colocados en los perros y algunos rasgos específicos del animal como forma de la oreja, tamaño de la cola así como diversas características que pudieran tener los animales fotografiados por el módulo.



Fig. 19.- Imágenes capturadas con exposición directa del sol. A la izquierda: Sensor 0.2mp, a la derecha: Sensor 5mp.



Fig. 20.- Imágenes capturadas con exposición directa del sol. A la izquierda: Sensor de 5mp, a la derecha: Sensor de 0.2mp.

A continuación, se muestra las diversas imágenes capturadas por el sensor de 5mp:



Fig. 21.- Imagen bajo sombra tomada con sensor 5mp.



Fig. 22.- Imagen bajo sombra tomada con sensor 5mp



Fig. 23.- Imagen bajo sombra tomada con sensor 5mp



Fig. 24.- Imagen bajo sombra tomada con sensor 5mp

Los tiempos de escritura de los datos de información son resumidos en la siguiente tabla.

Tabla II

Tiempos de escritura de las imágenes en la memoria SD.

Imagen	Tamaño de la imagen (KB)	Tiempo de escritura de toda imagen (ms)
Fig 17	797	1411.59
Fig 18	813	1381.42
Fig 19	964	1668.74
Fig 20	917	1597.91
Fig 21	865	1450.88
Fig 22	969	1659.51
Fig 23	761	1284.99
Fig 24	861	1448.19

Nota: Las imágenes fueron capturadas a una resolución de 5MP.

Una prueba final fue realizada con la finalidad de tomar imágenes consecutivamente. En este test la resolución de la cámara se configuró a 2 MP (1920 x 1080). El tiempo que tomo para tomar 6 imágenes fue de 3 segundos. Este resultado demuestra que la cámara es lo suficientemente rápida para tomar imágenes consecutivamente con una alta resolución. Esta característica permite a los biólogos e investigadores de la fauna silvestre poder seleccionar la cantidad de imágenes consecutivas que pueden tomar cuando se activa la cámara.

CONCLUSIONES

Se cumplió con el objetivo principal de la tesis, al haber logrado el diseño de un sistema de captura y registro de imágenes con una cámara cmos de alta resolución controlado por un microcontrolador dsPIC33 de 60MIPS, tal como se muestran en las imágenes de las figuras 17, 18, 19, 20, 21, 22, 23 y 24, las mismas que se convertirán en los insumos para los biólogos investigadores, quienes, a partir de estos datos-imagen, podrán cumplir con su trabajo de cuantificar y determinar la abundancia relativa de la especie investigada en una determinada área geográfica de interés.

Se logró el bajo consumo de energía del sistema, debido al uso apropiado de dispositivos electrónicos y a la adecuada elaboración del producto de software para el control de captura de imágenes con una resolución de 5MP, que se evidencia en las siguientes afirmaciones:

- a. Uso de un mosfet para controlar el suministro de energía eléctrica del módulo-cámara solo cuando el sensor de proximidad detecte la presencia de uno o varios mamíferos grandes, logrando de esta manera optimizar el consumo de energía y autonomía de todo el sistema de captura fotográfica.
- b. Haber obtenido imágenes suficientemente nítidas para visualizar en detalle las diversas características del o los mamíferos fotografiados.
- c. Lograr la disponibilidad de hasta 32GB de almacenamiento, para el apoyo en los trabajos de investigación de los mamíferos realizado por los biólogos el cual les permitirá llevar un registro de sus investigaciones y de esta manera tener un soporte adicional para la medición de la abundancia relativa de diversas especies en el área estudiada.

RECOMENDACIONES

Es posible lograr mejoras en los tiempos de escritura de los datos en la memoria micro SD optando por otro microcontrolador con mayor velocidad del bus SPI y el soporte para un protocolo USB como es el caso de la familia PIC de Microchip.

Otra mejora para evitar pérdida de datos durante el almacenamiento de los datos-imagen cuando esta sobrepase el tamaño de 1MB se puede utilizar una memoria FIFO de mayor capacidad.

En el caso que se requiera menor resolución y una mayor velocidad de captura de imágenes por segundo es posible mejorar este requerimiento por software y de esta manera tener una mayor cantidad de imágenes por segundo.

BIBLIOGRAFÍA

- Ancrenaz, M., Hearn, A., Ross, J., Sollmann, R., & Wilting, A. (2012). *Handbook for wildlife monitoring using camera-traps*. . Kota Kinabalu, Sabah, Malaysia: BBEC II Secretariat.
- Arducam. (2016, Enero). *Arducam 5MP Plus OV5642 Mini Module Camera Shield SPI Camera Module for Arduino UNO Mega2560 Board*. Retrieved from SPI Camera for Arduino: <https://www.arducam.com/product/arducam-5mp-plus-spi-cam-arduino-ov5642/>
- AVERLOGIC. (2016, Febrero). *FIFOs*. Retrieved from Averlogic: <http://www.averlogic.com/AL440B.asp>
- Camacho, I., Baquerizo, R., Palomino, J., & Gagliardi, G. (2016). Design and Deployment of a Multimedia Wireless Sensor Network for Wildlife Inventory in Western Amazon Rainforest. *UNESCO Chair in Technologies for Development from innovation to School Impact*.
- Cham, E. (2019, Octubre 14). *FatFs*. Retrieved from Generic FAT Filesystem Module: http://elm-chan.org/fsw/ff/00index_e.html
- Cutler, T. L., & Swann, D. E. (1999). Using Remote Photography in Wildlife Ecology: A Review. *Wildlife Society Bulletin*, 571-581.
- Gaines, W. L., Harrod, R. J., & Lehmkuhl, J. F. (1999). Monitoring Biodiversity: Quantification and Interpretation. *USDA Forest Service - General Technical Report PNW*, 27.
- Hernández-Sampieri, R., & Mendoza Torres, C. P. (2018). *Metodología de la investigación: Las rutas cuantitativa, cualitativa y mixta*.
- Lyra-Jorge, M., Ciocheti, G., Pivello, V., & Meirelles, S. (2008). Comparing methods for sampling large- and medium-sized mammals: Camera traps and track plots. *European Journal of Wildlife Research*, 739-744.
- Matt. (2015, Noviembre 22). *Raspberry Pi microSD Card Shoot-out*. Retrieved from Raspberry-spy: <https://www.raspberrypi-spy.co.uk/2015/11/raspberry-pi-microsd-card-shoot-out/>

- McCullough, D., Pei, K., & Wang, Y. (2000). Home Range, Activity Patterns, and Habitat Relations of Reeves' Muntjacs in Taiwan. *Journal of Wildlife Management*, 430-441.
- Microchip. (11, Noviembre 29). *Serial Peripheral Interface (SPI)*. Retrieved from dsPIC33E/PIC24E Family Reference Manual Section 18: <http://ww1.microchip.com/downloads/en/DeviceDoc/70569C.pdf>
- Microchip. (2010, Enero 5). *Timers*. Retrieved from dsPIC33E/PIC24E Family Reference Manual Section 11: <http://ww1.microchip.com/downloads/en/devicedoc/s11.pdf>
- Microchip. (2011, Noviembre 29). *Direct Memory Access (DMA)*. Retrieved from dsPIC33E/PIC24E Family Reference Manual Section 22: <http://ww1.microchip.com/downloads/en/devicedoc/70348c.pdf>
- Microchip. (2011, Noviembre 29). *Oscillator*. Retrieved from dsPIC33E/PIC24E Family Reference Manual Section 7: <http://ww1.microchip.com/downloads/en/DeviceDoc/70580C.pdf>
- Microchip. (2011, Noviembre 29). *UART*. Retrieved from dsPIC33E/PIC24E Family Reference Manual Section 17: <http://ww1.microchip.com/downloads/en/DeviceDoc/70582C.pdf>
- Microchip. (2013, Octubre 28). *Inter-Integrated Circuit*. Retrieved from dsPIC33/PIC24 Family Reference Manual: <http://ww1.microchip.com/downloads/en/devicedoc/70000195f.pdf>
- Pitt, B. (2013, Setiembre 19). *How many megapixels do you need?* Retrieved from DPREVIEW: <https://www.dpreview.com/post/1313669123/how-many-megapixels>
- Rovero, F., Zimmermann, F., Berzi, D., & Meek, P. (2013). "Which camera trap type and how many do I need?" A review of camera features and study designs for a range of wildlife research applications. *Hystrix, the Italian Journal of Mammalogy*, 148-156.
- Rowcliffe, J. M., Field, J., Turvey, S. T., & Carbone, C. (2008). Estimating animal density using camera traps without the need for individual recognition. *Journal of Applied Ecology*, 1228-1236.
- Treves, A., Mwima, P., Plumptre, A., & Isoke, S. (2010). Camera-trapping forest-woodland wildlife of western Uganda reveals how gregariousness biases estimates of relative abundance and distribution. *Biological Conservation*, 521-528.
- UCTRONICS. (2015, Diciembre). *5 Mega pixel Camera Module OV5642 /w CS mount Lens*. Retrieved from UCTRONICS: <https://www.uctronics.com/mega-pixel-camera-module-ov5642-cs-mount-lens-p-1869l.html>

Zarzosa, M., Palomino, J., Chávez, P., & Camacho, L. (2016). Development of a dedicated controller for a 5MP CMOS sensor applied to a wireless image sensor network for large mammals monitoring. *ANDESCON 2016 IEEE*.

ANEXOS

1. Matriz de consistencia y operacionalización de las variables.

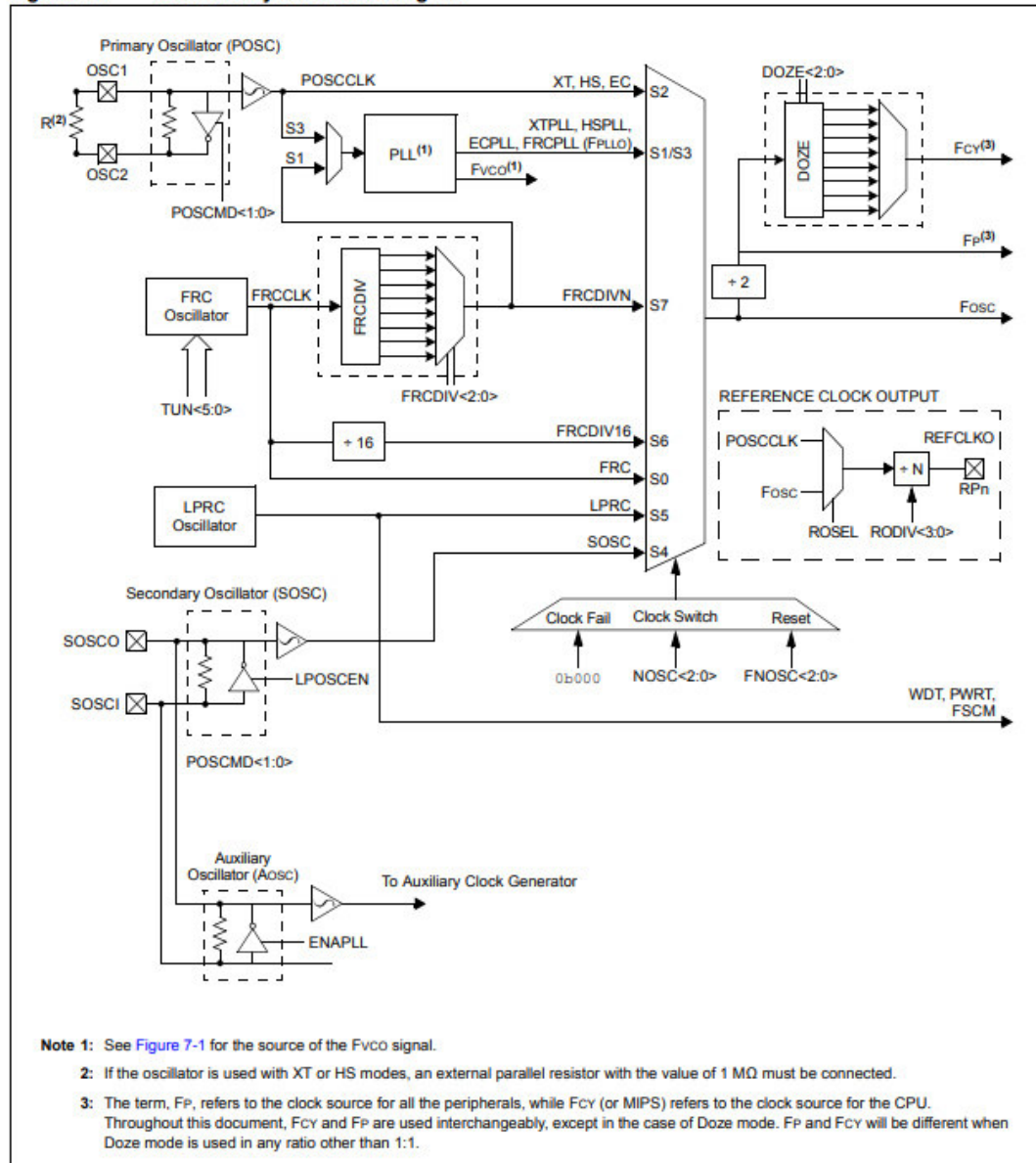
Este instrumento ha sido elaborado en una tabla mostrada a continuación, cuyas etiquetas de las columnas son: problema, objetivos, hipótesis, variables y dimensiones con sus respectivos indicadores. Esta información permite evaluar el grado de coherencia y conexión razonable con el título de la presente tesis. Observando en forma vertical esta nos muestra las actividades involucradas para evidenciar los resultados. Desde el punto de vista horizontal, se visualiza la relación causa efecto.

PROBLEMA	OBJETIVO	HIPOTESIS	VARIABLE	DIMENSIÓN	INDICADORES
¿En qué medida el diseño de un sistema de registro de imágenes 5MP de bajo consumo mejorará el monitoreo e inventario de los animales grandes en zonas rurales del país?	Diseñar un sistema de registro de imágenes de 5MP de bajo consumo para el monitoreo e inventario de animales grandes en zonas rurales del país.	El diseño de un sistema de registro de imágenes de 5MP de bajo consumo mejora el monitoreo e inventario de animales grandes en zonas rurales del país.	“Sistema de registro de imágenes de 5MP”: Detecta, captura y almacena imágenes de animales grandes.	- Dispositivos electrónicos - Eficiencia del sistema - Disponibilidad	- Sensor CMOs de 5MP. - Capacidad de almacenamiento. - Sensor de Movimiento. - Tolerancia a fallos. - Numero de fotografías registradas por segundo. - Control eficiente
¿De qué manera el diseño de dispositivos electrónicos para un sistema de registro de imágenes de 5MP permiten el monitoreo e inventario de los animales grandes en zonas rurales del país.?	Diseñar dispositivos electrónicos para un sistema de registro de imágenes de 5MP que permitan el monitoreo e inventario de los animales grandes en zonas rurales del país	El diseño de dispositivos electrónicos para un sistema de registro de imágenes de 5MP permite el monitoreo e inventario de los animales grandes en zonas rurales del país	“Monitoreo e Inventario de los animales grandes”: Es el seguimiento de la presencia de animales grandes en el área del alcance óptico del equipo.	- Calidad de las imágenes - Cuantificación de animales grandes captados - Almacenamiento de la imagen.	- Resolución de las imágenes. - Formato jpeg - Cantidad de imágenes almacenadas. - Especies capturadas - Tiempo de espera
¿De qué manera el diseño de un sistema de registro de imágenes de 5MP permite la cuantificación de animales grandes captados en el monitoreo e inventario de los animales grandes en zonas rurales del país ?	Diseñar un sistema de registro de imágenes de 5MP que permita la cuantificación de mamíferos captados en el monitoreo e inventario de los animales grandes en zonas rurales del país	El diseño de un sistema de registro de imágenes de 5MP mejora la cuantificación de mamíferos captados en el monitoreo e inventario de los animales grandes en zonas rurales del país			
¿Cómo el diseño de un sistema de registro de imágenes de 5MP mejora la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país?	Diseñar un sistema de registro de imágenes de 5MP que permita la mejora de la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país	El diseño de un sistema de registro de imágenes de 5MP mejora la calidad de las imágenes en el monitoreo e inventario de animales grandes en zonas rurales del país			

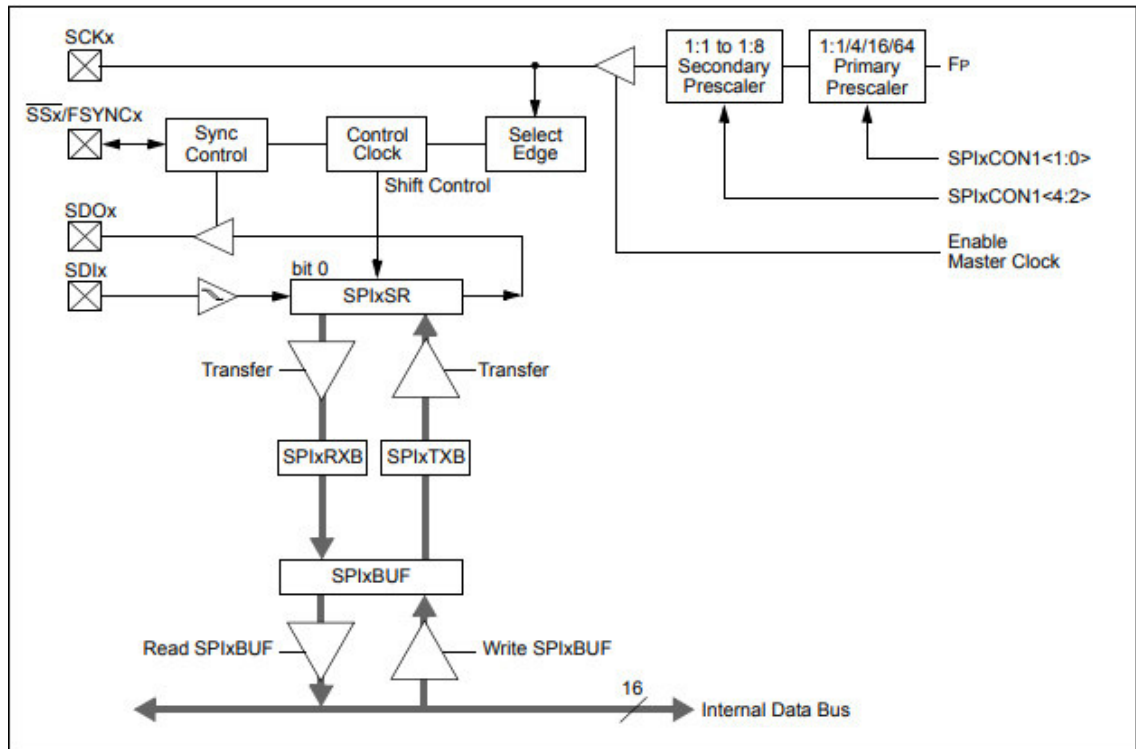
2. Diagrama de bloques del módulo del oscilador

Oscillator Module

Figure 1-1: Oscillator System Block Diagram



3. Diagrama de bloques del bus SPI (Modo estándar)



4. Código completo de la configuración de la cámara:

```
void Camara_Set_VGAprev_YUVmode(void) {

    //13.1.1 VGA Preview
    //; for the setting , 24M Mlck input and 24M Plck output
    //;15fps YUV mode
    Camara_EnviaBytesI2C(0x3103, 0x93);
    Camara_EnviaBytesI2C(0x3008, 0x82);
    Camara_EnviaBytesI2C(0x3017, 0x7f);
    Camara_EnviaBytesI2C(0x3018, 0xfc);
    Camara_EnviaBytesI2C(0x3810, 0xc2);
    Camara_EnviaBytesI2C(0x3615, 0xf0);
    Camara_EnviaBytesI2C(0x3000, 0x00);
    Camara_EnviaBytesI2C(0x3001, 0x00);
    Camara_EnviaBytesI2C(0x3002, 0x5c);
    Camara_EnviaBytesI2C(0x3003, 0x00);
    Camara_EnviaBytesI2C(0x3004, 0xff);
    Camara_EnviaBytesI2C(0x3005, 0xff);
    Camara_EnviaBytesI2C(0x3006, 0x43);
    Camara_EnviaBytesI2C(0x3007, 0x37);
    Camara_EnviaBytesI2C(0x3011, 0x08);
    Camara_EnviaBytesI2C(0x3010, 0x10);
    Camara_EnviaBytesI2C(0x460c, 0x22);
    Camara_EnviaBytesI2C(0x3815, 0x04);
    Camara_EnviaBytesI2C(0x370c, 0xa0);
    Camara_EnviaBytesI2C(0x3602, 0xfc);
    Camara_EnviaBytesI2C(0x3612, 0xff);
    Camara_EnviaBytesI2C(0x3634, 0xc0);
    Camara_EnviaBytesI2C(0x3613, 0x00);
    Camara_EnviaBytesI2C(0x3605, 0x7c);
    Camara_EnviaBytesI2C(0x3621, 0x09);
    Camara_EnviaBytesI2C(0x3622, 0x60);
    Camara_EnviaBytesI2C(0x3604, 0x40);
    Camara_EnviaBytesI2C(0x3603, 0xa7);
    Camara_EnviaBytesI2C(0x3603, 0x27);
    Camara_EnviaBytesI2C(0x4000, 0x21);
    Camara_EnviaBytesI2C(0x401d, 0x22);
    Camara_EnviaBytesI2C(0x3600, 0x54);
    Camara_EnviaBytesI2C(0x3605, 0x04);
    Camara_EnviaBytesI2C(0x3606, 0x3f);
    Camara_EnviaBytesI2C(0x3c01, 0x80);
    Camara_EnviaBytesI2C(0x5000, 0x4f);
    Camara_EnviaBytesI2C(0x5020, 0x04);

    Camara_EnviaBytesI2C(0x5181, 0x79);
    Camara_EnviaBytesI2C(0x5182, 0x00);
    Camara_EnviaBytesI2C(0x5185, 0x22);
    Camara_EnviaBytesI2C(0x5197, 0x01);
    Camara_EnviaBytesI2C(0x5001, 0xff);
    Camara_EnviaBytesI2C(0x5500, 0x0a);
    Camara_EnviaBytesI2C(0x5504, 0x00);
    Camara_EnviaBytesI2C(0x5505, 0x7f);
    Camara_EnviaBytesI2C(0x5080, 0x08);
    Camara_EnviaBytesI2C(0x300e, 0x18);
    Camara_EnviaBytesI2C(0x4610, 0x00);
    Camara_EnviaBytesI2C(0x471d, 0x05);
    Camara_EnviaBytesI2C(0x4708, 0x06);
    Camara_EnviaBytesI2C(0x3808, 0x02);
    Camara_EnviaBytesI2C(0x3809, 0x80);
    Camara_EnviaBytesI2C(0x380a, 0x01);
    Camara_EnviaBytesI2C(0x380b, 0xe0);
    Camara_EnviaBytesI2C(0x380e, 0x07);
    Camara_EnviaBytesI2C(0x380f, 0xd0);
    Camara_EnviaBytesI2C(0x501f, 0x00);
    Camara_EnviaBytesI2C(0x5000, 0x4f);
    Camara_EnviaBytesI2C(0x4300, 0x30);
```

```

Camara_EnviaBytesI2C(0x3503, 0x07);
Camara_EnviaBytesI2C(0x3501, 0x73);
Camara_EnviaBytesI2C(0x3502, 0x80);
Camara_EnviaBytesI2C(0x350b, 0x00);
Camara_EnviaBytesI2C(0x3503, 0x07);
Camara_EnviaBytesI2C(0x3824, 0x11);
Camara_EnviaBytesI2C(0x3501, 0x1e);
Camara_EnviaBytesI2C(0x3502, 0x80);
Camara_EnviaBytesI2C(0x350b, 0x7f);
Camara_EnviaBytesI2C(0x380c, 0x0c);
Camara_EnviaBytesI2C(0x380d, 0x80);
Camara_EnviaBytesI2C(0x380e, 0x03);
Camara_EnviaBytesI2C(0x380f, 0xe8);
Camara_EnviaBytesI2C(0x3a0d, 0x04);
Camara_EnviaBytesI2C(0x3a0e, 0x03);
Camara_EnviaBytesI2C(0x3818, 0xc1);
Camara_EnviaBytesI2C(0x3705, 0xdb);
Camara_EnviaBytesI2C(0x370a, 0x81);
Camara_EnviaBytesI2C(0x3801, 0x80);
Camara_EnviaBytesI2C(0x3621, 0x87);
Camara_EnviaBytesI2C(0x3801, 0x50);
Camara_EnviaBytesI2C(0x3803, 0x08);
Camara_EnviaBytesI2C(0x3827, 0x08);
Camara_EnviaBytesI2C(0x3810, 0x40);
Camara_EnviaBytesI2C(0x3804, 0x05);
Camara_EnviaBytesI2C(0x3805, 0x00);

Camara_EnviaBytesI2C(0x5682, 0x05);
Camara_EnviaBytesI2C(0x5683, 0x00);
Camara_EnviaBytesI2C(0x3806, 0x03);
Camara_EnviaBytesI2C(0x3807, 0xc0);
Camara_EnviaBytesI2C(0x5686, 0x03);
Camara_EnviaBytesI2C(0x5687, 0xbc);
Camara_EnviaBytesI2C(0x3a00, 0x78);
Camara_EnviaBytesI2C(0x3a1a, 0x05);
Camara_EnviaBytesI2C(0x3a13, 0x30);
Camara_EnviaBytesI2C(0x3a18, 0x00);
Camara_EnviaBytesI2C(0x3a19, 0x7c);
Camara_EnviaBytesI2C(0x3a08, 0x12);
Camara_EnviaBytesI2C(0x3a09, 0xc0);
Camara_EnviaBytesI2C(0x3a0a, 0x0f);
Camara_EnviaBytesI2C(0x3a0b, 0xa0);
Camara_EnviaBytesI2C(0x350c, 0x07);
Camara_EnviaBytesI2C(0x350d, 0xd0);
Camara_EnviaBytesI2C(0x3500, 0x00);
Camara_EnviaBytesI2C(0x3501, 0x00);
Camara_EnviaBytesI2C(0x3502, 0x00);
Camara_EnviaBytesI2C(0x350a, 0x00);
Camara_EnviaBytesI2C(0x350b, 0x00);
Camara_EnviaBytesI2C(0x3503, 0x00);
Camara_EnviaBytesI2C(0x528a, 0x02);
Camara_EnviaBytesI2C(0x528b, 0x04);
Camara_EnviaBytesI2C(0x528c, 0x08);
Camara_EnviaBytesI2C(0x528d, 0x08);
Camara_EnviaBytesI2C(0x528e, 0x08);
Camara_EnviaBytesI2C(0x528f, 0x10);
Camara_EnviaBytesI2C(0x5290, 0x10);
Camara_EnviaBytesI2C(0x5292, 0x00);
Camara_EnviaBytesI2C(0x5293, 0x02);
Camara_EnviaBytesI2C(0x5294, 0x00);
Camara_EnviaBytesI2C(0x5295, 0x02);
Camara_EnviaBytesI2C(0x5296, 0x00);
Camara_EnviaBytesI2C(0x5297, 0x02);
Camara_EnviaBytesI2C(0x5298, 0x00);
Camara_EnviaBytesI2C(0x5299, 0x02);
Camara_EnviaBytesI2C(0x529a, 0x00);
Camara_EnviaBytesI2C(0x529b, 0x02);

```

```

Camara_EnviaBytesI2C(0x529c, 0x00);
Camara_EnviaBytesI2C(0x529d, 0x02);
Camara_EnviaBytesI2C(0x529e, 0x00);
Camara_EnviaBytesI2C(0x529f, 0x02);
Camara_EnviaBytesI2C(0x3030, 0x2b);
Camara_EnviaBytesI2C(0x3a02, 0x00);
Camara_EnviaBytesI2C(0x3a03, 0x7d);
Camara_EnviaBytesI2C(0x3a04, 0x00);

Camara_EnviaBytesI2C(0x3a14, 0x00);
Camara_EnviaBytesI2C(0x3a15, 0x7d);
Camara_EnviaBytesI2C(0x3a16, 0x00);
Camara_EnviaBytesI2C(0x3a00, 0x78);
Camara_EnviaBytesI2C(0x3a08, 0x09);
Camara_EnviaBytesI2C(0x3a09, 0x60);
Camara_EnviaBytesI2C(0x3a0a, 0x07);
Camara_EnviaBytesI2C(0x3a0b, 0xd0);
Camara_EnviaBytesI2C(0x3a0d, 0x08);
Camara_EnviaBytesI2C(0x3a0e, 0x06);
Camara_EnviaBytesI2C(0x5193, 0x70);
Camara_EnviaBytesI2C(0x589b, 0x04);
Camara_EnviaBytesI2C(0x589a, 0xc5);
Camara_EnviaBytesI2C(0x401e, 0x20);
Camara_EnviaBytesI2C(0x4001, 0x42);
Camara_EnviaBytesI2C(0x401c, 0x04);
Camara_EnviaBytesI2C(0x528a, 0x01);
Camara_EnviaBytesI2C(0x528b, 0x04);
Camara_EnviaBytesI2C(0x528c, 0x08);
Camara_EnviaBytesI2C(0x528d, 0x10);
Camara_EnviaBytesI2C(0x528e, 0x20);
Camara_EnviaBytesI2C(0x528f, 0x28);
Camara_EnviaBytesI2C(0x5290, 0x30);
Camara_EnviaBytesI2C(0x5292, 0x00);
Camara_EnviaBytesI2C(0x5293, 0x01);
Camara_EnviaBytesI2C(0x5294, 0x00);
Camara_EnviaBytesI2C(0x5295, 0x04);
Camara_EnviaBytesI2C(0x5296, 0x00);
Camara_EnviaBytesI2C(0x5297, 0x08);
Camara_EnviaBytesI2C(0x5298, 0x00);
Camara_EnviaBytesI2C(0x5299, 0x10);
Camara_EnviaBytesI2C(0x529a, 0x00);
Camara_EnviaBytesI2C(0x529b, 0x20);
Camara_EnviaBytesI2C(0x529c, 0x00);
Camara_EnviaBytesI2C(0x529d, 0x28);
Camara_EnviaBytesI2C(0x529e, 0x00);
Camara_EnviaBytesI2C(0x529f, 0x30);
Camara_EnviaBytesI2C(0x5282, 0x00);
Camara_EnviaBytesI2C(0x5300, 0x00);
Camara_EnviaBytesI2C(0x5301, 0x20);
Camara_EnviaBytesI2C(0x5302, 0x00);
Camara_EnviaBytesI2C(0x5303, 0x7c);
Camara_EnviaBytesI2C(0x530c, 0x00);
Camara_EnviaBytesI2C(0x530d, 0x0c);
Camara_EnviaBytesI2C(0x530e, 0x20);
Camara_EnviaBytesI2C(0x530f, 0x80);
Camara_EnviaBytesI2C(0x5310, 0x20);
Camara_EnviaBytesI2C(0x5311, 0x80);

Camara_EnviaBytesI2C(0x5308, 0x20);
Camara_EnviaBytesI2C(0x5309, 0x40);
Camara_EnviaBytesI2C(0x5304, 0x00);
Camara_EnviaBytesI2C(0x5305, 0x30);
Camara_EnviaBytesI2C(0x5306, 0x00);
Camara_EnviaBytesI2C(0x5307, 0x80);
Camara_EnviaBytesI2C(0x5314, 0x08);
Camara_EnviaBytesI2C(0x5315, 0x20);
Camara_EnviaBytesI2C(0x5319, 0x30);

```

```

Camara_EnviaBytesI2C(0x5316, 0x10);
Camara_EnviaBytesI2C(0x5317, 0x00);
Camara_EnviaBytesI2C(0x5318, 0x02);
Camara_EnviaBytesI2C(0x5402, 0x3f);
Camara_EnviaBytesI2C(0x5403, 0x00);
Camara_EnviaBytesI2C(0x3406, 0x00);
Camara_EnviaBytesI2C(0x5180, 0xff);
Camara_EnviaBytesI2C(0x5181, 0x52);
Camara_EnviaBytesI2C(0x5182, 0x11);
Camara_EnviaBytesI2C(0x5183, 0x14);
Camara_EnviaBytesI2C(0x5184, 0x25);
Camara_EnviaBytesI2C(0x5185, 0x24);
Camara_EnviaBytesI2C(0x5186, 0x06);
Camara_EnviaBytesI2C(0x5187, 0x08);
Camara_EnviaBytesI2C(0x5188, 0x08);
Camara_EnviaBytesI2C(0x5189, 0x7c);
Camara_EnviaBytesI2C(0x518a, 0x60);
Camara_EnviaBytesI2C(0x518b, 0xb2);
Camara_EnviaBytesI2C(0x518c, 0xb2);
Camara_EnviaBytesI2C(0x518d, 0x44);
Camara_EnviaBytesI2C(0x518e, 0x3d);
Camara_EnviaBytesI2C(0x518f, 0x58);
Camara_EnviaBytesI2C(0x5190, 0x46);
Camara_EnviaBytesI2C(0x5191, 0xf8);
Camara_EnviaBytesI2C(0x5192, 0x04);
Camara_EnviaBytesI2C(0x5193, 0x70);
Camara_EnviaBytesI2C(0x5194, 0xf0);
Camara_EnviaBytesI2C(0x5195, 0xf0);
Camara_EnviaBytesI2C(0x5196, 0x03);
Camara_EnviaBytesI2C(0x5197, 0x01);
Camara_EnviaBytesI2C(0x5198, 0x04);
Camara_EnviaBytesI2C(0x5199, 0x12);
Camara_EnviaBytesI2C(0x519a, 0x04);
Camara_EnviaBytesI2C(0x519b, 0x00);
Camara_EnviaBytesI2C(0x519c, 0x06);
Camara_EnviaBytesI2C(0x519d, 0x82);
Camara_EnviaBytesI2C(0x519e, 0x00);
Camara_EnviaBytesI2C(0x5025, 0x80);
Camara_EnviaBytesI2C(0x5583, 0x40);

Camara_EnviaBytesI2C(0x5584, 0x40);
Camara_EnviaBytesI2C(0x5580, 0x02);
Camara_EnviaBytesI2C(0x5000, 0xcf);
Camara_EnviaBytesI2C(0x3710, 0x10);
Camara_EnviaBytesI2C(0x3632, 0x51);
Camara_EnviaBytesI2C(0x3702, 0x10);
Camara_EnviaBytesI2C(0x3703, 0xb2);
Camara_EnviaBytesI2C(0x3704, 0x18);
Camara_EnviaBytesI2C(0x370b, 0x40);
Camara_EnviaBytesI2C(0x370d, 0x03);
Camara_EnviaBytesI2C(0x3631, 0x01);
Camara_EnviaBytesI2C(0x3632, 0x52);
Camara_EnviaBytesI2C(0x3606, 0x24);
Camara_EnviaBytesI2C(0x3620, 0x96);
Camara_EnviaBytesI2C(0x5785, 0x07);
Camara_EnviaBytesI2C(0x3a13, 0x30);
Camara_EnviaBytesI2C(0x3600, 0x52);
Camara_EnviaBytesI2C(0x3604, 0x48);
Camara_EnviaBytesI2C(0x3606, 0x1b);
Camara_EnviaBytesI2C(0x370d, 0x0b);
Camara_EnviaBytesI2C(0x370f, 0xc0);
Camara_EnviaBytesI2C(0x3709, 0x01);
Camara_EnviaBytesI2C(0x3823, 0x00);
Camara_EnviaBytesI2C(0x5007, 0x00);
Camara_EnviaBytesI2C(0x5009, 0x00);
Camara_EnviaBytesI2C(0x5011, 0x00);
Camara_EnviaBytesI2C(0x5013, 0x00);

```

```

Camara_EnviaBytesI2C(0x519e, 0x00);
Camara_EnviaBytesI2C(0x5086, 0x00);
Camara_EnviaBytesI2C(0x5087, 0x00);
Camara_EnviaBytesI2C(0x5088, 0x00);
Camara_EnviaBytesI2C(0x5089, 0x00);
Camara_EnviaBytesI2C(0x302b, 0x00);
Camara_EnviaBytesI2C(0x5001, 0xFF);
Camara_EnviaBytesI2C(0x5583, 0x50);
Camara_EnviaBytesI2C(0x5584, 0x50);
Camara_EnviaBytesI2C(0x5580, 0x02);
Camara_EnviaBytesI2C(0x3c01, 0x80);
Camara_EnviaBytesI2C(0x3c00, 0x04);
//;LENS
Camara_EnviaBytesI2C(0x5800, 0x48);
Camara_EnviaBytesI2C(0x5801, 0x31);
Camara_EnviaBytesI2C(0x5802, 0x21);
Camara_EnviaBytesI2C(0x5803, 0x1b);
Camara_EnviaBytesI2C(0x5804, 0x1a);
Camara_EnviaBytesI2C(0x5805, 0x1e);
Camara_EnviaBytesI2C(0x5806, 0x29);
Camara_EnviaBytesI2C(0x5807, 0x38);

Camara_EnviaBytesI2C(0x5808, 0x26);
Camara_EnviaBytesI2C(0x5809, 0x17);
Camara_EnviaBytesI2C(0x580a, 0x11);
Camara_EnviaBytesI2C(0x580b, 0xe);
Camara_EnviaBytesI2C(0x580c, 0xd);
Camara_EnviaBytesI2C(0x580d, 0xe);
Camara_EnviaBytesI2C(0x580e, 0x13);
Camara_EnviaBytesI2C(0x580f, 0x1a);
Camara_EnviaBytesI2C(0x5810, 0x15);
Camara_EnviaBytesI2C(0x5811, 0xd);
Camara_EnviaBytesI2C(0x5812, 0x8);
Camara_EnviaBytesI2C(0x5813, 0x5);
Camara_EnviaBytesI2C(0x5814, 0x4);
Camara_EnviaBytesI2C(0x5815, 0x5);
Camara_EnviaBytesI2C(0x5816, 0x9);
Camara_EnviaBytesI2C(0x5817, 0xd);
Camara_EnviaBytesI2C(0x5818, 0x11);
Camara_EnviaBytesI2C(0x5819, 0xa);
Camara_EnviaBytesI2C(0x581a, 0x4);
Camara_EnviaBytesI2C(0x581b, 0x0);
Camara_EnviaBytesI2C(0x581c, 0x0);
Camara_EnviaBytesI2C(0x581d, 0x1);
Camara_EnviaBytesI2C(0x581e, 0x6);
Camara_EnviaBytesI2C(0x581f, 0x9);
Camara_EnviaBytesI2C(0x5820, 0x12);
Camara_EnviaBytesI2C(0x5821, 0xb);
Camara_EnviaBytesI2C(0x5822, 0x4);
Camara_EnviaBytesI2C(0x5823, 0x0);
Camara_EnviaBytesI2C(0x5824, 0x0);
Camara_EnviaBytesI2C(0x5825, 0x1);
Camara_EnviaBytesI2C(0x5826, 0x6);
Camara_EnviaBytesI2C(0x5827, 0xa);
Camara_EnviaBytesI2C(0x5828, 0x17);
Camara_EnviaBytesI2C(0x5829, 0xf);
Camara_EnviaBytesI2C(0x582a, 0x9);
Camara_EnviaBytesI2C(0x582b, 0x6);
Camara_EnviaBytesI2C(0x582c, 0x5);
Camara_EnviaBytesI2C(0x582d, 0x6);
Camara_EnviaBytesI2C(0x582e, 0xa);
Camara_EnviaBytesI2C(0x582f, 0xe);
Camara_EnviaBytesI2C(0x5830, 0x28);
Camara_EnviaBytesI2C(0x5831, 0x1a);
Camara_EnviaBytesI2C(0x5832, 0x11);
Camara_EnviaBytesI2C(0x5833, 0xe);
Camara_EnviaBytesI2C(0x5834, 0xe);

```

```

Camara_EnviaBytesI2C(0x5835, 0xf);
Camara_EnviaBytesI2C(0x5836, 0x15);
Camara_EnviaBytesI2C(0x5837, 0x1d);

Camara_EnviaBytesI2C(0x5838, 0x6e);
Camara_EnviaBytesI2C(0x5839, 0x39);
Camara_EnviaBytesI2C(0x583a, 0x27);
Camara_EnviaBytesI2C(0x583b, 0x1f);
Camara_EnviaBytesI2C(0x583c, 0x1e);
Camara_EnviaBytesI2C(0x583d, 0x23);
Camara_EnviaBytesI2C(0x583e, 0x2f);
Camara_EnviaBytesI2C(0x583f, 0x41);
Camara_EnviaBytesI2C(0x5840, 0xe);
Camara_EnviaBytesI2C(0x5841, 0xc);
Camara_EnviaBytesI2C(0x5842, 0xd);
Camara_EnviaBytesI2C(0x5843, 0xc);
Camara_EnviaBytesI2C(0x5844, 0xc);
Camara_EnviaBytesI2C(0x5845, 0xc);
Camara_EnviaBytesI2C(0x5846, 0xc);
Camara_EnviaBytesI2C(0x5847, 0xc);
Camara_EnviaBytesI2C(0x5848, 0xd);
Camara_EnviaBytesI2C(0x5849, 0xe);
Camara_EnviaBytesI2C(0x584a, 0xe);
Camara_EnviaBytesI2C(0x584b, 0xa);
Camara_EnviaBytesI2C(0x584c, 0xe);
Camara_EnviaBytesI2C(0x584d, 0xe);
Camara_EnviaBytesI2C(0x584e, 0x10);
Camara_EnviaBytesI2C(0x584f, 0x10);
Camara_EnviaBytesI2C(0x5850, 0x11);
Camara_EnviaBytesI2C(0x5851, 0xa);
Camara_EnviaBytesI2C(0x5852, 0xf);
Camara_EnviaBytesI2C(0x5853, 0xe);
Camara_EnviaBytesI2C(0x5854, 0x10);
Camara_EnviaBytesI2C(0x5855, 0x10);
Camara_EnviaBytesI2C(0x5856, 0x10);
Camara_EnviaBytesI2C(0x5857, 0xa);
Camara_EnviaBytesI2C(0x5858, 0xe);
Camara_EnviaBytesI2C(0x5859, 0xe);
Camara_EnviaBytesI2C(0x585a, 0xf);
Camara_EnviaBytesI2C(0x585b, 0xf);
Camara_EnviaBytesI2C(0x585c, 0xf);
Camara_EnviaBytesI2C(0x585d, 0xa);
Camara_EnviaBytesI2C(0x585e, 0x9);
Camara_EnviaBytesI2C(0x585f, 0xd);
Camara_EnviaBytesI2C(0x5860, 0xc);
Camara_EnviaBytesI2C(0x5861, 0xb);
Camara_EnviaBytesI2C(0x5862, 0xd);
Camara_EnviaBytesI2C(0x5863, 0x7);
Camara_EnviaBytesI2C(0x5864, 0x17);
Camara_EnviaBytesI2C(0x5865, 0x14);
Camara_EnviaBytesI2C(0x5866, 0x18);
Camara_EnviaBytesI2C(0x5867, 0x18);

Camara_EnviaBytesI2C(0x5868, 0x16);
Camara_EnviaBytesI2C(0x5869, 0x12);
Camara_EnviaBytesI2C(0x586a, 0x1b);
Camara_EnviaBytesI2C(0x586b, 0x1a);
Camara_EnviaBytesI2C(0x586c, 0x16);
Camara_EnviaBytesI2C(0x586d, 0x16);
Camara_EnviaBytesI2C(0x586e, 0x18);
Camara_EnviaBytesI2C(0x586f, 0x1f);
Camara_EnviaBytesI2C(0x5870, 0x1c);
Camara_EnviaBytesI2C(0x5871, 0x16);
Camara_EnviaBytesI2C(0x5872, 0x10);
Camara_EnviaBytesI2C(0x5873, 0xf);
Camara_EnviaBytesI2C(0x5874, 0x13);
Camara_EnviaBytesI2C(0x5875, 0x1c);

```

```

Camara_EnviaBytesI2C(0x5876, 0x1e);
Camara_EnviaBytesI2C(0x5877, 0x17);
Camara_EnviaBytesI2C(0x5878, 0x11);
Camara_EnviaBytesI2C(0x5879, 0x11);
Camara_EnviaBytesI2C(0x587a, 0x14);
Camara_EnviaBytesI2C(0x587b, 0x1e);
Camara_EnviaBytesI2C(0x587c, 0x1c);
Camara_EnviaBytesI2C(0x587d, 0x1c);
Camara_EnviaBytesI2C(0x587e, 0x1a);
Camara_EnviaBytesI2C(0x587f, 0x1a);
Camara_EnviaBytesI2C(0x5880, 0x1b);
Camara_EnviaBytesI2C(0x5881, 0x1f);
Camara_EnviaBytesI2C(0x5882, 0x14);
Camara_EnviaBytesI2C(0x5883, 0x1a);
Camara_EnviaBytesI2C(0x5884, 0x1d);
Camara_EnviaBytesI2C(0x5885, 0x1e);
Camara_EnviaBytesI2C(0x5886, 0x1a);
Camara_EnviaBytesI2C(0x5887, 0x1a);
//;AWB
Camara_EnviaBytesI2C(0x5180, 0xff);
Camara_EnviaBytesI2C(0x5181, 0x52);
Camara_EnviaBytesI2C(0x5182, 0x11);
Camara_EnviaBytesI2C(0x5183, 0x14);
Camara_EnviaBytesI2C(0x5184, 0x25);
Camara_EnviaBytesI2C(0x5185, 0x24);
Camara_EnviaBytesI2C(0x5186, 0x14);
Camara_EnviaBytesI2C(0x5187, 0x14);
Camara_EnviaBytesI2C(0x5188, 0x14);
Camara_EnviaBytesI2C(0x5189, 0x69);
Camara_EnviaBytesI2C(0x518a, 0x60);
Camara_EnviaBytesI2C(0x518b, 0xa2);
Camara_EnviaBytesI2C(0x518c, 0x9c);
Camara_EnviaBytesI2C(0x518d, 0x36);
Camara_EnviaBytesI2C(0x518e, 0x34);

Camara_EnviaBytesI2C(0x518f, 0x54);
Camara_EnviaBytesI2C(0x5190, 0x4c);
Camara_EnviaBytesI2C(0x5191, 0xf8);
Camara_EnviaBytesI2C(0x5192, 0x04);
Camara_EnviaBytesI2C(0x5193, 0x70);
Camara_EnviaBytesI2C(0x5194, 0xf0);
Camara_EnviaBytesI2C(0x5195, 0xf0);
Camara_EnviaBytesI2C(0x5196, 0x03);
Camara_EnviaBytesI2C(0x5197, 0x01);
Camara_EnviaBytesI2C(0x5198, 0x05);
Camara_EnviaBytesI2C(0x5199, 0x2f);
Camara_EnviaBytesI2C(0x519a, 0x04);
Camara_EnviaBytesI2C(0x519b, 0x00);
Camara_EnviaBytesI2C(0x519c, 0x06);
Camara_EnviaBytesI2C(0x519d, 0xa0);
Camara_EnviaBytesI2C(0x519e, 0xa0);
//;D/S
Camara_EnviaBytesI2C(0x528a, 0x00);
Camara_EnviaBytesI2C(0x528b, 0x01);
Camara_EnviaBytesI2C(0x528c, 0x04);
Camara_EnviaBytesI2C(0x528d, 0x08);
Camara_EnviaBytesI2C(0x528e, 0x10);
Camara_EnviaBytesI2C(0x528f, 0x20);
Camara_EnviaBytesI2C(0x5290, 0x30);
Camara_EnviaBytesI2C(0x5292, 0x00);
Camara_EnviaBytesI2C(0x5293, 0x00);
Camara_EnviaBytesI2C(0x5294, 0x00);
Camara_EnviaBytesI2C(0x5295, 0x01);
Camara_EnviaBytesI2C(0x5296, 0x00);
Camara_EnviaBytesI2C(0x5297, 0x04);
Camara_EnviaBytesI2C(0x5298, 0x00);
Camara_EnviaBytesI2C(0x5299, 0x08);

```

```

Camara_EnviaBytesI2C(0x529a, 0x00);
Camara_EnviaBytesI2C(0x529b, 0x10);
Camara_EnviaBytesI2C(0x529c, 0x00);
Camara_EnviaBytesI2C(0x529d, 0x20);
Camara_EnviaBytesI2C(0x529e, 0x00);
Camara_EnviaBytesI2C(0x529f, 0x30);
Camara_EnviaBytesI2C(0x5282, 0x00);
Camara_EnviaBytesI2C(0x5300, 0x00);
Camara_EnviaBytesI2C(0x5301, 0x20);
Camara_EnviaBytesI2C(0x5302, 0x00);
Camara_EnviaBytesI2C(0x5303, 0x7c);
Camara_EnviaBytesI2C(0x530c, 0x00);
Camara_EnviaBytesI2C(0x530d, 0x10);
Camara_EnviaBytesI2C(0x530e, 0x20);
Camara_EnviaBytesI2C(0x530f, 0x80);
Camara_EnviaBytesI2C(0x5310, 0x20);

Camara_EnviaBytesI2C(0x5311, 0x80);
Camara_EnviaBytesI2C(0x5308, 0x20);
Camara_EnviaBytesI2C(0x5309, 0x40);
Camara_EnviaBytesI2C(0x5304, 0x00);
Camara_EnviaBytesI2C(0x5305, 0x30);
Camara_EnviaBytesI2C(0x5306, 0x00);
Camara_EnviaBytesI2C(0x5307, 0x80);
Camara_EnviaBytesI2C(0x5314, 0x08);
Camara_EnviaBytesI2C(0x5315, 0x20);
Camara_EnviaBytesI2C(0x5319, 0x30);
Camara_EnviaBytesI2C(0x5316, 0x10);
Camara_EnviaBytesI2C(0x5317, 0x00);
Camara_EnviaBytesI2C(0x5318, 0x02);
//;CMX
Camara_EnviaBytesI2C(0x5380, 0x01);
Camara_EnviaBytesI2C(0x5381, 0x00);
Camara_EnviaBytesI2C(0x5382, 0x00);
Camara_EnviaBytesI2C(0x5383, 0x1f);
Camara_EnviaBytesI2C(0x5384, 0x00);
Camara_EnviaBytesI2C(0x5385, 0x06);
Camara_EnviaBytesI2C(0x5386, 0x00);
Camara_EnviaBytesI2C(0x5387, 0x00);
Camara_EnviaBytesI2C(0x5388, 0x00);
Camara_EnviaBytesI2C(0x5389, 0xE1);
Camara_EnviaBytesI2C(0x538A, 0x00);
Camara_EnviaBytesI2C(0x538B, 0x2B);
Camara_EnviaBytesI2C(0x538C, 0x00);
Camara_EnviaBytesI2C(0x538D, 0x00);
Camara_EnviaBytesI2C(0x538E, 0x00);
Camara_EnviaBytesI2C(0x538F, 0x10);
Camara_EnviaBytesI2C(0x5390, 0x00);
Camara_EnviaBytesI2C(0x5391, 0xB3);
Camara_EnviaBytesI2C(0x5392, 0x00);
Camara_EnviaBytesI2C(0x5393, 0xA6);
Camara_EnviaBytesI2C(0x5394, 0x08);
//;GAMMA
Camara_EnviaBytesI2C(0x5480, 0x0c);
Camara_EnviaBytesI2C(0x5481, 0x18);
Camara_EnviaBytesI2C(0x5482, 0x2f);
Camara_EnviaBytesI2C(0x5483, 0x55);
Camara_EnviaBytesI2C(0x5484, 0x64);
Camara_EnviaBytesI2C(0x5485, 0x71);
Camara_EnviaBytesI2C(0x5486, 0x7d);
Camara_EnviaBytesI2C(0x5487, 0x87);
Camara_EnviaBytesI2C(0x5488, 0x91);
Camara_EnviaBytesI2C(0x5489, 0x9a);
Camara_EnviaBytesI2C(0x548A, 0xaa);
Camara_EnviaBytesI2C(0x548B, 0xb8);

Camara_EnviaBytesI2C(0x548C, 0xcd);

```

```

Camara_EnviaBytesI2C(0x548D, 0xdd);
Camara_EnviaBytesI2C(0x548E, 0xea);
Camara_EnviaBytesI2C(0x548F, 0x1d);
Camara_EnviaBytesI2C(0x5490, 0x05);
Camara_EnviaBytesI2C(0x5491, 0x00);
Camara_EnviaBytesI2C(0x5492, 0x04);
Camara_EnviaBytesI2C(0x5493, 0x20);
Camara_EnviaBytesI2C(0x5494, 0x03);
Camara_EnviaBytesI2C(0x5495, 0x60);
Camara_EnviaBytesI2C(0x5496, 0x02);
Camara_EnviaBytesI2C(0x5497, 0xB8);
Camara_EnviaBytesI2C(0x5498, 0x02);
Camara_EnviaBytesI2C(0x5499, 0x86);
Camara_EnviaBytesI2C(0x549A, 0x02);
Camara_EnviaBytesI2C(0x549B, 0x5B);
Camara_EnviaBytesI2C(0x549C, 0x02);
Camara_EnviaBytesI2C(0x549D, 0x3B);
Camara_EnviaBytesI2C(0x549E, 0x02);
Camara_EnviaBytesI2C(0x549F, 0x1C);
Camara_EnviaBytesI2C(0x54A0, 0x02);
Camara_EnviaBytesI2C(0x54A1, 0x04);
Camara_EnviaBytesI2C(0x54A2, 0x01);
Camara_EnviaBytesI2C(0x54A3, 0xED);
Camara_EnviaBytesI2C(0x54A4, 0x01);
Camara_EnviaBytesI2C(0x54A5, 0xC5);
Camara_EnviaBytesI2C(0x54A6, 0x01);
Camara_EnviaBytesI2C(0x54A7, 0xA5);
Camara_EnviaBytesI2C(0x54A8, 0x01);
Camara_EnviaBytesI2C(0x54A9, 0x6C);
Camara_EnviaBytesI2C(0x54AA, 0x01);
Camara_EnviaBytesI2C(0x54AB, 0x41);
Camara_EnviaBytesI2C(0x54AC, 0x01);
Camara_EnviaBytesI2C(0x54AD, 0x20);
Camara_EnviaBytesI2C(0x54AE, 0x00);
Camara_EnviaBytesI2C(0x54AF, 0x16);
Camara_EnviaBytesI2C(0x54B0, 0x01);
Camara_EnviaBytesI2C(0x54B1, 0x20);
Camara_EnviaBytesI2C(0x54B2, 0x00);
Camara_EnviaBytesI2C(0x54B3, 0x10);
Camara_EnviaBytesI2C(0x54B4, 0x00);
Camara_EnviaBytesI2C(0x54B5, 0xf0);
Camara_EnviaBytesI2C(0x54B6, 0x00);
Camara_EnviaBytesI2C(0x54B7, 0xDF);
Camara_EnviaBytesI2C(0x5402, 0x3f);
Camara_EnviaBytesI2C(0x5403, 0x00);
//;UV ADJUST
Camara_EnviaBytesI2C(0x5500, 0x10);

Camara_EnviaBytesI2C(0x5502, 0x00);
Camara_EnviaBytesI2C(0x5503, 0x06);
Camara_EnviaBytesI2C(0x5504, 0x00);
Camara_EnviaBytesI2C(0x5505, 0x7f);
//;AE
Camara_EnviaBytesI2C(0x5025, 0x80);
Camara_EnviaBytesI2C(0x3a0f, 0x30);
Camara_EnviaBytesI2C(0x3a10, 0x28);
Camara_EnviaBytesI2C(0x3a1b, 0x30);
Camara_EnviaBytesI2C(0x3a1e, 0x28);
Camara_EnviaBytesI2C(0x3a11, 0x61);
Camara_EnviaBytesI2C(0x3a1f, 0x10);
Camara_EnviaBytesI2C(0x5688, 0xfd);
Camara_EnviaBytesI2C(0x5689, 0xdf);
Camara_EnviaBytesI2C(0x568a, 0xfe);
Camara_EnviaBytesI2C(0x568b, 0xef);
Camara_EnviaBytesI2C(0x568c, 0xfe);
Camara_EnviaBytesI2C(0x568d, 0xef);
Camara_EnviaBytesI2C(0x568e, 0xaa);

```

```
Camara_EnviaBytesI2C(0x568f, 0xaa);  
}
```