UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

DISEÑO DE UNA CÁMARA DIGITAL

TRABAJO FIN DE GRADO

Septiembre - 2020

AUTOR: D. Juan Manuel Ferrández Aznar

DIRECTOR: D. José Antonio Carrasco Hernández

Agradecimientos

Me gustaría agradecer principalmente el haber llegado hasta aquí a mi familia y, en concreto, a mis padres y mi hermana, por haberme inculcado tanta curiosidad desde pequeño, junto a su apoyo incondicional y su perseverancia para que nunca me rindiera, ha hecho posible que haya llegado hasta aquí.

También me gustaría hacer una mención especial a mi tutor, José Antonio, y a los profesores del grado, por poder aprender de ellos durante todos estos años y haberme descubierto un mundo de conocimientos nuevos, cuando entre al grado no imaginaba que al salir pudiera lograr proyectos como el de la presente memoria.

Por último, quisiera darles las gracias a mis amigos, especialmente a Marta y Julia, por haber estado a mi lado siempre que les he necesitado.

Resumen

En el presente trabajo explica el diseño y desarrollo de una cámara para la captación de imágenes. Cubre tanto la parte de diseño del hardware como el software que lo controla. Se investigan las distintas tecnologías para captar y procesar las imágenes, a partir de ahí se seleccionan todos los componentes electrónicos necesarios para su funcionamiento, con todos ellos se diseña el circuito y se monta una PCB. Por último, se prueba que las distintas partes del montaje funcionan y se desarrolla el software.

Índice general

Índice de figuras

Índice de tablas

1. INTRODUCCIÓN

1.1. MOTIVACIÓN DEL PROYECTO

El mundo de la fotografía ha evolucionado exponencialmente en los últimos 20 años. Apenas se utilizan las cámaras analógicas y las cámaras digitales están presentes en nuestro día a día. Con el surgimiento de las redes sociales y los smartphones la fotografía ya no se utiliza únicamente con fines profesionales ni para guardar recuerdos, se ha convertido en una herramienta social indispensable en la vida de muchas personas.

El hecho de ser una tecnología tan presente y utilizada, junto a la curiosidad por comprender como la luz es captada por el sensor, procesada y transformada para poder guardar y visualizar posteriormente la imagen ha sido una de las motivaciones de este proyecto.

La otra gran motivación ha sido la posibilidad que se me ha otorgado de poner en práctica todos los conocimientos aprendidos a lo largo de las distintas asignaturas del Grado, al diseñar y desarrollar en su totalidad un sistema embebido de tiempo real y relativamente complejo, con la libertad total de estudiar y seleccionar los componentes y tecnologías que consideré más adecuadas para el proyecto.

1.2. OBJETIVOS DEL PROYECTO

La presente memoria describe el proceso de diseño y desarrollo de una cámara digital. La cámara digital es un sistema electrónico embebido capaz de captar imágenes y vídeo, procesarlas y transformarlas en formatos estándares, guardarlas en una memoria externa, así como, transferirlas a un ordenador para su posterior visualización.

Los distintos objetivos del proyecto son:

- El sistema ha de ser capaz de tomar fotografías con una resolución 5MP.

- El sistema convertirá las fotografías a formato JPEG y las guardará en una memoria microSD externa.
- El usuario decidirá el momento en el que tomar la fotografía mediante un botón.
- El usuario podrá seleccionar mediante botones la resolución de imagen deseada.
- El usuario podrá seleccionar mediante botones el tiempo de exposición deseado.
- La cámara informará de su estado, resolución seleccionada, tiempo de exposición, etc., al usuario mediante LEDs.
- El sistema se podrá comunicar con un ordenador mediante una interfaz UART para transferirle la imagen, así como, modificar la resolución y tiempo de exposición de la imagen e indicarle el momento en el que tomar la foto.
- El sistema ha de ser capaz de funcionar con una única fuente de alimentación de 5V.
- El hardware ha de ser lo suficiente rápido como para en un futuro poder soportar la grabación de vídeo en alta resolución HD (1080p) y una tasa 30 imágenes por segundo.

1.3. METODOLOGÍA DEL PROYECTO

Los pasos seguidos para el desarrollo de trabajo de fin de grado son:

- **Conocer el funcionamiento de los sensores de imagen CMOS y de las cámaras digitales.** Este paso consiste en aprender el funcionamiento de los distintos tipos de sensores y procesadores de imagen, así como las alternativas existentes, con el fin de comprender el proceso de como una imagen es digitalizada y guardada en memoria.

- **Seleccionar el sensor de imagen y la tecnología para controlar dicho sensor de imagen.** Dependiendo del tipo de sensor que se utilice, la

tecnología para controlarlo y leerlo varía, por tanto, hay que explorar las diferentes alternativas existentes para elegir la tecnología que más se adapte al proyecto.

- **Seleccionar los componentes electrónicos.** La cámara digital no solo se basa de un sensor de imagen y un procesador, necesita otros componentes para funcionar: memoria externa para guardar la imagen, memoria RAM para procesarla, conectores para comunicarse con el ordenador, adaptadores de voltaje entre las distintas partes del circuito, botones y LEDs para interactuar con el usuario, etc.

- **Calcular y diseñar el esquema electrónico.** Una vez seleccionados los componentes de la cámara hay que diseñar el circuito y realizar la conexión entre todos ellos asegurándonos de que no haya ninguna incompatibilidad con los niveles de tensión.

- **Diseñar y montar la placa de circuito impreso.** El circuito hay que implementarlo en una placa de circuito de impreso (PCB). Consiste en posicionar todos los componentes y conectarlos entre ellos, respetando las reglas de diseño impuestas por el fabricante de PCB y llevando cuidado de que no se produzcan futuros problemas de interferencias o ruidos eléctricos.

- **Probar las distintas partes de la cámara.** Después de fabricar y montar la cámara hay que testear las distintas partes y componentes para comprobar que no se ha producido ningún fallo en alguna de las fases anteriores.

- **Desarrollar el software del microcontrolador.** Aunque esté montada la placa hay que diseñar el software que funciona sobre el microcontrolador. Este software ha de comunicarse con el sensor para captar las imágenes, aplicarle los algoritmos y transformaciones necesarias, guardarlas en

memoria, comunicarse con el usuario a través de la interfaz UART, botones y LEDs, etc.

1.4. CONTENIDO DE LA MEMORIA

En esta sección se explica resumidamente el contenido de cada capítulo y anexo de esta memoria:

- **Capítulo 1: Introducción.** Se explica la motivación del proyecto, sus objetivos y un breve resumen de la metodología y contenidos de la memoria.

- **Capítulo 2: Estado del arte**. Ofrece una descripción del estado actual de la tecnología existente en los sensores de imagen y procesadores de imagen.

- **Capítulo 3: Selección de componentes.** Se exponen tanto los criterios como los cálculos que se han utilizado para elegir los distintos componentes que forman la cámara.

- **Capítulo 4: Diseño de la PCB.** Se describe el proceso y criterios que se ha seguido a la hora de elegir la disposición y conexión de los componentes electrónicos sobre la placa de circuito impreso.

- **Capítulo 5: Comprobación del montaje.** Se comprueba que los componentes elegidos y su montaje funcionan de la forma esperada.

- **Capítulo 6: Prueba del sensor.** Al igual que en el capítulo anterior se comprueba que el sensor de imagen se comunica correctamente con el microcontrolador y toma las imágenes correctamente.

- **Capítulo 7: Software.** Se presentan los diagramas de flujo del programa que utiliza el microcontrolador y además se comenta la estructura y las distintas funciones del código.

- **Capítulo 8: Conclusiones y trabajos futuros.** Se comparan los resultados obtenidos tras finalizar el proyecto con los objetivos. Además, se presentan futuros desarrollos para ampliar este proyecto.

- **Anexo A: Esquemas eléctricos.** Se muestran los esquemas eléctricos.
- **Anexo B: PCB.** Se presenta la imagen de las distintas capas de la PCB.
- **Anexo C: Código fuente.** Se presenta el código utilizado en el microcontrolador.

2. ESTADO DEL ARTE

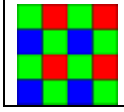
2.1. SENSOR DE IMAGEN

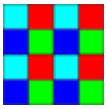
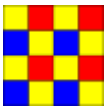
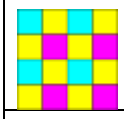
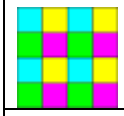
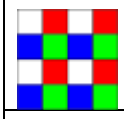
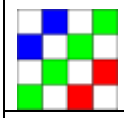
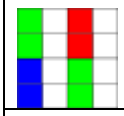
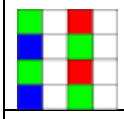
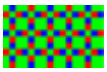
El sensor es el dispositivo electrónico encargado de transformar la luz incidente en una señal eléctrica para su posterior procesado. Está formado por una matriz de fotodetectores llamados píxeles. Dependiendo de su tecnología de fabricación podemos encontrar sensores CCD o sensores CMOS ambas basadas en el efecto fotoeléctrico. [1]

Este efecto es la emisión de electrones u otros portadores libres de carga cuando la luz golpea un material. En el caso de que el material sea una unión de semiconductores PN (fotodiodo) los electrones liberados generan huecos o cargas positivas. Cuando el fotodiodo se encuentra polarizado en inversa (ánodo al potencial negativo y cátodo al positivo) estos huecos permiten que una corriente fluya desde cátodo a ánodo. Al ser la corriente proporcional a la cantidad de luz incidente esta resulta fácil de medir. [2]

Si la intensidad de luz es independiente de la longitud de onda, entonces para separar y obtener y la información del color se utiliza una matriz de filtros de color o CFA (*color filter array*). Estos filtros permiten pasar en cada píxel solo una determinada longitud de onda, es decir, un solo color. Para recuperar todos los colores la imagen se utiliza un algoritmo de interpolación cromática o *demosaicing*. [3]

Tabla 1: Ejemplos de matrices de filtros de color[F1]

Patrón	Nombre	Descripción	Tamaño del patrón (en píxeles)
	Filtro Bayer	Filtro RGB más común. Con un azul, un rojo y dos verdes.	2×2

	Filtro RGBE	Similar al Bayer con uno de los filtros verdes modificado a "esmeralda"; usado en algunas cámaras Sony.	2×2
	Filtro RYYB	Modificando el verde por el amarillo del filtro Bayer. Utilizado en el smartphone Huawei P30.	2×2
	Filtro CYYM	Un cian, dos amarillos y uno magenta; usado en algunas cámaras Kodak.	2×2
	Filtro CYGM	Uno cian, uno amarillo, uno verde y uno magenta; usado en algunas cámaras.	2×2
	RGBW Bayer	RGBW tradicional, similar a los patrones Bayer y RGBE.	2×2
	RGBW #1	Filtros RGBW de Kodak, con 50% blanco.	4×4
	RGBW #2		
	RGBW #3		2×4
	X-Trans	Filtro RGB específico de Fujifilm con un gran patrón, para reducir el efeto Moirè.	6×6

2.1.1. SENSORES CCD

Los sensores de dispositivo de carga acoplada CCD (*charge-coupled device*) o sensor de píxeles pasivos (PPS) consisten en un fotodiodo con el cátodo dividido en canales aislantes, sobre el que se han depositado distintas tiras conductoras o electrodos. Cada píxel está confinado por dos canales aislantes y tres electrodos. Todos los píxeles separados por los dos mismos canales forman una

columna y los que comparten electrodos se les llama filas. Siendo el número de filas por el número de columnas la resolución del sensor.

El ánodo y dos de los electrodos de cada fila se polarizan negativamente y el otro se polariza positivamente. Cuando el obturador se abre, permite que la luz incida en el fotodiodo fluyendo la carga hacía el electrodo positivo.

Alternando el voltaje en los distintos electrodos de cada fila la carga es desplazada hasta el borde del sensor donde se encuentra un registro de desplazamiento serie. Este registro envía la carga hacia un amplificador con un conversor ADC (Analog-to-Digital Converter) conectado al bus del procesador. Por tanto, el procesador lee la imagen columna a columna y fila a fila hasta que recoge todas las cargas y el sensor está listo para otra exposición. [2] [4]

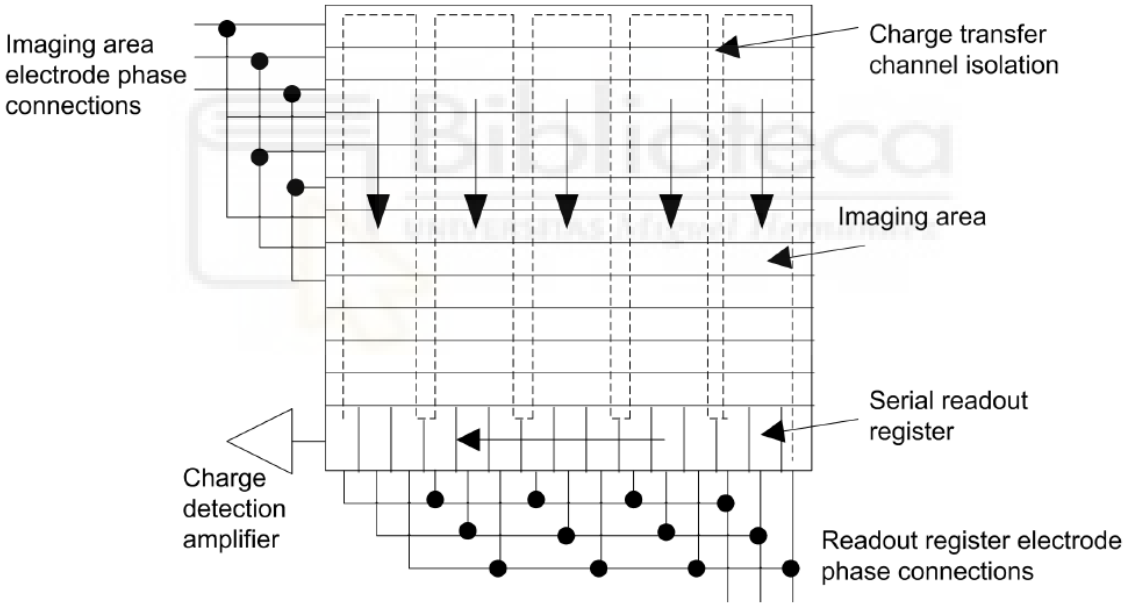


Figura 1: Arquitectura de una matriz completa de tipo CCD. [F1]

Las ventajas de este tipo de sensores es que la imagen no tiene apenas ruido.

Los problemas es que en imágenes grandes en cada desplazamiento se pierde carga, es un método lento ya que hay leer y desplazar los píxeles de uno en uno y para mover las cargas hay que aplicar un potencial lo cual no es energéticamente muy eficiente. [2]

2.1.2. SENSORES CMOS

Los sensores CMOS o sensor de píxeles activos (APS) también se basan en el fotoeléctrico pero el fotodiodo se fabrica con tecnología CMOS. Esto permite incorporar a cada píxel transistores y no tener que utilizar el acoplamiento de carga para transportar las cargas. [2]

Cada píxel esta formados por el fotodiodo y 3 transistores, el de sensado, el de selección de fila y el de reset. El transistor de sensado actúa como un buffer (seguidor de fuente) amplificando el voltaje de cada píxel y conservando la carga acumulada. Las puertas de todos los transistores de selección de una misma fila están conectadas, al igual que todas las fuentes de una misma columna. Al igual que en los sensores CCD existe un registro de desplazamiento serie, cuando se activa el transistor de selección de una fila todas las cargas de esa fila se introducen en el registro. Para eliminar toda la carga del fotodiodo se utiliza el transistor de reset, cuando este conduce, la tensión en el fotodiodo pasa a ser V_{rd}. [5]

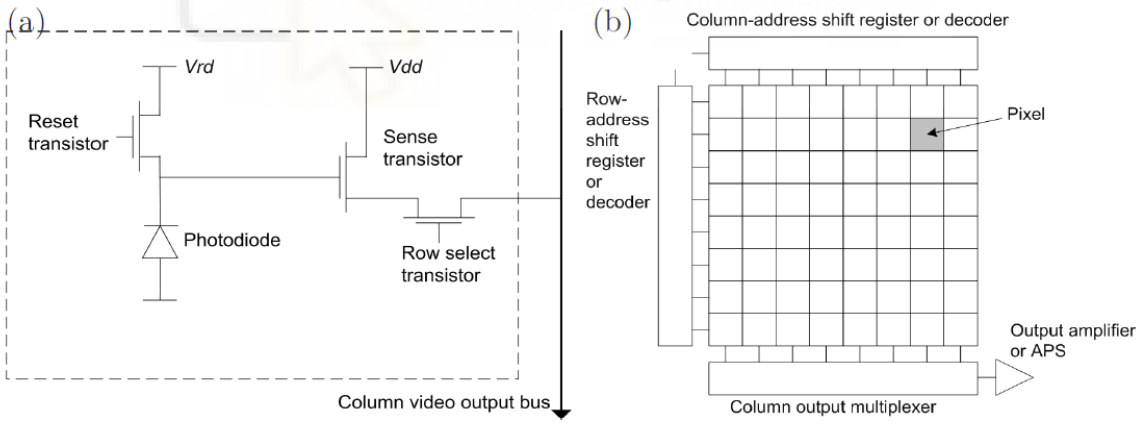


Figura 2: (a) Sensor activo con 3 transistores CMOS. (b) Arquitectura de la matriz. [F1]

Las ventajas son un menor consumo eléctrico, un coste económico menor al no necesitar tantos componentes externos, si prescindimos del registro serie se pueden leer varias filas a la vez, se puede integrar el conversor ADC en el mismo integrado reduciendo costes a cambio de una menor linealidad y mayor ruido

(efecto de iluminación) ya que los amplificadores de los sensores de imagen se suelen encontrar en la parte trasera del sensor.

Las desventajas están en un mayor ruido, menos superficie receptora de la luz por píxel y, si no se utiliza el mismo obturador para todos los píxeles (*global shutter*) y se utiliza un obturador para cada fila las imágenes que se mueven rápidamente pueden aparecer deformadas, efecto que se conoce como *rolling shutter*. [6]

2.2. PROCESADOR DE IMAGEN

El procesador de la cámara o procesador de imagen es el circuito lógico encargado principalmente de controlar los parámetros del sensor (por ejemplo, la resolución o el tiempo de exposición), leer la luz de cada píxel y aplicar el algoritmo de interpolación para recuperar el color de la imagen.

En los últimos años han evolucionado de manera muy significativa realizando cada vez más funciones como controlar el autoenfoque, corregir las imperfecciones de la lente (como el sombreado de blancos), conseguir imágenes de alto rango dinámico (HDR) combinando múltiples tomas, reducción de ruido o detección de caras u objetos. Además, se encarga de codificar las imágenes en un formato de imagen (como jpg o png) o vídeo (h.264 o MPEG-4) con o sin compresión. [7]

Por tanto, el procesador de imagen es un circuito complejo que debe ser capaz de realizar todas estas operaciones a alta velocidad y con muy baja latencia. El procesador de una cámara que graba vídeo con resolución *Full HD* (1920 x 1080 pixeles) a 30 imágenes por segundos tiene que procesador más de 62 millones de píxeles por segundo.

2.2.1. PROCESADOR DE SEÑAL DE IMAGEN (ISP)

Un procesador de señal de imagen o ISP (*Image Signal Processor)* es un ASIC (circuito integrado de aplicación específica) basado en un procesador digital de señal (DSP) especializado en tratar las imágenes obtenidas por el sensor.

Los sistemas capaces de realizar todas las tareas nombradas anteriormente son dispositivos con arquitectura de procesador multinúcleo que, además, suelen utilizar técnicas de computación paralela como SIMD (una instrucción, múltiples datos) o MIMD (múltiples instrucciones, múltiples datos) para incrementar la velocidad y eficiencia. Para integrar estos sistemas en dispositivos embebidos suelen presentarse como un sistema en chip (SoC). [8]

Las grandes compañías de electrónica tienen algunos modelos de ISPs. En el caso de *Texas Instruments* tenemos el *TMS320DM355* o *TMS320DSC21* ambos basados en la familia de DSP *TMS320* [9] o en el caso de *STMicroelectronics* el modelo *STV0991* [10]. Al ser diseños orientados a productos muy específicos con grandes volúmenes de fabricación es difícil tener acceso a ellos.

Figura 3: Procesador de imagen *TMS320DSC21* en *Olympus D-395*.

Por último, la significativa evolución de las tareas que realiza el ISP ha tendido a que los diseños de los procesadores sean propios de cada fabricante de cámaras o móviles como por ejemplo *DIGIC* de *Canon*, *EXPEED* de *Nikon*, *BIONZ* de *Sony*, *EXR* de *Fujifilm*, *TruePic* de *Olympus*, *VENUS Engine* de *Panasonic*, *DRIMe* de *Samsung* o *ImageSense* de *HTC*. [11]

2.2.2. FPGA

Una FPGA (*field-programmable gate array*) o matriz de puertas programables es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento, mediante un lenguaje de descripción especializado. Cada bloque lógico contiene una o más LUT (*Look Up Table*) y biestables, estos bloques pueden implementar cualquier función lógica (de unos pocos bits) o combinarse para crear registros. Por tanto, la FPGA en su totalidad se utiliza para implementar circuitos asignando las puertas lógicas y registros en bloques lógicos. [12]

Al principio, las FPGAs se desarrollaron para comprobar los diseños de hardware, pero con el aumento de la velocidad y densidad de los bloques lógicos se puede implementar el mismo circuito que en un ASIC, con la ventaja económica de no tener que mandar a fabricar grandes volúmenes de integrados. [13] Además, en los últimos años han surgido integrados system-on-chip que incluyen en el mismo silicio una FPGA y un microcontrolador o microprocesador de propósito general de uno o varios núcleos, la familia *ZYNQ* de *XILINX* utiliza microprocesadores *ARM Cortex-A9*, *ARM Cortex-A53* y *ARM Cortex-R5* [14] y *SmartFusion* de *Microsemi* utiliza *ARM Cortex-M3* [15].

En el campo del procesamiento de imágenes y visión por ordenador, las compañías vendedoras de FPGA ofrecen IP cores (*Intellectual Property core*) diseñados exclusivamente para el procesamiento de imagen [16][17]. Además, hay proyectos en los que se han utilizado para acelerar el seguimiento de puntos

en tiempo real [18], estéreo visión [19], detección de objetos basada en color [20] y compresión de vídeo e imágenes [21].

Por el contrario, ofrecen un consumo mayor y una velocidad menor (mayor retardo en las puertas lógicas y una velocidad de reloj menor) respecto al mismo circuito en un ASIC y son más caras cuando se hay que fabricar grandes volúmenes. Pese a que las compañías ofrecen IP cores, a no ser que coincidan exactamente con el procesador de imagen que queremos tendremos que diseñar otras partes con un lenguaje de descripción de hardware normalmente VHDL o Verilog. Esto repercute en un mayor tiempo y coste de desarrollo frente a otras soluciones.

2.2.3. MICROCONTROLADOR

Un microcontrolador (MCU) es un circuito integrado que incluye un procesador (CPU) con memoria para datos (RAM estáticas), memoria del programa no volátil (*flash*) y periféricos integrados. Los periféricos pueden ser protocolos de comunicación como SPI, I2C, USB o Ethernet; buses como PCI, tarjetas de memoria flash o memorias externas; interfaces analógicas como comparadores, ADCs multiplexados, DACs y entradas para sensores de imagen y sonidos; e interfaces de propósito dedicado como PWM, LCD drivers, audio digital o WiFi [22].

Los MCUs comenzaron en los años 70 utilizando pequeños y baratos microprocesadores de 4 y 8 bits [23]. En la actualidad siguen existiendo arquitecturas de 8 bits (*AVR* de *Atmel* o *PIC* de *Microchip*) pero el standard está marcado por microprocesadores de 32bits, siendo la familia *ARM Cortex-M* una de las más populares y utilizada en los últimos años por los distintos fabricantes de microcontroladores [23].

La gama alta de esta familia, núcleos *ARM Cortex-M4* y *ARM Cortex-M7*, incluyen instrucciones específicas de procesadores digitales de señal (DSP) y unidad de coma flotante (FPU) de precisión simple (SP-FPU) o doble (DP-FPU)

para ofrecen un alto rendimiento en el procesamiento de voz, audio e imágenes, aplicaciones de control industrial o aplicaciones de aprendizaje de máquinas [24].

Al igual que ha ocurrido en las FPGA, existen microcontroladores que incluyen varios núcleos e incluso microprocesadores de propósito general capaces de utilizar sistemas operativos de alto nivel como Linux [25].

Como alternativa a las FPGAs y procesadores de imagen, un microcontrolador ofrece un rendimiento menor y menos funcionalidades a la hora de procesar las imágenes, pero por el contrario ofrecen un tiempo de desarrollo y consumo menores que las FPGAs. Además, como se ha explicado en el primer párrafo, ofrecen periféricos para todo tipo de funcionales que no incluyen los procesadores de imagen y que habría que desarrollar para FPGAs.

2.2.4. SBC

Un ordenador de placa reducida o *single board computer* (SBC), es una pequeña placa que incluye todo lo necesario para que un ordenador funcione, microprocesador, RAM, memorias, puertos de E/S, tarjeta de vídeo y audio, etc. Algunas de las placas más conocidas son la familia *Raspberry PI* [26] y la familia *BeagleBoard* de *Texas Instruments* [27].

Figura 4: Ejemplo de un SBC, Raspberry Pi Model B. [F2]

Al igual que en el caso de los microcontroladores algunas placas incluyen un puerto para controlar un sensor de imagen ofreciendo un rendimiento mayor que estos, pero inferior a los procesadores de imagen y FPGAs.

Las contras es que la interfaz del sensor tiene que ser compatible con la SBC tanto a nivel de hardware (buses, conectores, voltajes, etc.) como en el tema de drivers, ya que estas placas suelen utilizar un sistema operativo de alto nivel (Linux, Windows, Android…) ya que no resulta trivial desarrollar drivers para sistemas operativos. Por último, a la hora de utilizarlos en un diseño embebido compacto es difícil pues hay que realizar una placa que se conecte a la SBC resultando en un diseño aparatoso y poco compacto o realizar una placa que incluya los módulos del SBC, lo cual complica mucho el desarrollo.

3. SELECCIÓN DE COMPONENTES

3.1. SELECCIÓN DEL SENSOR CMOS

El sensor propuesto para el proyecto fue el *MT9P031* de *On Semiconductor* por su facilidad para obtenerlo a bajo coste y una resolución más que suficiente de 5 megapíxeles.

Aun así, se evaluaron otras opciones, pero lamentablemente, a través de nuestros distribuidores solo se tiene acceso a la empresa de sensores *On Semiconductor*. Hubiera sido interesante también evaluar las opciones de otros fabricantes como *OmniVision*.

Tabla 2: Sensores de imagen de OnSemi [1]

Producto	Resolución (MP)	Imágenes por segundo	Formato óptico	Interfaz salida	Encapsulado
AR0543	5	15	1/4 inch	-	ODCSP-54
MT9P001	5	15	1/2.5 inch	-	ILCC-48
MT9P006	5	15	1/2.5 inch	Paralela	ILCC-48
MT9P031	5	15	1/2.5 inch	Paralela	ILCC-48
MT9P401	5	15	1/2.5 inch	Paralela	ILCC-48
AR0521	5.1	60	1/2.5 inch	Multi	PLCC-52
AR0522	5.1	60	1/2.5 inch	Multi	PLCC-52
PYTHON5000	5.3	45 100	1 inch	LVDS	LBGA-128 LCC-84
VITA5000	5.3	75	1 inch	LVDS	LCC-68

XGS 5000	5.3	132	2/3 inch	HiSPi™	ILGA163 16x16

En la tabla 2 se ha recogido los distintos modelos de sensores de imagen del fabricantes *OnSemi* con una resolución cercana a 5MP [28]. Solo se tiene acceso a los *datasheet* de los siguientes modelos: *AR0543*, *MT9P001*, *MT9P031*, *MT9P401*, *PYTHON5000* y VITA5000.

Los sensores *PYTHON5000* y *VITA5000* están pensados para utilizar señales diferenciales de bajo voltaje (LVDS) y, además, utilizan encapsulados de 84 y 68 pines respectivamente, estas dos cosas complican en exceso el diseño de la cámara.

El *AR0534* utiliza la salida de datos *MIPI CSI* (Interfaz Serie para Cámaras), esta interfaz es una especificación dentro del estándar *MIPI* que define la interfaz entre un sensor y un procesador anfitrión. Esta especificación consta de varias versiones, cada una con distintas capas [29]. Esto requiere una gran investigación posterior y encontrar un procesador que soporte la misma especificación que el sensor, por ello se ha descartado.

Se han comparado las características de los tres sensores restantes (*MT9P001*, *MT9P031* y *MT9P401*) y no se han observado diferencias en los *datasheet*. Por lo que se ha escogido el sensor propuesto *MT9P031*. Otra de las razones por las que escoger este sensor es que ha sido ampliamente utilizado y probado en distintos proyectos, por ejemplo, en el famoso móvil *Nokia N95* se utilizaba [30] y también en el proyecto *Frankencamera* [31].

3.2. SELECCIÓN DEL PROCESADOR DE IMAGEN

Entre los distintos tipos de procesadores presentados en el apartado 2.2, los primeros que se descartaron fueron los ISP, ya que como se explicó la tendencia es que son ASICs hechos a medida para los distintos fabricantes de cámaras

digitales, y no se tiene mucha información sobre ellos (*datasheet,* herramientas de programación, etc.).

Los ordenadores embebidos o SBC también se descartaron por los motivos dados en el apartado 2.2.4.: se requiere que el hardware del SBC sea compatible con el sensor *MT9P031,* habría que desarrollar los drivers y además el diseño no quedaría embebido en una sola PCB.

Las FPGAs desde un punto académico resultan lo más interesante pues hay que realizar todo el diseño digital del hardware: el periférico encargado los datos del sensor, el puerto I2C, los algoritmos de *demosaicing* y codificación JPEG, el módulo para leer y escribir datos de la tarjeta SD con una partición *FAT32*, etc.

Todo esto llevaría mucho tiempo y el trabajo se alargaría mucho, por ello, en el caso de utilizar una FPGA es más lógico a optar por un System-on-Chip. Específicamente se valoró la opción de usar una placa de desarrollo o evaluación de *Zedboard* del fabricante *Xilinx.*

El SoC *XC7Z020* que incluye esta placa cuesta más de 100 euros [32] por lo que no es asequible realizar un diseño a medida en una PCB y, asimismo, diseñar una placa con memoria *DDR SDRAM* (*Double Data Rate Synchronous Dynamic Random-Access Memory*) no resulta trivial debido a la alta frecuencia que utilizan las señales de este tipo de memoria.

El *XC7Z020*, incluye dos núcleos *Cortex-A9* a 800MHz más una FPGA con 85000 células lógicas [33]. Sobre estos núcleos se puede ejecutar un sistema operativo Linux que va grabado en una tarjeta SD. Con los IP cores disponibles de *Xilinx* y la potencia de cálculo de los dos núcleos, habría que desarrollar muy pocos módulos.

Sería perfecto para nuestra aplicación, excepto porque para comunicar los datos desde el sensor hasta los procesadores ARM hay que utilizar el controlador *DMA* (*Direct Memory Access*) y desarrollar unos drivers para Linux compatibles con

este controlador. Como en el caso del SBC la dificultad del software sumado a que el hardware no sería compacto en una sola PCB hace que se descarte esta posibilidad.

La única alternativa viable que nos queda es un microcontrolador. En concreto se valoraron los modelos con interfaz para cámara de la familia *STM32* de *STMicroelectronics* y la familia de *NXP*, *i.MX RT Crossover*, como se recoge en las tablas 3 y 4, respectivamente [34][35].

Tabla 3: Microcontroladores STM32 con interfaz de cámara

Modelo	CPU	Memoria RAM	Frecuencia máx. cámara	Codificación JPEG	DMA2D
STM32F2x7	Cortex-M3 @ 120 MHz	128 KB	48 MHz	No	No
STM32F407/417	Cortex-M4 @ 168 MHz	192 KB	54 MHz	No	No
STM32F427/437	Cortex-M4 @ 180 MHz	256 KB	54 MHz	No	Sí
STM32F429/439	Cortex-M4 @ 180 MHz	256 KB	54 MHz	No	Sí
STM32F446	Cortex-M4 @ 180 MHz	128 KB	54 MHz	No	No

STM32F469/479	Cortex-M4 @ 180 MHz	384 KB	54 MHz	No	Sí
STM32F7x5	Cortex-M7 @ 216 MHz	512 KB	54 MHz	No	Sí
STM32F7x6	Cortex-M7 @ 216 MHz	320 KB	54 MHz	No	Sí
STM32F7x7	Cortex-M7 @ 216 MHz	512 KB	54 MHz	Sí	Sí
STM32F7x8/7x9	Cortex-M7 @ 216 MHz	512 KB	54 MHz	Sí	Sí
STM32L4x6	Cortex-M4 @ 80 MHz	320 KB	32 MHz	No	Sí
STM32H7x3	Cortex-M7 @ 400 MHz	1 MB	80 MHz	Sí	Sí

Tabla 4: Microcontroladores NXP con interfaz de cámara

Modelo	CPU	Encapsulado	Memoria RAM	Interfaces cámara

i.MX RT1170	Cortex-M7 @ 1GHz + Cortex-M4 @ 400MHz	289 BGA	2MB	Paralelo, MIPI
i.MX RT1064	Cortex-M7 @ 600MHz	196 BGA	1MB	Paralelo
i.MX RT1060	Cortex-M7 @ 600MHz	196 BGA	1MB	Paralelo
i.MX RT1050	Cortex-M7 @ 600MHz	196 BGA	512KB	Paralelo

Los *i.MX RT Crossover*, son unos microcontroladores muy interesantes para esta aplicación por su alta velocidad de procesador, 200MHz por encima del modelo más potente de *ST* el *STM32H7x3*. La desventaja es que solo están disponibles en encapsulado de BGA, esto obliga a diseñar una PCB con más capas lo que dificulta el rutado, además, de dificultar las fase de montaje y prueba, ya que no permite tener acceso a los pines directamente.

Por estas razones se ha escogido el microcontrolador STM32 más potente que existe, el *STM32H743*, la diferencia con el *STM32H753* es que este último incluye medidas de seguridad avanzadas como módulos de criptografía en hardware [36] que no nos interesan en este proyecto. Este microcontrolador está disponible en varios encapsulados tanto planos (QFP) o de bolas (BGA) [37], se ha elegido el TQFP144 porque con 144 pines es más que suficiente para enrutar todo el hardware. Además de la facilidad de diseño y montaje, tiene características interesantes para el procesamiento de imagen como, por ejemplo, el codificador JPEG en hardware o el *DMA2D* que permite convertir imágenes de un espacio de color YCbCr a RGB reduciendo así la carga en el procesador y permitiendo una mayor velocidad [38].

El periférico de la cámara digital puede funcionar hasta una frecuencia de 80MHz, es decir, a 80MPíxeles/s. Esta velocidad permite grabar vídeo 1080p o

Full HD a una tasa de $\frac{80\ MPíx}{s} = \frac{80 \times 10^6 Píx}{s} \times \frac{1\ Frame}{1920 \times 1080\ Pixel} \approx 38.6\ FPS$, muy por encima de los 30FPS que se requieren en el proyecto.

3.3. MEMORIA RAM

El microcontrolador no tiene suficiente memoria RAM interna para poder guardar los datos del sensor con lo que se debe utilizar memoria RAM externa. El *STM32H743* acepta tres tipos de RAM distintas: SRAM, PSRAM y DRAM (SDRAM/Mobile LPSDR SDRAM).

La SRAM o RAM estática se caracteriza por estar basada en flip-flops y no requerir una señal de reloj, esto permite un acceso a los datos asíncrono con una velocidad de lectura/escritura bastante alta. Por el contrario, la DRAM o RAM dinámica utiliza condensadores y necesita una señal de reloj tanto para leer o escribir datos (acceso síncrono) como para refrescar periódicamente los condensadores evitando que se descarguen y se pierdan los datos [39]. Por último, la PSRAM o RAM pseudo-estática es una memoria DRAM que incluye la lógica del reloj y refresco en la propia memoria para trabajar como si fuera una la RAM estática [40].

Para esta aplicación, no necesitamos una velocidad de memoria en el orden de los gigahercios, por tanto, se ha elegido una memoria DRAM que nos ofrece una menor velocidad a cambio de un menor precio y mayor consumo (al necesitar la señal de reloj constantemente).

El modelo escogido ha sido el *MT48LC8M16A2 de Micron*. Se trata de una memoria con 4 bancos de memoria, cada uno de 2 millones por 16 bits. Se ha elegido que el ancho de bus de los datos sea de 16 bits, en vez de 8 o 32 bits, porque el tamaño de cada píxel del sensor es de 12 bits, lo que permite escribir un píxel entero accediendo solo una vez a la memoria.

En concreto se ha escogido este modelo de *Micron* porque la placa de evaluación *32f746GDiscovery* utiliza una memoria *MT48LC4M32B2B5* [41], de la misma

familia que la *MT48LC8M16A2*. De esta forma estamos seguros de que es completamente compatible en voltajes, velocidad, tiempos y latencias de espera, etc.

3.4. REGULADORES DE TENSIÓN

El microcontrolador, la tarjeta microSD y la memoria RAM trabajan a 3.3V, mientras que el sensor requiere de 3 voltajes: entre 1.7V y 3.1V para alimentar los pines de entrada y salida, 1.8V para la lógica digital y entre 2.6 y 3.1V para la circuitería analógica [42].

Para reducir de los 5V de la alimentación, requeridos por el proyecto, a los 3.3V se utiliza un LDO. Este LDO ha de soportar la suma de las corrientes de alimentación de cada componente que funciona a 3.3V además de la corriente del LDO de 1.8V que se utiliza para el sensor de imagen ya que van conectado en serie.

Aunque en el datasheet del microcontrolador indique un consumo de hasta 620mA [43], esto es el consumo máximo absoluto. La realidad es que su consumo depende de la frecuencia del reloj y de los periféricos habilitados, trabajando normalmente con mucha menos corriente. Con la herramienta *STM32CubeMX* se ha calculado el consumo del microcontrolador a la frecuencia máxima con todos los periféricos que se utilizan en el proyecto habilitados: DCMI, JPEG, DMA, I2C, etc. En la figura 5, se muestra que ha calculado un valor de aproximadamente 83mA con una frecuencia de reloj máxima.

En el datasheet de la memoria RAM indica que tiene un consumo máximo de 170mA [44] mientras que el de la tarjeta uSD es de 300mA [45]. En la parte de los 1.8V consume 35 mA [42].

El consumo total del LDO es, por tanto, de $I = 83mA + 170mA + 300mA + 35mA \approx 600mA$. Para no ir al límite se ha elegido uno de por lo menos, de un

33% más de la corriente necesaria, es decir de 800mA. El modelo ha sido el clásico *TLV1117-33* de Texas Instruments.

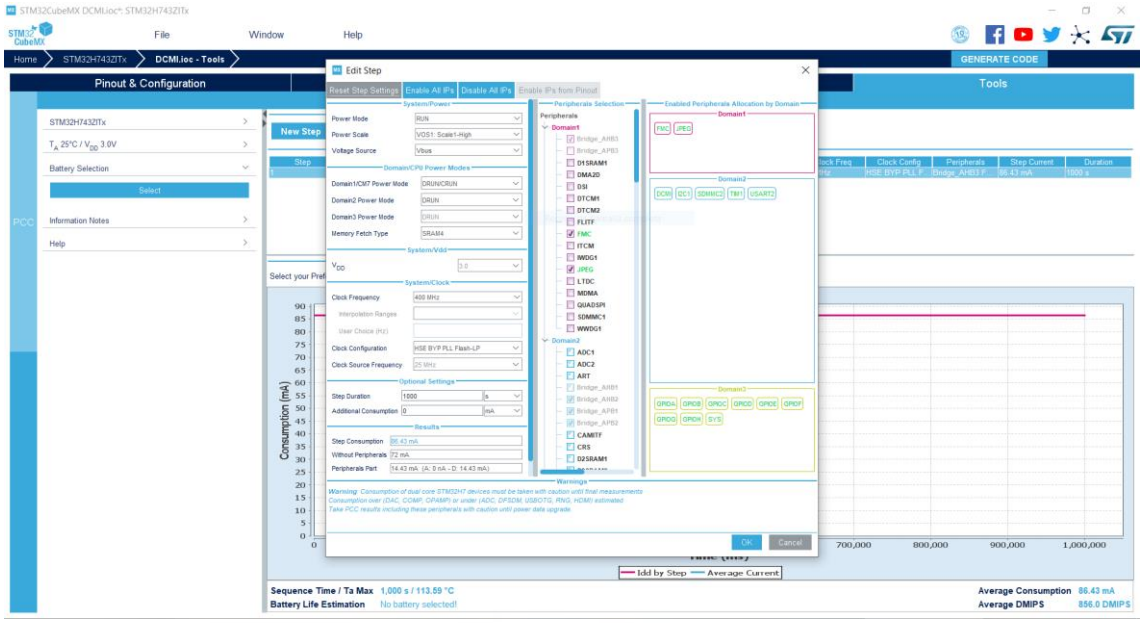


Figura 5: Calculo del consumo con *STM32CubeMX*

La parte de los 2.8V del sensor solo consume 50mA [42]. Aun así, se ha escogido la versión ajustable del TLV1117. El cálculo de las resistencias es $Vout = 1.25 * \left(1 + \frac{R_2}{R_1}\right) = 2.8V \rightarrow \frac{R_2}{R_1} = \frac{2.8}{1.25} - 1 = 1.24$. Si se considera que R_1 es de 10 kΩ, R_2tiene que ser de 12.4kΩ.

El mismo modelo de LDO ha sido elegido para los 1.8V, en este caso el TLV1117-1.8, requiriendo una corriente de solo 35mA como se ha dicho anteriormente.

Por último, la alimentación de la parte analógica del sensor necesita el menor ruido posible, ya que de lo contrario el ruido podría influir en la imagen. El ruido en el voltaje de entrada tiene que ser menor que 1/ $2^{Nº\ BITS\ ADC}$, en este caso, el número de bits del ADC son 12, así que el ruido tiene que ser menor que $\frac{1}{2^{12}} \approx 0.025\%$.

Para la referencia de tensión se ha elegido el *LTC6655BHMS8-3.0* de la serie *LTC6655* de *Analog Devices*. Esta referencia es de 3V (recordar que el sensor el voltaje analógico tiene que ser de entre 2.6 y 3.1V) con un error máximo del 0.025% y una corriente máxima de +-5mA [46].

Para proporcionar los 80mA requeridos el sensor [42], se ha utilizado un regulador lineal de ultra bajo ruido *LT3042*, también de *Analog*. Este regulador proporciona hasta 200mA, con un ruido de solo 0.8 μV_{RMS} [47], muy por debajo de los 0.73mV ($\frac{3V}{2^{12}}$) que hay de diferencia entre dos valores consecutivos en el ADC.

En la página 25 del *datasheet* del *LTC6655* se propone un circuito para aplicaciones de ultra bajo ruido en el que conecta la referencia de tensión con el regulador lineal *LT3042*.

Otras consideraciones que se han tenido en cuenta para reducir el ruido ha sido usar condensadores de tántalo y unir la masa de esta parte analógica con la digital usando una ferrita.

3.5. OTROS COMPONENTES

Para el socket de la tarjeta microSD se ha utilizado un zócalo bastante común y fácil de encontrar, es la referencia *693072010801* de *Wurth Elektronik*.

Se ha añadido un conector USB para proporcionar los 5V. Como en el caso de la tarjeta, se ha buscado un modelo que también fuera bastante común, el *47346-000*1 de *Molex*. Para futuras revisiones del proyecto se podría incluir un conversor de UART a USB como el *FT230X* de *FTDI* o incluso usar el propio USB del microcontrolador.

Para los LEDs se ha escogido unos sencillos en encapsulado 0603 mientras que los botones son los típicos de 6.00x6.00 mm de montaje superficial. Cualquiera de esas dimensiones debería valer, en concreto se ha escogido *B3SL-1022* de *Omron*.

Se han utilizado pines hembras con espaciado de 2.54mm para el conector del puerto serie, además en este puerto se han añadido un par de pines que están conectados a la alimentación de 5V y 3.3V y un par de masas. Esto permite mayor flexibilidad a la hora de alimentar la placa y poder usar fuentes con limitación de corriente durante las pruebas.

Seguramente se utilizará para programar y depurar el código del microcontrolador un programador *ST-Link V2*, con lo que se ha añadido una serie de pines hembras que encajan con el conector del programador. También se ha añadido el conector standard de 10 pines para depuración de ARM [48]. Este conector es una adaptación del conector JTAG standard de 20 pines con un paso de 0.1 pulgadas, pero en 10 pines y con un paso de 0.05 pulgadas. Permite usar cualquier depurador para ARM Cortex-M que soporte conexiones JTAG, SWD o SWO sin apenas quitar espacio de la placa, ocupa menos de 7x7mm.

Por último, el cristal que se ha elegido para el microcontrolador es el *NX2016SA* de *NDK*, este cristal oscila a una frecuencia de 25MHz. Configurando los PLL internos del microcontrolador permite proporcionar a la CPU una frecuencia de 480MHz.

4. DISEÑO DE LA PCB

La PCB se ha diseñado con el programa *KiCad*. Pese a que al ser estudiantes del grado podemos acceder a una licencia de *Eagle* con la que no se tiene ningún un límite en el área a trabajar ni en el número de capas, se ha preferido usar *KiCad* porque es un software libre que no impone ningún límite ni requiere ningún tipo de licencia o pago.

Además, *Eagle* se impartió durante el grado, usando *KiCad* en este proyecto nos prepara por si en el futuro laboral se requiriera utilizarlo.

4.1. IMPORTACIÓN DE COMPONENTES CON ULTRA LIBRARIAN

Antes de comenzar con el diseño del esquemático se verifica si los componentes están en las librerías predeterminadas de *KiCad*. Están todos menos los reguladores de tensión de la parte analógica el *LT3042* y *LTC6655*, el sensor de imagen *MT9P031* y la memoria RAM *MT48LC8M16A2*.

En la web de *Analog* está disponible el símbolo y huella del *LT3042* y del *LTC6655* [49] [50]. En el archivo comprimido que se descarga de la web incluye un documento pdf con distintos parámetros y medidas y un archivo .bxl. Este archivo es el formato de archivos del programa *Ultra Librarian*. *Ultra Librarian* es un programa gratuito que permite generar componentes para distintos software CAD (como por ejemplo *KiCad*, *Altium*, *Eagle*, *Cadence*, etc.) a partir del archivo .bxl.

Para generar el componente es tan fácil como cargar el archivo .bxl con el botón *Load Data*, seleccionar *KiCad* en el menú de selección de herramientas y clicar sobre *Export to Selected Tools*. Los componentes generados se encuentran en la subcarpeta .*Library**Exported**KiCad*\. dentro de la carpeta de instalación del programa.

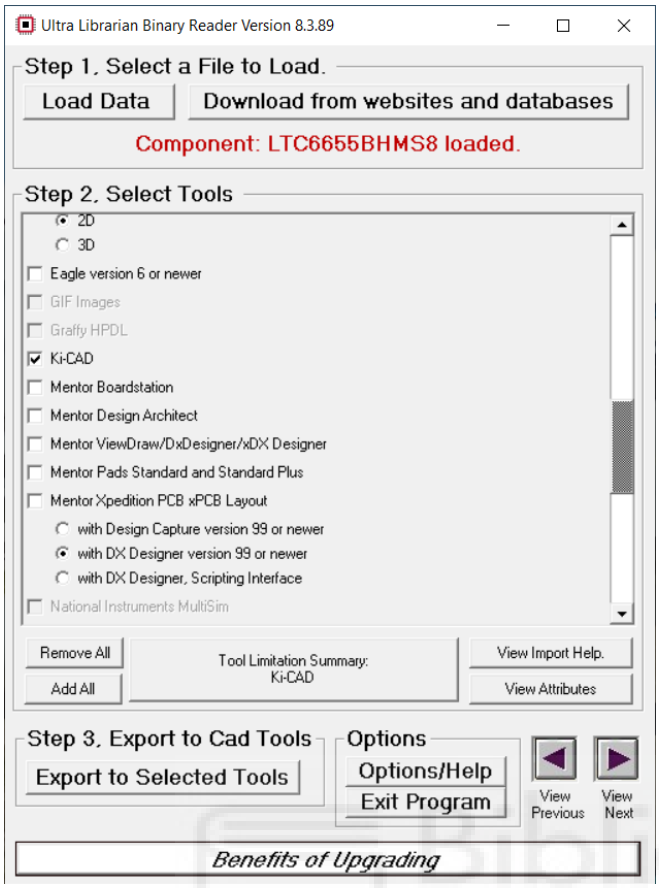


Figura 6: Ventana de *Ultra Librarian*

El símbolo del esquemático es un archivo .li. Para añadirlo a *KiCad* hay que irse al menú *Archivo* de la ventana del *Editor símbolos de esquema* y clicar *Add library…*, se selecciona el símbolo y pregunta si queremos que sea una librería global o del proyecto, en este caso esta opción no es relevante.

Para añadir la huella, en el *Editor de huellas* se pulsa en el menú *Archivo* en *Import Footprint from KiCad file*. En este caso no pregunta si es para el Proyecto o Global.

A pesar de que en las librerías de *KiCad*, está el símbolo y huella del *STM32H743*, se ha descargado la última versión de la web de ST [51] por si hubiera alguna diferencia en los pines de una revisión a otra, debido, por ejemplo, a que se hayan corregido errores.

4.2. CREACIÓN DE COMPONENTES

El primer componente que se crea es el del sensor *MT9P031*. En *KiCad* los componentes se ordenan en librerías, siendo diferentes librerías para los símbolos o las huellas, por tanto, para un componente se tiene una librería de símbolos y otra de huellas. Las librerías de símbolos son archivos *.lib* y *.dcm*, mientras que la de huellas es una carpeta cuyo nombre acaba en *.pretty* y cada huella es un archivo *.KiCad_mod* que se encuentra dentro de ella.

Una particularidad de este programa es que cada símbolo se puede asociar con diferentes huellas de tal manera que una vez acabado el esquemático tenemos que elegir la huella que tiene cada componente. Por ejemplo, en el caso de tener una resistencia en el esquemático, a la hora de crear la PCB se puede elegir si es de tamaño 0603, 0805, inserción, etc., de esta forma se puede modificar la PCB sin necesidad de rehacer el esquemático.

Para crear el componente, se crea primero una librería llamada *MT9P031* y luego un nuevo símbolo, con el mismo nombre. La referencia que se ha escogido para el componente es “U”, igual que “R” es la típica para resistencias o “C” para condensadores, “U” es la normal para los circuitos integrados. El “número de unidades por empaquetado” se ha dejado a 1, esto está pensado para hacer componentes que pueden tener varios símbolos, por ejemplo, un interruptor doble, aunque en el componente estén los dos interruptores físicamente juntos, en el esquemático puede resultar interesante tener un símbolo para cada uno de ellos y poder situarlos en distintas partes simplificando el diseño. El resto de las opciones también se dejan por defecto.

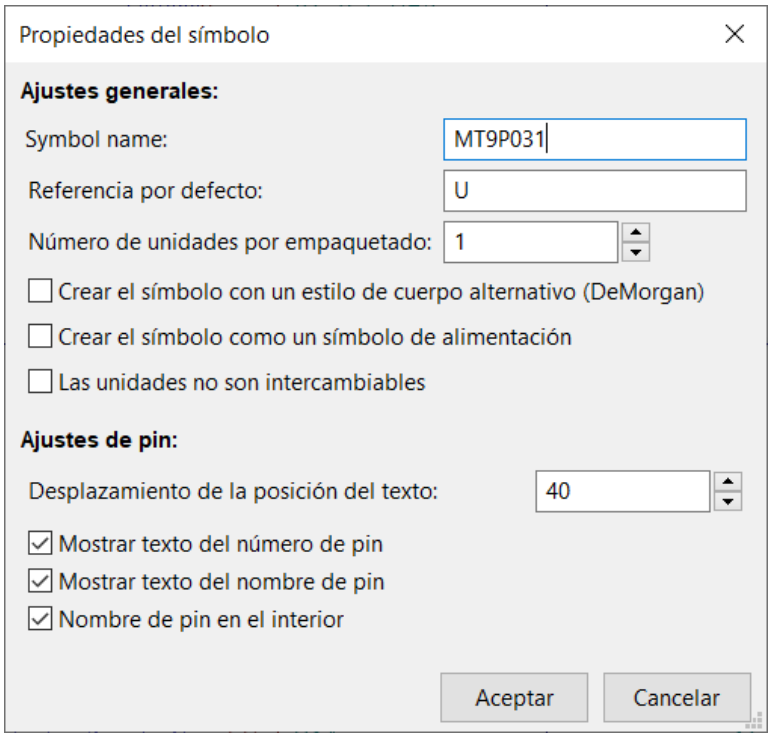


Figura 7: Propiedades del símbolo en *KiCad*

Los pines se han dividido en varias secciones: los relacionados con el I2C; los que controlan el reset, standby y habilitación de salida; el puerto paralelo de salida de los píxeles; la señal de reloj de los píxeles y las señales línea y frame válido que indican cuando leer el puerto de los píxeles; los no conectados; la alimentación digital; alimentación analógica; y las masas digitales y analógicas.

Para dibujar el rectángulo que representa al componente se empieza añadiendo un par de líneas verticales. Sobre ellas se han situado los pines de cada grupo, dejando cierta distancia entre los grupos para mejorar la legibilidad. A la hora de añadir un pin pregunta por el nombre, número, tipo (entrada, salida, alimentación, etc.) y otros parámetros como la posición, orientación, etc., que se dejan por defecto.

Una vez añadidos todos los pines, se mueven las dos líneas con los pines para que queden centradas y a una distancia que el componente resulte estético. Por últimos, se añaden la línea de arriba y abajo. En la figura 8, se observa el símbolo creado.

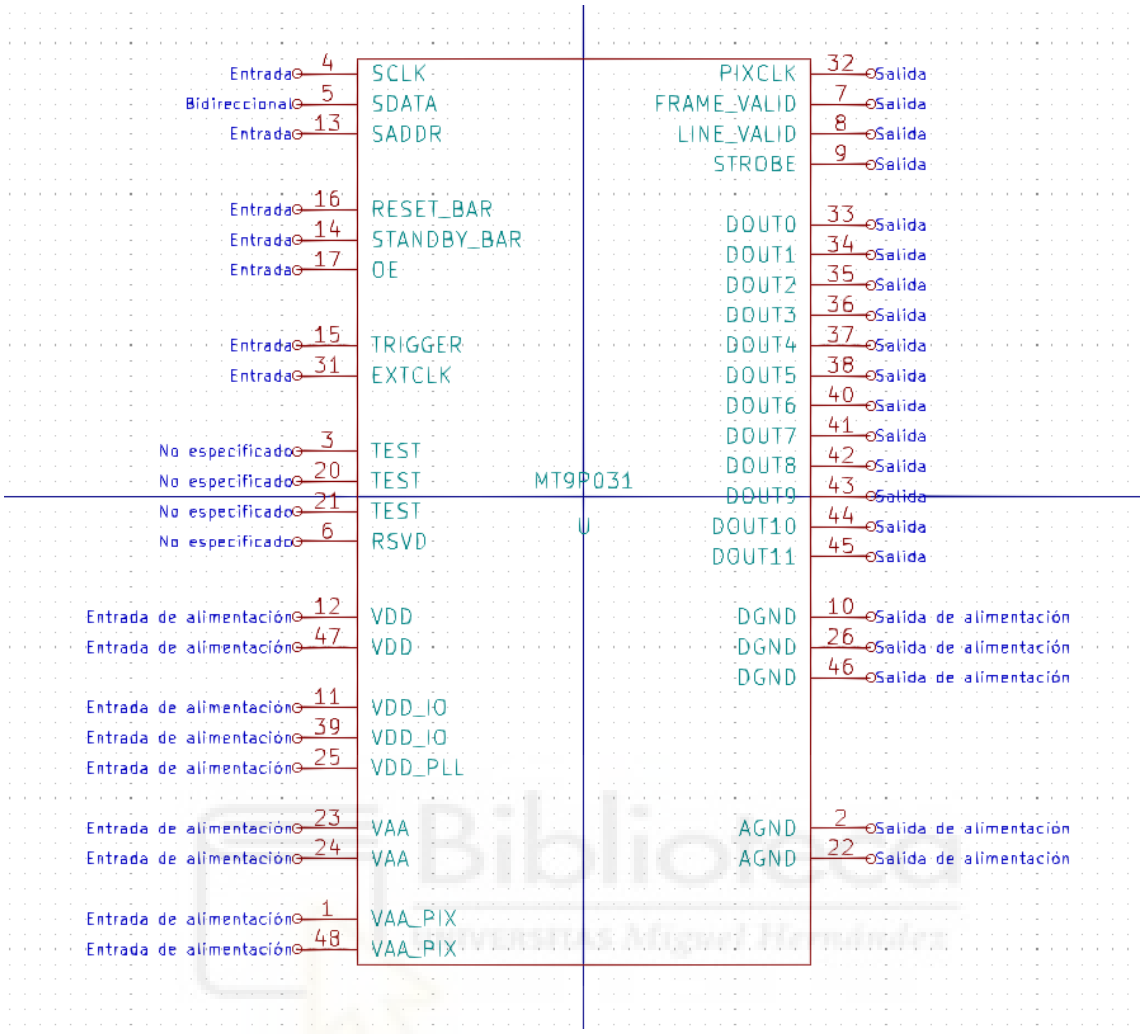


Figura 8: Símbolo del sensor MT9P031

El proceso de crear la huella en general resulta aún más sencillo. La mayoría de los circuitos integrados comparten los mismos encapsulados, a no ser que sean circuito muy particular, como es el caso de la cámara.

Primero se crea una librería de huellas, también llamada MT9P031, en la misma carpeta donde hemos creado la del símbolo. Se crea una nueva huella con el mismo nombre y aparecen dos líneas de texto: la referencia del componente (se ha indicado al crear el símbolo que aparecerá una U) y el nombre de la huella.

Se empieza dibujando el rectángulo de 10x10mm, que indica los límites del componente, en la capa delantera de fabricación. Se añaden los pines con las

medidas, para cambiar la forma se selecciona en sus propiedades que sea de forma rectangular y se introducen las dimensiones indicadas en la página 31 del *datasheet*, es importante también seleccionar en las capas técnicas las opciones de máscara de soldadura y pasta de soldar. Por último, se añade en la capa de serigrafia, las cuatro esquinas del componente para facilitar su colocación.

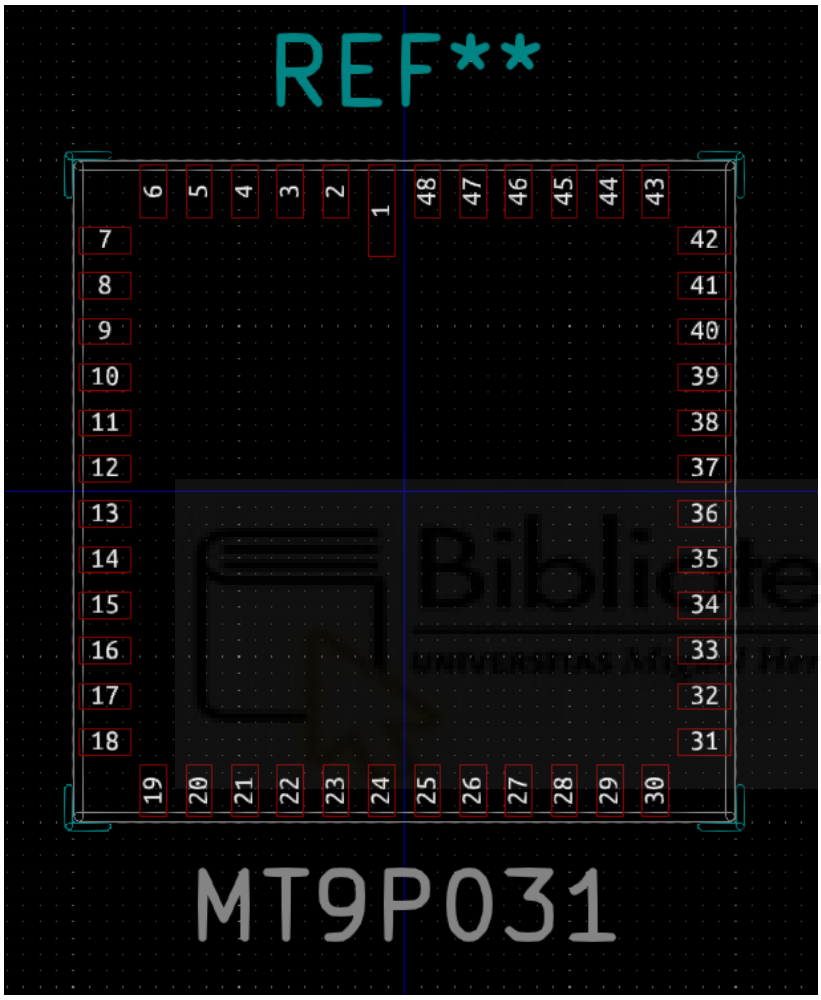


Figura 9: Huella del sensor MT9P031

Para crear el símbolo de la memoria SDRAM, se siguen los mismos pasos. Se crea una nueva librería y componente llamado MT48LC8M16A2.

Los pines en este caso se han dividido en: las entradas de dirección; selección de banco; máscaras de datos; puerto de acceso a los datos; las señales de control, como el reloj, selección de chip, etc.; las alimentaciones del *die* para evitar ruido; y las alimentaciones.

El resto del proceso es similar a cuando se ha creado el símbolo del sensor.

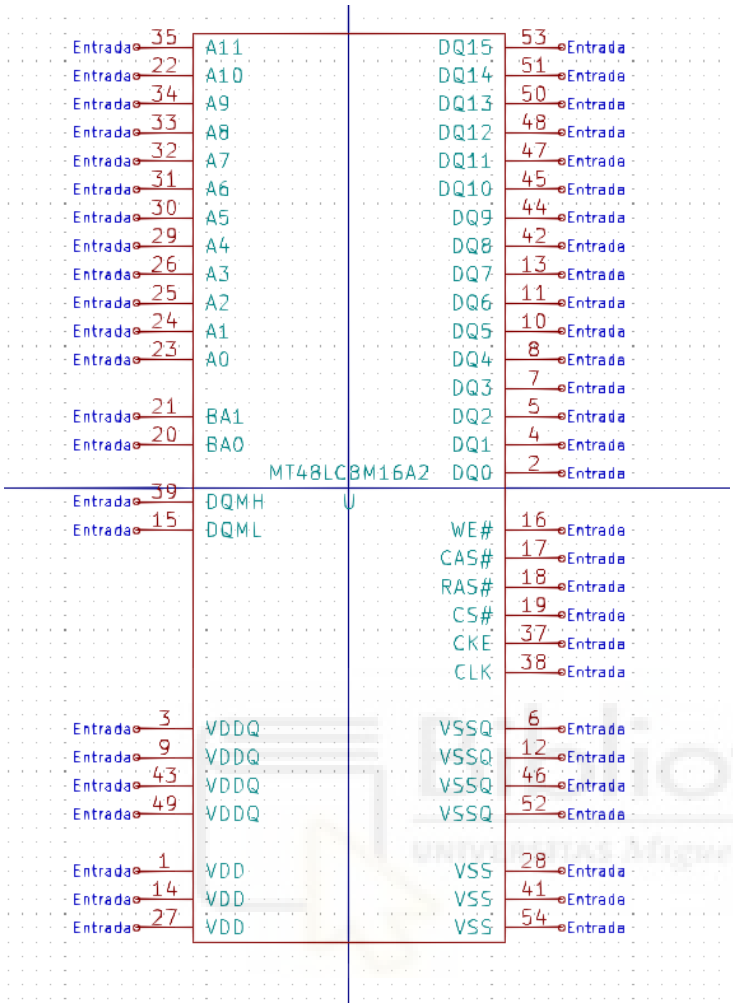


Figura 10: Símbolo de la memoria SDRAM MT9P031

En este caso la huella no es necesaria diseñarla, ya que utiliza un encapsulado TSOP de 54 pines. En la librería por defecto de *KiCad* está disponible con el nombre de *TSOP-II-54_22.2x10.16mm_P0.8mm*.

4.3. CREACIÓN DEL ESQUEMÁTICO

Una vez se tienen todos los componentes se procede a crear el esquemático. El esquemático no es más que juntar todos los componentes que se han explicado a lo largo del capítulo anterior.

El esquemático se ha dividido en 4 hojas: la primera con las conexiones del microcontrolador, otra para el sensor, una para la memoria SDRAM y la última incluye conectores y toda la parte de los reguladores de tensión.

Las distintas partes se han unido mediante etiquetas globales. Como se aprendió durante el grado, se han añadido condensadores en la alimentación lo más cercano posible a cada IC para filtrar ruidos y que el voltaje sea más estable, además también se han añadido cerca de los conectores de alimentación.

Las conexiones del microcontrolador con los distintos circuitos integrados y conectores se obtuvieron con el programa *STM32CubeMX* y se revisaron que fueran correctas comparándolas en el *datasheet* [52].

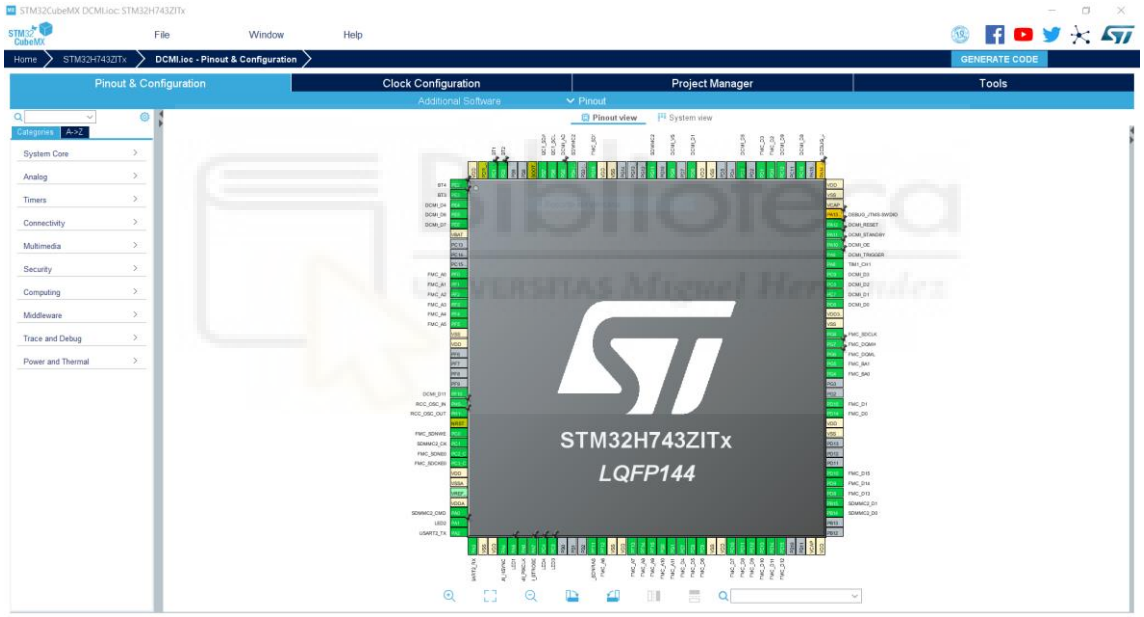


Figura 11: Conexiones del microcontrolador en *STM32CubeMX*

En las pistas de alimentación de cada IC se ha añadido resistencias de 0 ohmios para que actúen como *jumpers*. Por defecto se encuentran abiertos, esto permite cerrar la alimentación de cada IC de manera diferente durante las pruebas y en caso de fallo por sobretensión o cortocircuito evitar que se dañen todos los componentes. Además, para proteger cada LDO en caso de cambio de polaridad, se ha añadido un diodo de protección, el clásico *1N4148*.

Las resistencias de pull-up del bus I2C entre el sensor y el microcontrolador, se han conectado a la alimentación de 2.8V y 3.3V con dos resistencias de 0 ohmios distintas para poder seleccionar entre cada una de ellas. Las salidas de los integrados que se conectan al bus I2C trabajan siempre con configuración de colector abierto, esto significa que puede estar en dos estados: alta impedancia o conectado a masa. Cuando ambos lados del bus están en alta impedancia, a través de la resistencia de *pull-up*, hay un voltaje alto en el bus, en cuanto uno cierra, en el bus se indica 0 voltios. Esto permite que los datos sean bidireccionales en el bus, al contrario que pasa, por ejemplo, en el bus SPI. El sensor detecta un nivel alto en el bus cuando hay una tensión de entre 2 y 3.3V [53], mientras que el microcontrolador detecta un nivel alto a partir de 1.8V [54]. Para ambos integrados debería funcionar tanto si se alimenta a 2.8V o a 3.3V.

En el Anexo A se incluyen las hojas del esquemático.

4.4. DISEÑO DE LA PCB

Una vez listo el esquemático antes de comenzar se tiene que asignar una huella a cada componente. Las resistencias, ferritas, leds y condensadores se han elegido de tamaño 0603 (1608 en métrico), excepto los condensadores de tántalo de 100nF, que no están disponible en este tamaño y se ha elegido el inmediatamente superior, 0805 (2012 en métrico).

Los diodos *1N4148* para la protección de cambio de polaridad de los LDO se han elegido en encapsulado SOD123. Como se indicó en el capítulo anterior, los conectores de programación y comunicación serie son los típicos pines hembra con paso de 2.54mm.

Se han seguido las guías de diseño *High-Speed Interface Layout Guidelines* de *Texas Instrumens* y el libro *High Speed Digital Design* de *Howard W. Johson* y *Martin Graham*. Se ha prestado especial cuidado a los siguiente puntos:

- Para una placa de 4 capas, la disposición de las capas es: 1ª capa para componentes y señales, 2ª capa para el plano de masa, 3ª capa para alimentaciones y 4ª capa también para señales.
- Evitar llevar en paralelo dos pistas que se encuentra una sobre otra en capas consecutivas para evitar inducciones de corriente.
- Cuando dos pistas que están en capas consecutivas se cruzan hacerlo en un ángulo de 90º.
- Separar los planos de masa analógica y digital.
- Rodear las señales e integrados analógicos y digitales con sus respectivos planos de masa.
- Superponer los planos en todas las capas, y no superponer nunca dos planos distintos para evitar capacidades parasitas entre los planos.

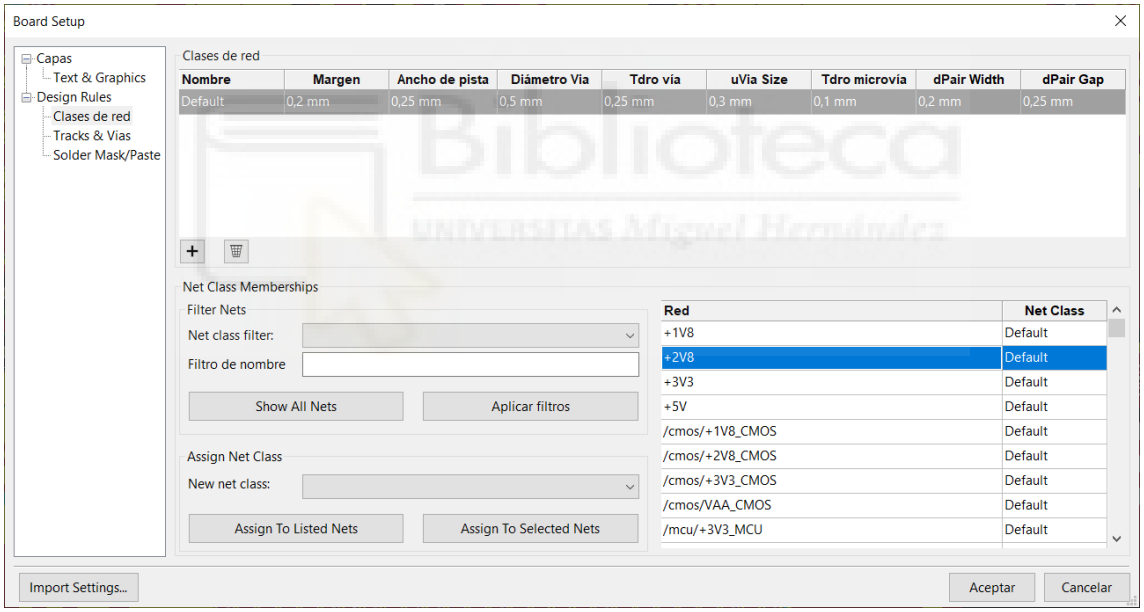


Figura 12: Tamaño de pistas y vías en *KiCad*

Se ha utilizado como referencia para los tamaños y distancias mínimas para las vías y pistas los indicados en la web del fabricante de PCB *JLCPCB* [55]. El tamaño y distancia mínima entre pistas debe ser mayor de 0.127mm y para el diámetro y taladro de la vía son de 0.2 y 0.45mm respectivamente, las microvías no se han usado en este proyecto. Por si se utilizará otro fabricante de PCB, se

ha escogido un tamaño mayor. Para las pistas, el tamaño y distancia de margen ha sido de 0.2mm mientras que los diámetros de agujero y totales de las vías han sido de 0.25mm y 0.50mm.

Cuando se comienza la placa, lo primero es dibujar las dimensiones que tiene la placa en la capa *edge.cuts*, se ha decidido que sea aproximadamente de 100x60mm, pero se puede variar en caso de necesitar más espacio.

Viendo hacía donde van las líneas desde el microcontrolador al sensor y a la memoria SDRAM se decide colocarlas y empezar a cablearlas. En caso de usar un sensor o memoria que utilizarán una frecuencia más alta habría que llevar cuidado e igualar las longitudes de las pistas para que tuvieran el mismo retraso y no se produjeran desfases, pero a estas frecuencias no resulta crítico.

Después se sitúan el zócalo de la memoria SD y el conector USB y se decide como va a ser las disposición de los distintos componentes en la placa. En la parte de abajo del sensor, lo más cerca de él se decide colocar los LDOs que le alimentan (los de 1.8V, 2.8V y el que va a la alimentación analógica). El botón de reset y los conectores de programación van arriba del MCU. Al lado del MCU, del sensor y de la memoria RAM se ponen sus respectivos condensadores. Los botones del usuario van el lado contrario al sensor pues es donde más espacio hay disponible, los diodos LEDs entre la tarjeta SD y el MCU, el LDO de 3.3V se coloca lo más cerca del USB y del conector serie, pues se va a alimentar a través de ellos.

En la Figura 13 se muestra de forma gráfica la disposición aquí explicada con las señales del sensor y de la memoria SDRAM trazadas, mientras que en la Figura 14 se muestran los componentes ya colocados.

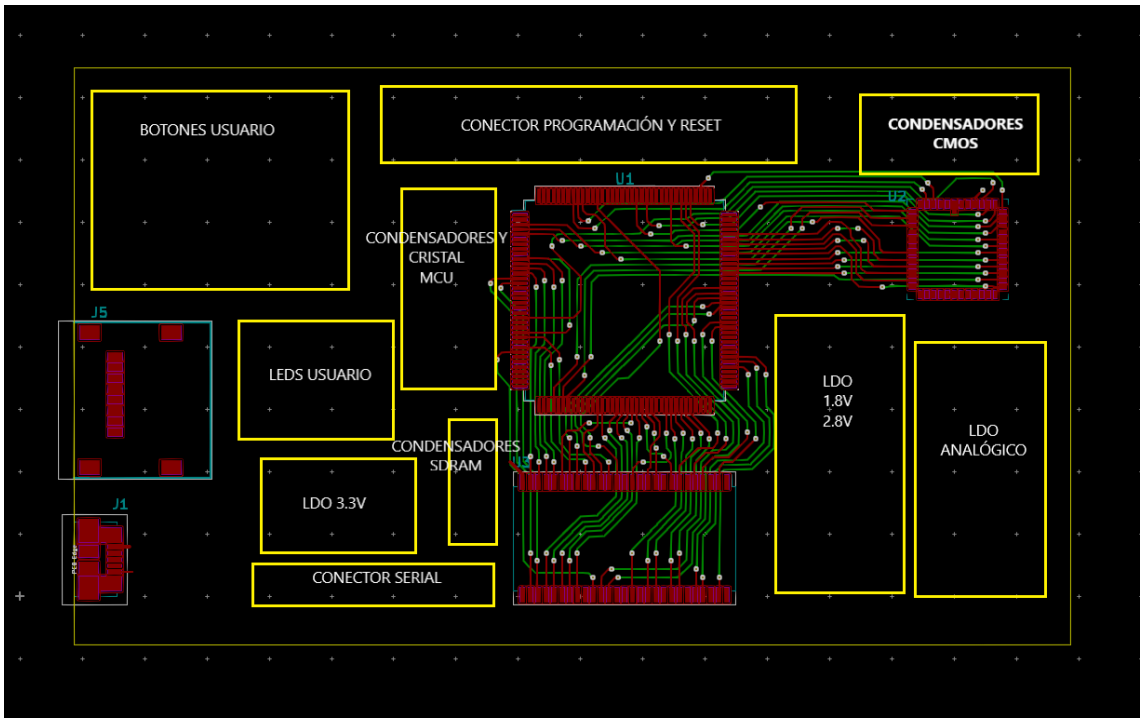


Figura 13: Disposición de las distintas partes de la PCB

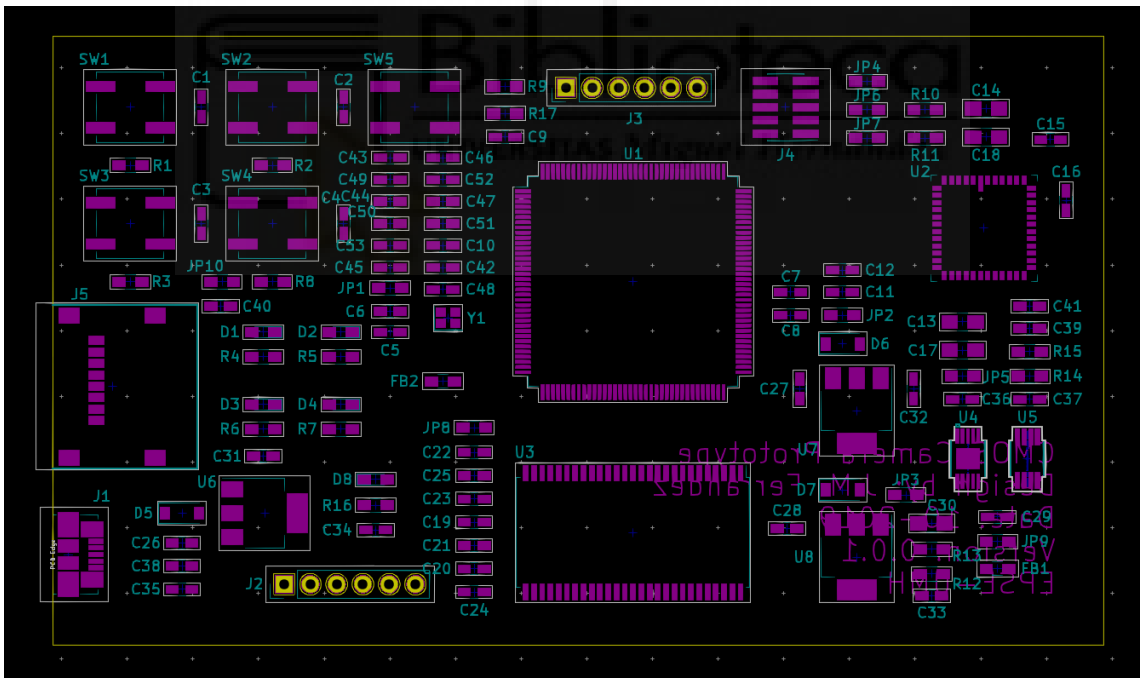


Figura 14: Componentes dispuestos en la PCB

Por último, se conectan todas las partes y se comprueba en las reglas de diseño que no haya ningún fallo, antes de mandarlo a producción.

En la figura se muestra el diseño final de la PCB, en el anexo B se incluye las litografías, más en detalle, de todas las capas de la PCB.

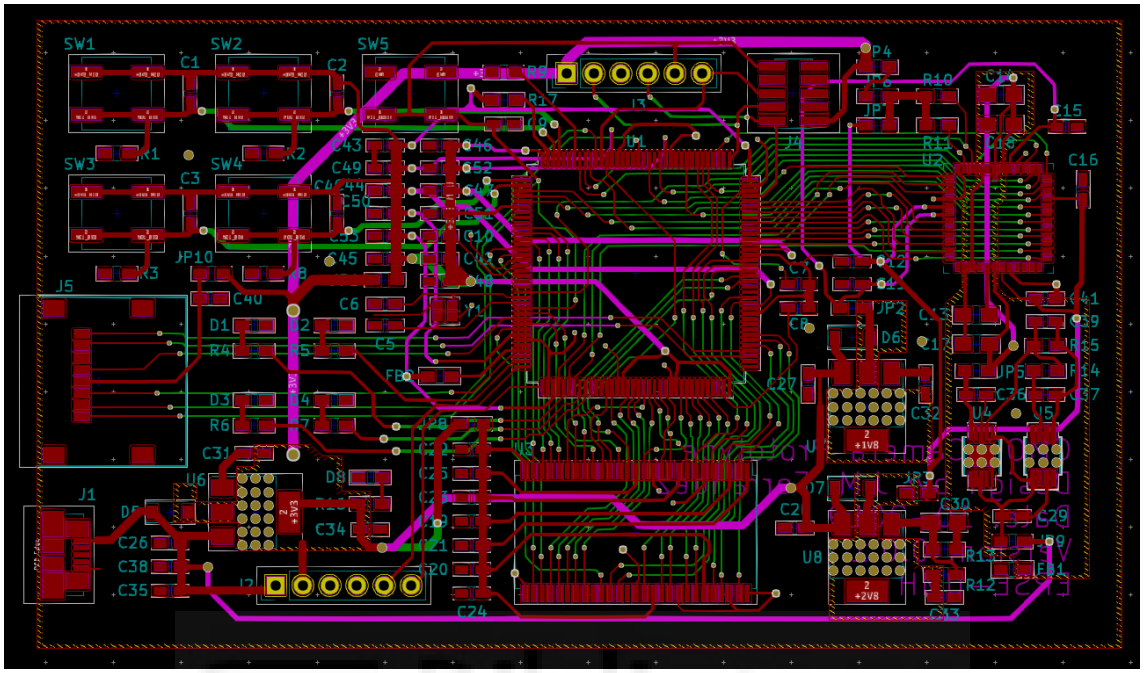


Figura 15: PCB finalizada

5. COMPROBACIÓN DEL MONTAJE

Después de fabricar y montar la placa es necesario comprobar que no ha habido fallos durante la selección de componentes, el diseño de la placa o el montaje.

Figura 16: PCB montada

5.1. TENSIONES

Se alimenta la placa a través de J2 con una fuente de 5V y con la corriente limitada a unos 100mA, así en caso de cortocircuito por un mal diseño o montaje se evita que algún componente pueda sufrir daños por sobrecorriente.

La primera observación es que el LED D8 que está en paralelo con la alimentación no se ha encendido. Se miden el voltaje en los distintos LDOs, 3.3V en U6, 1.8V en U7 y 2.5V en U8.

El voltaje en U8 debería ser de 2.8V pero se cometió un error en el diseño debido a que el voltaje headroom mínimo es de 1V (en este caso parece que 0.8V es suficiente). Seguramente no haya problemas debido a que este voltaje se utiliza

para las IO del sensor, las únicas entradas que tiene son las del bus I2C que utiliza resistencias pull-up a 2.5V o 3.3V (seleccionable) y en el lado del MCU, la tensión de entrada de alto nivel es de 0.7*Vdd [56], en este caso Vdd es 3.3V, así que a partir de 2.3V no hay problema.

Respecto al LED se verifica que esté bien montado y, efectivamente, se soldó al revés, se vuelve a montar y enciende sin problemas. Se comprueban todos los LEDs y lo mismo le ha ocurrido a D2, D3 y D4.

Se cortocircuita el jumper JP9 y se verifica que la tensión analógica a la salida de U4 es de 3.0V.

5.2. MICROCONTROLADOR

Una vez que las tensiones se han verificado para se cierra el jumper JP1 para probar el microcontrolador. El depurador lo reconoce. en este caso se ha utilizado el *ST-LINK/V2* con el programa *STM32CubeProgrammer*.

La primera prueba de programación consiste en un pequeño programa que enciende y apaga los LEDs (D1, D2, D3 y D4) dependiendo del estado de los botones (SW1, SW2, SW3 y SW4):

```
while(1){
    HAL_GPIO_WritePin(D1_GPIO_Port, D1_Pin, HAL_GPIO_ReadPin(BT1_GPIO_Port, BT1_Pin));
    HAL_GPIO_WritePin(D2_GPIO_Port, D2_Pin, HAL_GPIO_ReadPin(BT2_GPIO_Port, BT2_Pin));
    HAL_GPIO_WritePin(D3_GPIO_Port, D3_Pin, HAL_GPIO_ReadPin(BT3_GPIO_Port, BT3_Pin));
    HAL_GPIO_WritePin(D4_GPIO_Port, D4_Pin, HAL_GPIO_ReadPin(BT4_GPIO_Port, BT4_Pin));
}
```

Después de esta prueba se verifica que el cristal externo de 25MHz oscila. Hay que configurar los distintos PLL y multiplexores para que los relojes tanto del procesador como de los distintos buses de los periféricos utilicen el oscilador externo de alta velocidad (*HSE*). Esta configuración se realiza a través de los

registros del bloque RCC (*Reset and Clock Control*), para facilitar la tarea se recurre al programa *STM32CubeMX* ya que permite configurarlos gráficamente.

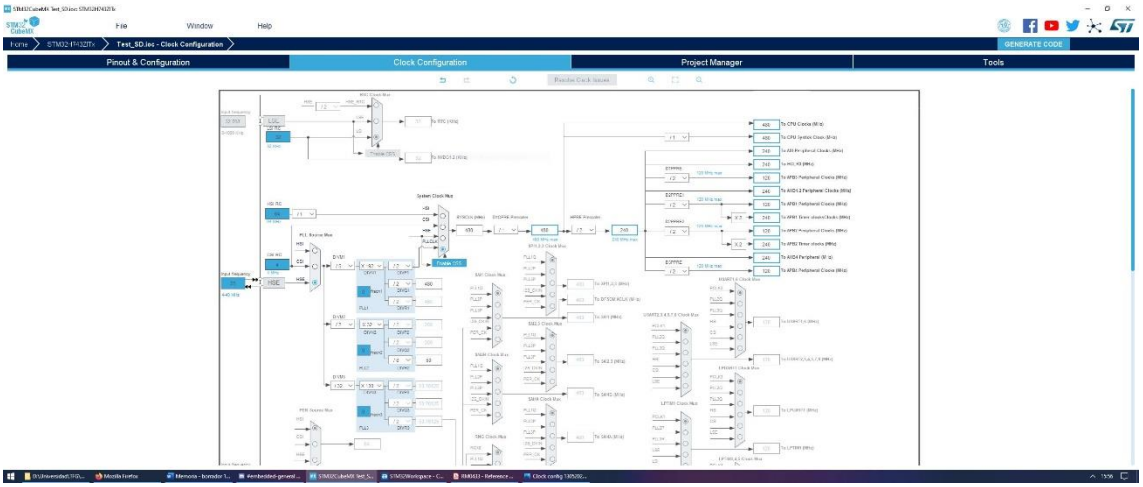


Figura 17: Configuración RCC con STM32CubeMX.

Como se muestra en la figura 1, con la configuración de los PLL seleccionada, la velocidad del procesador es de 480MHz y de los periféricos de 240MHz.

Para saber que está oscilando hay que observar el bit 7 (HSERDY) del registro CR del bloque RCC [57].

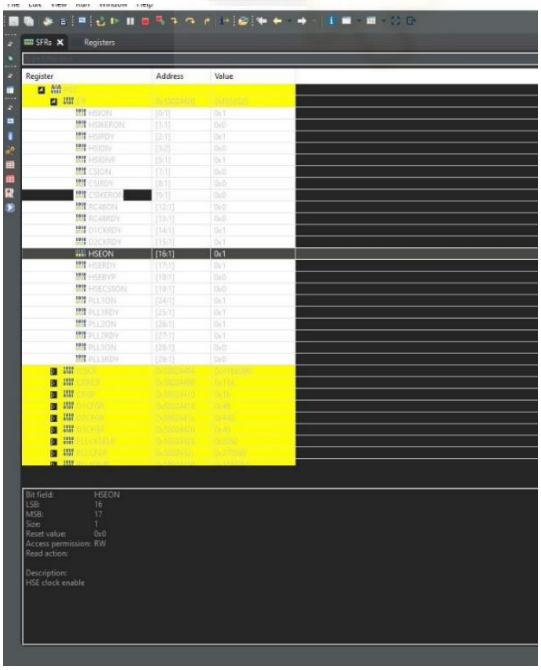


Figura 18: Verificación de que el oscilador externo está en funcionamiento.

5.3. PUERTO SERIE

Lo siguiente que se prueba es que el puerto serie funcione correctamente. La forma más sencilla para comprobar que la transmisión y recepción funcionan correctamente es programar el microcontrolador para que envíe todo lo que reciba, a esto también se lo conoce como funcionar en modo *echo*.

Esta prueba dio problemas de precisión cuando se comprobó con el oscilador interno, pero con el externo no hubo ningún problema.

```
while(1){
    c = 0;
    HAL_UART_Receive(&huart2, &c, 1, 100000);
    if(0 != c){
        HAL_UART_Transmit(&huart2, &c, 1, 100000);
    }
}
```

5.4. TARJETA MICROSD

Se comprueba que la tarjeta microSD es detectada con un código muy sencillo. Simplemente inicializa la tarjeta con el bus en modo de 4 bits y con el depurador lee las variables del microcontrolador que guardan los parámetros de la tarjeta. Para hacerlo más visual, en caso de ocurrir un error se iluminan todos los LEDs, y en cambio, si la lectura es correcta solo se enciende el LED D1.

```
hsd2.Instance = SDMMC2;
hsd2.Init.ClockEdge = SDMMC_CLOCK_EDGE_RISING;
hsd2.Init.ClockPowerSave = SDMMC_CLOCK_POWER_SAVE_DISABLE;
hsd2.Init.BusWide = SDMMC_BUS_WIDE_4B;
hsd2.Init.HardwareFlowControl = SDMMC_HARDWARE_FLOW_CONTROL_DISABLE;
hsd2.Init.ClockDiv = 0;
hsd2.Init.TranceiverPresent = SDMMC_TRANSCEIVER_NOT_PRESENT;
if (HAL_SD_Init(&hsd2) != HAL_OK){
    while(1){
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
        HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
        HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 1);
```

```
    }
} else{
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
}
```

En esta fase se encontraron muchos problemas con modelos distintos de tarjetas microSD igual que ocurre en la Raspberry Pi [58], pero consiguió funcionar con una tarjeta *Kingston Industrial* de 8 GB de capacidad. En la Figura X se muestra en el depurador la estructura *hsd2.*SdCard que contiene los parámetros de la tarjeta una vez leída por el MCU.

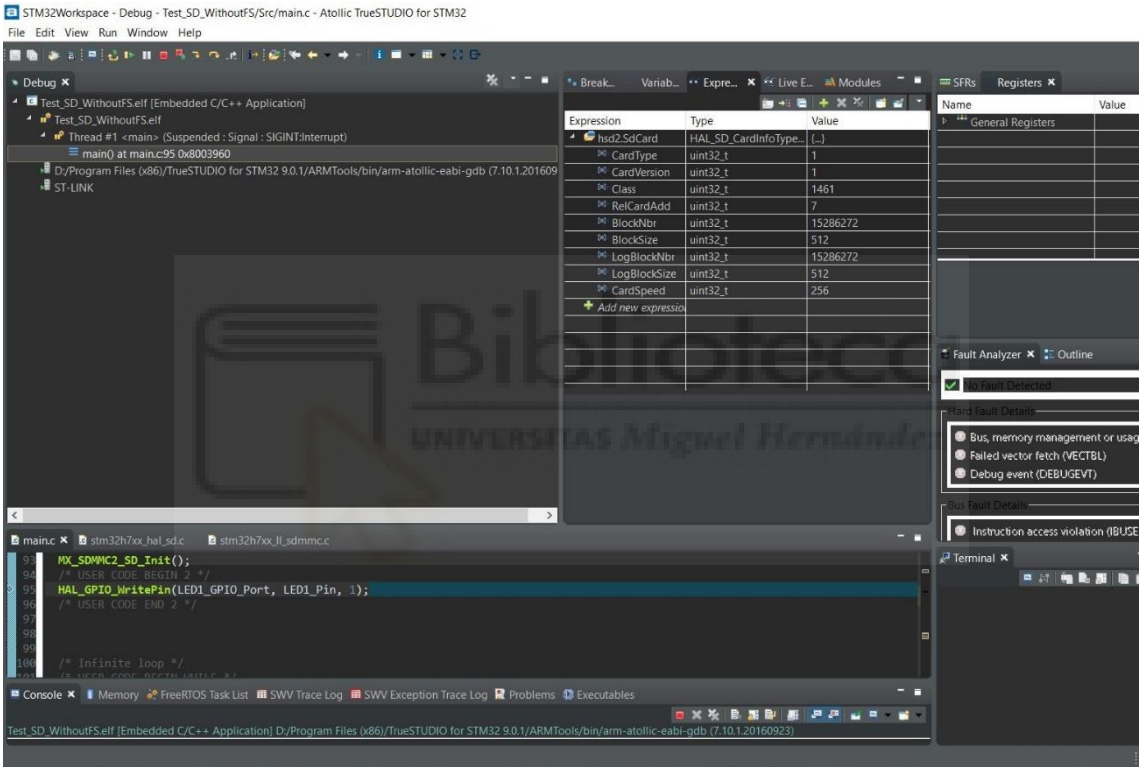


Figura 19: Datos de una tarjeta *Kingston Industrial*.

La mayoría de las veces se observa que el fallo es porque las tarjetas no responden ante el comando 41 (ACMD41). Este comando se utiliza para negociar el voltaje que se utiliza en la comunicación y consultar si la tarjeta ha terminado la secuencia de inicialización [59].

Una teoría es que los fallos pueden ser debidos a que en la fase de diseño se olvidó colocar las resistencias de 50kOhm de pullup en las señales SD_CMD,

SD_D0, SD_D1, SD_D2, SD_D3. Para intentar solventarlo se probaron las resistencias de pullup que llevan los propios pines del MCU, pero no sin resultado alguno, así que también puede ser debido a los numerosos fallos que hay en las librerías HAL con algunas tarjetas [60] [61].

La siguiente prueba que se hizo es escribir un pequeño archivo de texto utilizando el módulo FatFs. FatFs es una librería que nos proporciona acceso a cualquier tipo de memoria como un sistema de archivos Fat32 simplificando las tareas de leer y escribir archivos. Requiere de unos drivers que declaran las funciones de lectura, escritura, estado del dispositivo, etc., por suerte, ST lo ha implementado de forma nativa para el periféricos de la tarjeta SD [62].

El código de prueba crea un archivo llamado “STM32.txt” y escribe en la frase “This is STM32 working with FatFs", como antes, en caso de éxito se enciende el LED D1 y en caso de error todos los LEDs.

Una vez comprobado que escribe bien, se sube poco a poco la frecuencia del reloj del periférico para conseguir unas velocidades de lectura/escritura mayores. Al final se consigue alcanzar una velocidad de 50MHz, a partir de esta la tarjeta daba problemas.

```
int main(){
    ...
    BSP_SD_Init();
    res = f_mkfs(SDPath, FM_ANY, 0, workBuffer, sizeof(workBuffer));
    if (res != FR_OK){
        Error_Handler();
    }
    FS_FileOperations();
    while(1){}
}

static void FS_FileOperations(void){
    /* FatFs function common result code */
    FRESULT res;
    /* File write/read counts */
```

```
    uint32_t byteswritten, bytesread;
    /* File write buffer */
    uint8_t wtext[] = "This is STM32 working with FatFs";
    /* File read buffer */
    uint8_t rtext[100];
    FIL MyFile;
    /* Register the file system object to the FatFs module */
    if(f_mount(&SDFatFS, (TCHAR const*)SDPath, 0) == FR_OK){
        /* Create and Open a new text file object with write access */
        if(f_open(&MyFile, "STM32.TXT", FA_CREATE_ALWAYS | FA_WRITE) ==
FR_OK){
            /* Write data to the text file */
            res = f_write(&MyFile, wtext, sizeof(wtext), (void
*)&byteswritten);
            if((byteswritten > 0) && (res == FR_OK)){
                /* Close the open text file */
                f_close(&MyFile);
                /* Open the text file object with read access */
                if(f_open(&MyFile, "STM32.TXT", FA_READ) == FR_OK){
                    /* Read data from the text file */
                    res = f_read(&MyFile, rtext, sizeof(rtext), (void
*)&bytesread);
                    if((bytesread > 0) && (res == FR_OK)){
                        /* Close the open text file */
                        f_close(&MyFile);
                        /* Compare read data with the expected data */
                        if((bytesread == byteswritten)){
                            /* Success of the demo: no error occurrence */
                            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
                            return;
                        }
                    }
                }
            }
        }
    }
    /* Error */
    Error_Handler();
}
```

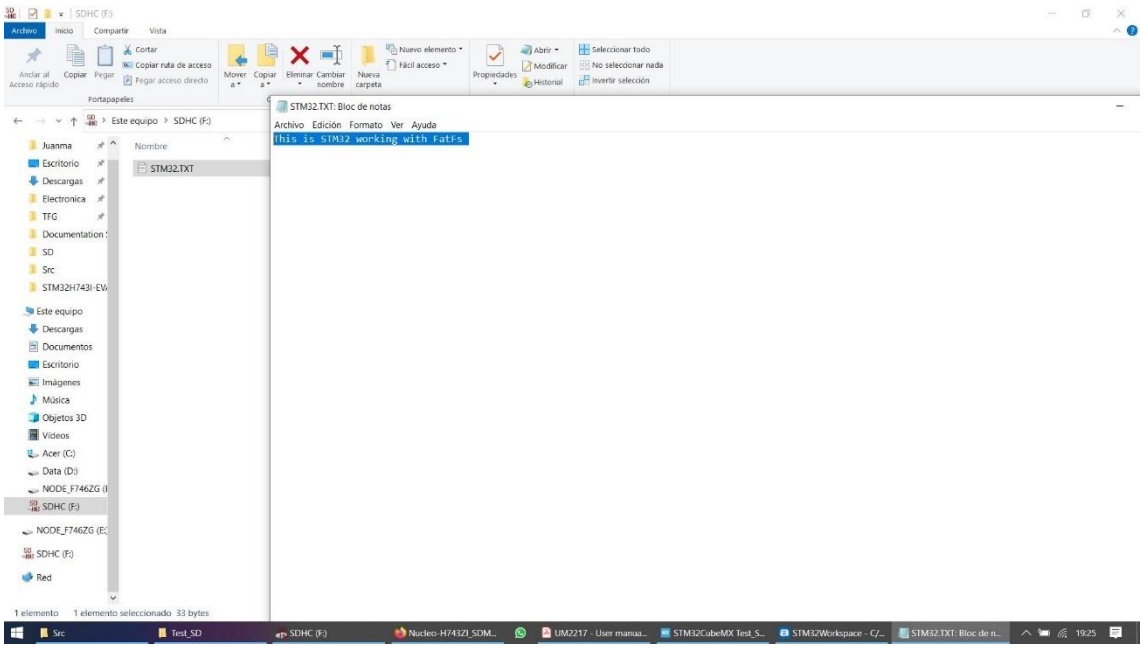


Figura 20: Escritura de un archivo de texto en la tarjeta SD.

5.5. MEMORIA SDRAM

El microcontrolador se comunica con el módulo de SDRAM a través del periférico FMC (*Flexible Memory Controller*). Este periférico permite utilizar distintos módulos de memoria (NOR, NAND, SRAM, SDRAM, etc.) con lo que su configuración no es trivial, hay que configurar los distintos tiempos, ciclos y delay dependiendo de las características del módulo. Como se escogió el mismo módulo que se utiliza en la placa *STM32F746G-DISCO*, la configuración es la misma que la utilizada en los códigos de ejemplo.

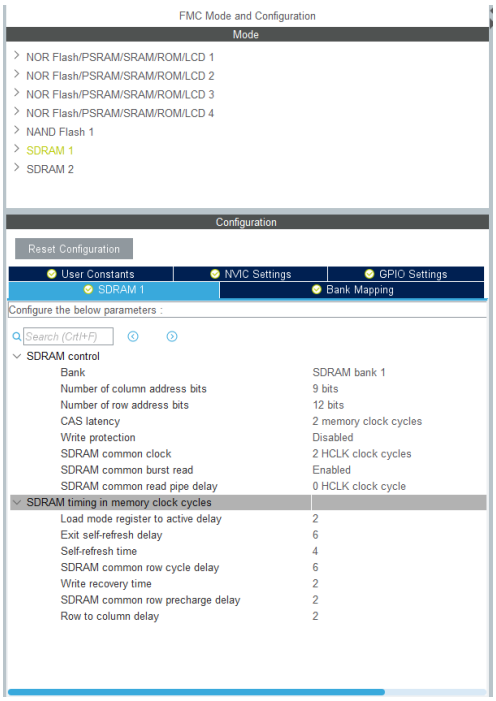


Figura 21: Configuración FMC del módulo de RAM.

En la documentación se observa que se pueden seleccionar distintas direcciones de acceso a la SDRAM, siendo la dirección por defecto `0xC0000000` [63]. El código de prueba también está basado en los ejemplos de la placa *STM32F746G-DISCO* y consiste en dos funciones. La primera, `fillBuffer` simplemente escribe en la dirección `pBuffer` (este puntero apunta a `0xC0000000`) una serie de números consecutivos de longitud `uwBufferLenght`, comenzando por `uwOffset`. La otra función, `compareBuffer`, vuelve a generar la serie y comprueba que los datos en la SDRAM son correctos.

```
#define BUFFER_SIZE 8388608
#define BUFFER_OFFSET 0x2222

int main(void){
    FMC_SDRAM_CommandTypeDef command;
    uint16_t * ptr;
    uint16_t buffer;
    ...
    /* Write data to the SDRAM memory */
    HAL_SDRAM_WriteProtection_Disable (&hsdram1);
    ptr = (uint16_t *) 0xC0000000;
```

```
    fillBuffer(ptr, BUFFER_SIZE, BUFFER_OFFSET);
    if(compareBuffer(ptr, BUFFER_SIZE, BUFFER_OFFSET)){
        Error_Handler();
    }
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
    while (1) {
    }
}

static void fillBuffer(uint16_t *pBuffer, uint32_t uwBufferLenght, uint16_t
uwOffset){
    uint32_t tmpIndex = 0;
    /* Put in global buffer different values */
    for (tmpIndex = 0; tmpIndex < uwBufferLenght; tmpIndex++){
        pBuffer[tmpIndex] = tmpIndex + uwOffset;
    }
}

static uint8_t compareBuffer(uint16_t *pBuffer, uint32_t uwBufferLenght,
uint16_t uwOffset){
    uint32_t tmpIndex = 0;
        /* Put in global buffer different values */
        for (tmpIndex = 0; tmpIndex < uwBufferLenght; tmpIndex++){
            if(pBuffer[tmpIndex] != tmpIndex + uwOffset){
                return 1;
            }
        }
        return 0;
}
```

En la prueba se concluye que el tamaño máximo de la serie es de 8388608. Teniendo en cuenta que cada número que se ha escrito es de 2 bytes, el tamaño de la SDRAM debe ser de 16777216 bytes. En la figura X se observa que la última dirección valida es la `0x0C0FFFFFF`, el valor 0xFFFFFF en decimal equivale a 16777215.

En la documentación de Micron [64], especifican que la memoria consiste en 4 bancos cada uno de 4096 filas por 512 columnas siendo el tamaño de cada

elemento de 2 bytes, 4*4096*512*2 = 16777216 bytes. Concuerda con el que tamaño averiguado experimentalmente.

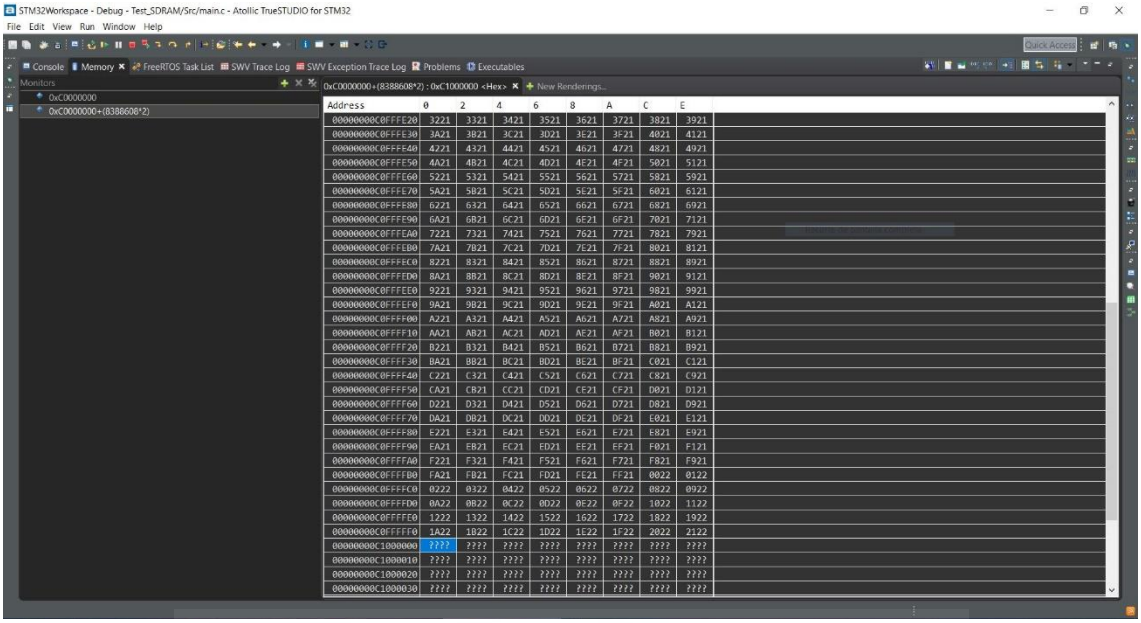


Figura 22: Memoria SDRAM llena.

5.6. BUS I2C Y RELOJ DEL SENSOR CMOS

Probar que el sensor CMOS funcionará bien es una de las partes más delicadas ya que un mal diseño o montaje de algún bloque de alimentación puede destruirlo. Se sueldan los jumpers JP2, JP3, JP4 y JP5 y con la corriente de la fuente de alimentación limitada se comprueba que está no es excesiva ni que el sensor se calienta.

Para el bus I2C las primeras pruebas se realizan con el jumper JP7 soldado, este jumper proporciona 2.5V (se había diseñado para 2.8V como se explica en la sección 6.1) mientras que el jumper JP6 proporciona 3.3V al bus.

Probar el bus I2C no resulta fácil, en las primeras pruebas se configura el bus a 400KHz tal y como indica en el datasheet [65] y se intenta que el sensor responda a alguna de sus direcciones (0xBA o 0x90) con el bit ACK. En estas pruebas las lecturas del bus eran muy erráticas. Así que se decide hacer más pruebas bajando la velocidad del bus a 50KHz, soldando con el jumper JP6 o jugando

con el nivel de la señal *SADDR* que determina que dirección tiene el sensor, pero nada mejora los resultados.

Con el osciloscopio y el analizador lógico se observan las señales y se llega a la conclusión de que el microcontrolador no se controla como se espera. Investigando la librería HAL_I2C se descubre que hay que utilizar las funciones *HAL_I2C_Mem_Read* y *HAL_I2C_Mem_Write*.

Utilizando estas funciones se consigue que el sensor responda tal y como se espera. Funciona con el bus a 2.5V y la frecuencia se prueba que funciona hasta los 900KHz, de todas formas, se ajusta a 400KHz para evitar problemas de lectura/escritura. En la Figura 11 se muestra una lectura de varias registros consecutivos en el analizador lógico, mientras que en la Figura 12 el valor leído del registro 0x00 es 0x18010 tal y como se indica en la referencia de registros [66].

Figura 23: Lectura de varios registros visto en el analizador lógico.

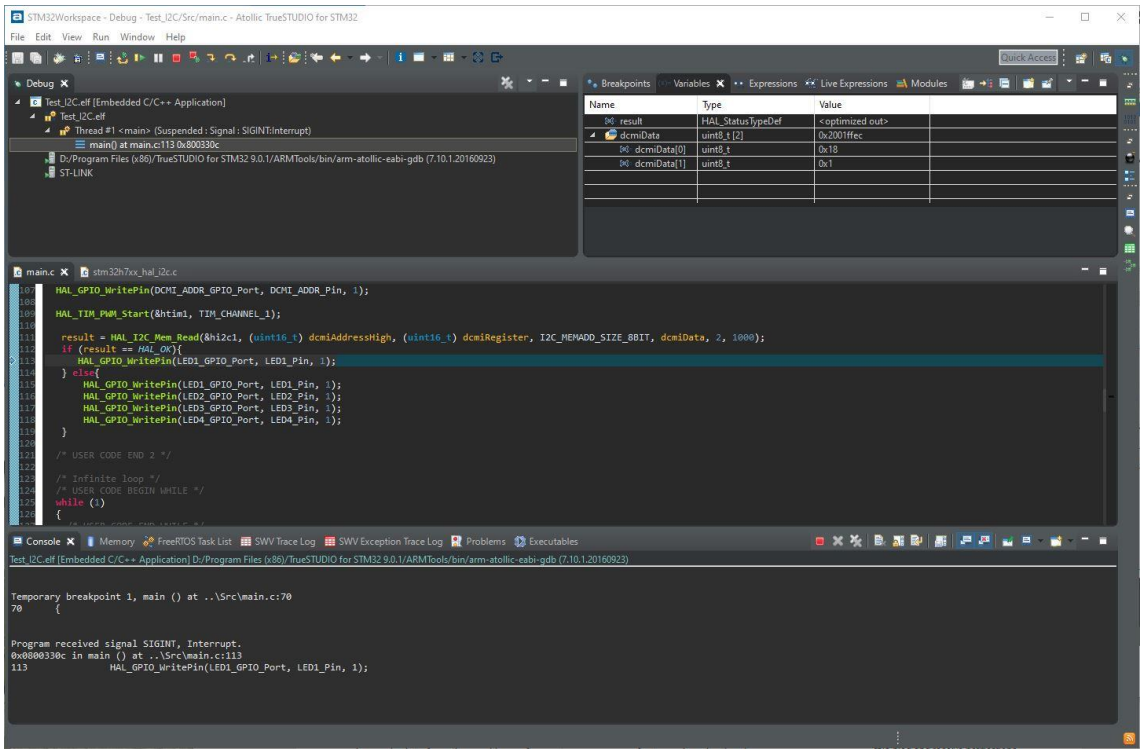


Figura 24: Lectura del registro 0x00 de configuración del sensor.

Según indica el datasheet [67] la frecuencia del reloj externo debe ser de entre 6 y 27 MHz con el PLL activado. Este reloj se genera con una señal PWM del microcontrolador. Esta señal tiene un duty cycle del 50% y, para evitar problemas de ruido y puesto que la frecuencia se va a multiplicar en el PLL, se decide que sea de 6MHz.

La frecuencia de los timers en el MCU es de 240MHz en el microcontrolador (se indica en la sección 6.2 de la hoja de datos). Se calcula que el número de periodos de 240MHz que hay en una señal de 6MHz es de 40 (diviendo 240MHz/6MHz). Este valor se utiliza en la configuración del registro de *autoreload* (*ARR*) (en el registro ARR el valor de 0 corresponde a 1 periodo, el número de periodos no puede ser menor de 1 porque indica cada cuantos ciclos de reloj se reinicia el periodo de PWM, en este caso el valor escrito en el registro ARR es de 39). El duty cycle se ajusta con el registro de captura y comparación (*CCR*) a 40*50% = 20.

Se comprueba con el osciloscopio que la señal es de unos 6MHz y el duty del 50%. En la Figura 13 se ve que hay algo de error en las medidas y *ringing*, debido a que se ha utilizado un osciloscopio y sondas de 50MHz para medir la señal.

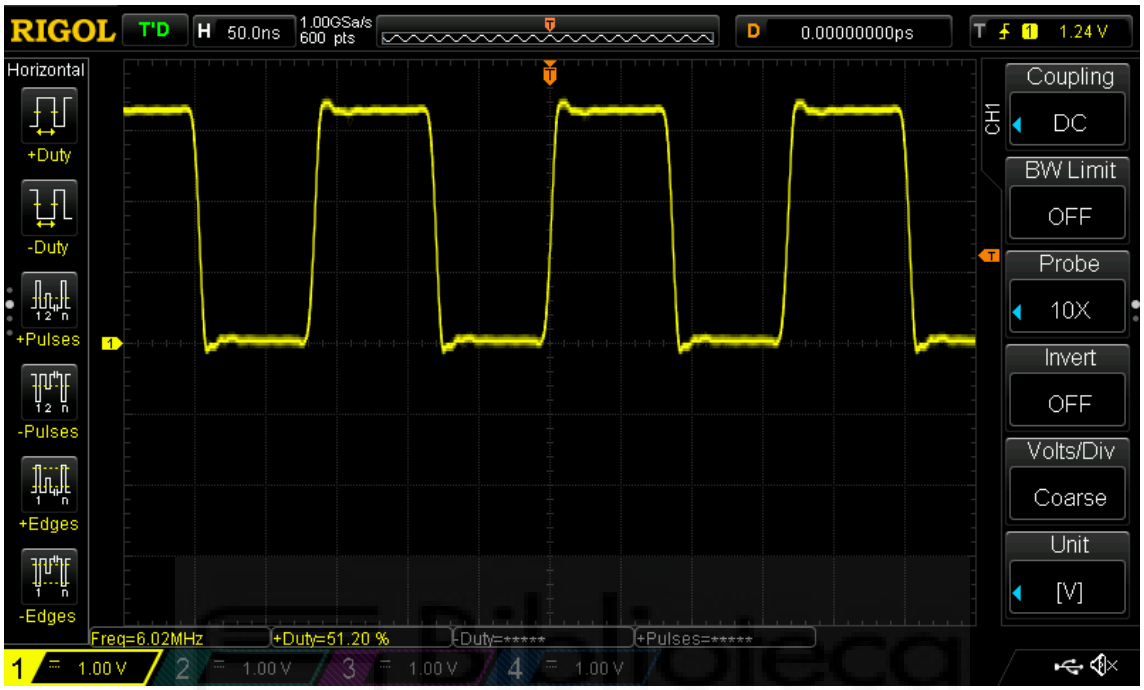


Figura 25: Medición de la señal de reloj con el osciloscopio.

5.7. COMPROBANDO TODO A LA VEZ

Por últimos, utilizando las mismas funciones que las pruebas anteriores se comprueba que todo funciona a la vez.

6. PRUEBA DEL SENSOR

6.1. CONFIGURACIÓN DE LOS PERIFÉRICOS

El periférico DCMI tiene la función de leer los datos de los píxeles provenientes del sensor CMOS. Estos datos se transmiten en una interfaz paralela de 12 bits (señal desde la *DCMI_DO* a *DCMI_D11*), la señal *DCMI_PIXCLK* sirve para sincronizar los datos y las señales *DCMI_HSYNC* y *DCMI_VSYNC* indican que son válidos cuando están se encuentran a nivel alto.

El periférico se configura a 12 bits por píxel con sincronización externa y, en contra de la intuición, hay que indicar que la polaridad de las señales *DCMI_HSYNC* y *DCMI_VSYNC* es baja, para que detecte que los datos son válidos a alto nivel [68].

```
hdcmi.Instance = DCMI;
hdcmi.Init.SynchroMode = DCMI_SYNCHRO_HARDWARE;
hdcmi.Init.PCKPolarity = DCMI_PCKPOLARITY_FALLING;
hdcmi.Init.VSPolarity = DCMI_VSPOLARITY_LOW;
hdcmi.Init.HSPolarity = DCMI_HSPOLARITY_LOW;
hdcmi.Init.CaptureRate = DCMI_CR_ALL_FRAME;
hdcmi.Init.ExtendedDataMode = DCMI_EXTEND_DATA_12B;
hdcmi.Init.JPEGMode = DCMI_JPEG_DISABLE;
hdcmi.Init.ByteSelectMode = DCMI_BSM_ALL;
hdcmi.Init.ByteSelectStart = DCMI_OEBS_ODD;
hdcmi.Init.LineSelectMode = DCMI_LSM_ALL;
hdcmi.Init.LineSelectStart = DCMI_OELS_ODD;
```

El DMA (*Direct Memory Access)* se encarga de enviar los datos desde el registro *DR* (*Data Register*) del periférico DCMI hasta la memoria SDRAM. El tamaño del registro es de 32 bits y, por tanto, el ancho del bus de datos tiene que configurarse, tanto en el origen como en el destino (el DMA de este MCU permite convertir datos de un tamaño a otro durante la transferencia), a 32 bits. Para designar el ancho de 32 bits se utiliza el término palabra o *word* en inglés.

El DCMI manda los datos a través de la memoria FIFO (*first in first out*) de 4 palabras (4 x 32bit) del DMA [69]. Se permite variar la velocidad de escritura (o cadencia de escritura) de la memoria FIFO ajustando el tamaño de ráfaga (*Burst Size*). Este tamaño es el tamaño de datos que se acceden en cada lectura desde el periférico al bus DMA o en cada escritura desde el bus a la memoria. Se puede configurar para que acceda de palabra en palabra (cada 32 bits) o cada 4 palabras, permitiendo transferencias 4 veces más rápidas que la velocidad del bus DMA.

En esta aplicación el procesador y los periféricos funcionan a máxima frecuencia, la SDRAM es de 2 bytes y una frecuencia de 166MHz (la velocidad de lectura y escritura es de 322MBps), mientras que la velocidad máxima de la cámara es 96MPPS, es decir, 2 bytes (en realidad 12 bits) a 96MHz o 192MBps. Por tanto, no hay diferencia si se utiliza un tamaño de ráfaga de 1 o 4 palabras.

El modo normal de la DMA, al contrario que el modo circular, está pensando para que cuando una transferencia termine y se quiera volver a comenzar haya que indicárselo y no se haga de forma automática sobrescribiendo los datos. Como se busca realizar solamente una foto el DMA se configura en este modo.

Por último, cada vez que los datos del registro *DR* se escriben en memoria, la dirección de memoria debe incrementarse hasta transferir todos los píxeles del sensor.

```
hdma_dcmi.Instance = DMA1_Stream0;
hdma_dcmi.Init.Request = DMA_REQUEST_DCMI;
hdma_dcmi.Init.Direction = DMA_PERIPH_TO_MEMORY;
hdma_dcmi.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_dcmi.Init.MemInc = DMA_MINC_ENABLE;
hdma_dcmi.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
hdma_dcmi.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
hdma_dcmi.Init.Mode = DMA_NORMAL;
hdma_dcmi.Init.Priority = DMA_PRIORITY_VERY_HIGH;
hdma_dcmi.Init.FIFOMode = DMA_FIFOMODE_ENABLE;
hdma_dcmi.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_FULL;
```

```
hdma_dcmi.Init.MemBurst = DMA_MBURST_INC4;
hdma_dcmi.Init.PeriphBurst = DMA_PBURST_SINGLE;
```

6.2. FUNCIONES I2C

La familia de microcontroladores STM32 utiliza un *endianess little-endian*. Esto significa, que al contrario que nosotros, en el procesador, en un número de 16 bits se guarda en la primera posición el byte menos significativo y en la segunda el más significativo, por ejemplo, el número 5789 en hexadecimal es 0x169D, pero en memoria se guarda como 0x9D16.

En cambio, el sensor *MT9P031* en los registros de 16 bits utiliza el formato *big-endian*. Para leer o escribir datos al sensor desde el MCU hay que darles la vuelta a los bytes para que no se manden como *little-endian*. Una de las funciones más sencillas para intercambiar los bytes de una variable de 16 bits es la función *__REV16* de la librería de *ARM CMSIS*

Si se interactuará directamente con el registro del periférico I2C, transmitir un mensaje de 16 bits al sensor sería tan fácil como mandarle primero el byte más significativo seguido del segundo byte del número porque en los microcontroladores STM32 los datos que se transmiten a través de la interfaz I2C se escriben o leen en un registro de 1 byte [70], pero las funciones *HAL_I2C_Mem_Write* y *HAL_I2C_Mem_Read* de los drivers *STM32 HAL*, reciben un puntero a una variable de 16 bits.

La figura X muestra en una ventana el depurador que se manda el número 2005 (0x07D5) pero en la segunda ventana en el analizador lógico los datos enviados son 0xD507.

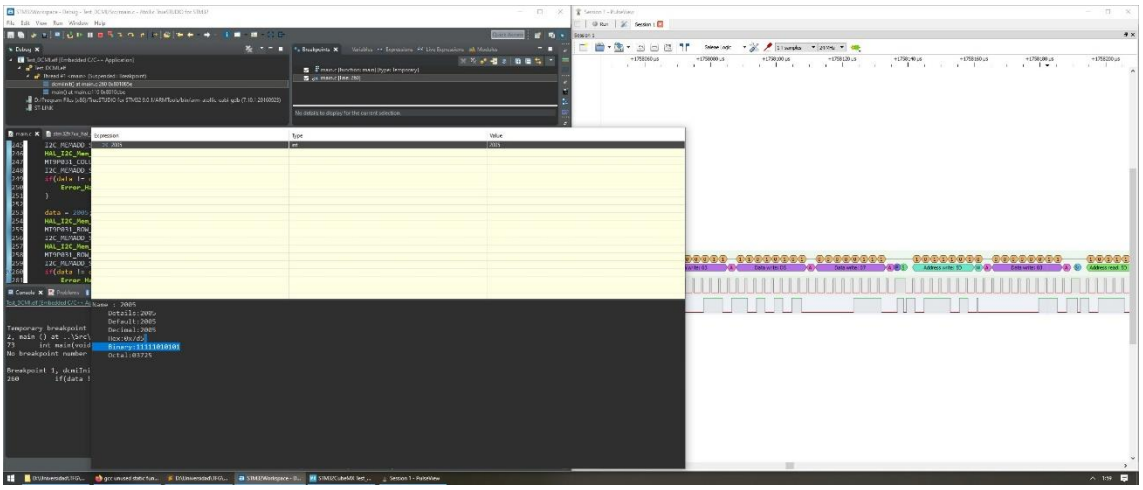


Figura 26: *Endianess* en el bus I2C.

Para automatizar el proceso de intercambio de bits y simplificar el envío de datos se han creado las funciones *MT9P031_I2C_Write* y *MT9P031_I2C_Read*:

```
static void MT9P031_I2C_Write(uint8_t address, uint16_t reg, uint16_t data){
    data = __REV16(data);
    HAL_I2C_Mem_Write(&hi2c1, address, reg, I2C_MEMADD_SIZE_8BIT,
        (uint8_t *) &data, 2, 1000);
}

static void MT9P031_I2C_Read(uint8_t address, uint16_t reg, uint16_t* data){
    HAL_I2C_Mem_Read(&hi2c1, address, reg, I2C_MEMADD_SIZE_8BIT,
        (uint8_t *) data, 2, 1000);
    *data = __REV16(*data);
}
```

Además, teniendo como referencia el controlador del sensor MT9P031 para Linux [71] se ha creado un archivo de cabecera con las definiciones de los valores hexadecimales de los distintos registros y parámetros por los cuales se puede configurar el sensor.

En concreto se han añadido algunas configuraciones que faltaban del registro *Restart* (0x00B), *PLL Config 1* (0x011), *PLL Config 2*(0x012), *Read Mode 1* (0x01E), *Read Mode 2* (0x020), *Test_Pattern_control* (0x0A0) y *Test_Pattern_Bar_Width* (0x0A4).

6.3. PROCESAMIENTO DE IMÁGENES EN PYTHON

Una forma rápida de visualizar los datos obtenidos con la cámara es en Python y sus librerias *Numpy*, *MatplotLib* y *OpenCV*.

Lo primero es abrir el archivo donde están guardados los datos. La función *fromfile* del módulo *Numpy*, devuelve un array de valores a partir de un archivo binario. El primer argumento es el objecto de tipo archivo donde se encuentran los datos, el segundo el tipo de datos y el tercero la cantidad de datos a leer. Para indicar que es un tipo de datos se utiliza la función np.dtype, en este caso con el argumento '<u2'. La '<' indica que se ha guardado en un sistema *little-endian* (si fuera *big-endian* sería con '>', la 'u' que cada dato es de tipo *unsigned int* y el '2' el número de bytes que forman cada tipo.

La cantidad de datos a leer es el tamaño en píxeles de la imagen, por tanto, lo calculamos multiplicando el ancho por el alto de esta. El array que devuelve *fromfile* es de dimensiones 1 por el tamaño en píxeles. Se redimensiona con la función *reshape((height, width))* a un array de dimensiones altura por ancho, en píxeles de la imagen.

Como se explicó en la sección 2.1 para convertir los valores del patrón en una imagen con color hay que utilizar un algoritmo de *demosaicing*. *OpenCV* (*open computer vision*) es una librería pensada para el procesamiento de imágenes en tiempo real e incluye más de 2500 algoritmos [72], entre ellos de *demosaicing*. La función *cvtColor* convierte una imagen de un espacio de color a otro, en este caso de Bayer a RGB con el argumento *cv2.COLOR_BayerGR2RGB*. A parte del tiempo de desarrollo que se ahorra en un algoritmo de *demosaicing*, evita también dudar de él en caso de que la imagen que se obtenga no sea la correcta.

A parte, otra función interesante de *OpenCV* es *imwrite*. Permite guardar los datos en un archivo .jpg.

Por último, *matplotlib* es una libraría para crear visualizaciones en Python, sus funciones están basadas en Matlab. Así que igual que en Matlab con la función *imwshow* se muestran los valores como si fuera una imagen.

El script completo para un archivo llamado “test.hex” de dimensiones 2750x2004 píxeles es el siguiente.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

path = 'test.hex'

height = 2004
width = 2750
imsize = height * width

with open(path, "rb") as rawimage:
    bin_image = np.fromfile(rawimage, np.dtype('<u2'),
        imsize).reshape((height, width))
    bin_image = cv2.cvtColor(bin_image, cv2.COLOR_BayerGR2RGB)
    cv2.imwrite('test.jpg', bin_image)
    plt.imshow(bin_image, vmin=0, vmax=4096)
    plt.show(block=True)
```

6.4. GUARDADO DE LAS IMÁGENES EN LA TARJETA USD

Uno de los problemas a la hora de guardar las fotos de prueba es que el depurador *ST-Link v2* es realmente lento. El depurador accede a los datos de la SDRAM a través del protocolo *Serial Wire Debugging* (SWD) que no está pensado para acceder a grandes cantidades de datos, por ejemplo, para copiar 2000 bytes, una imagen de 1Megapíxel, tardaba más de hora y media.

Por ello, se decide guardar los datos en la tarjeta microSD. Solamente hay que hacer una función que lea cada píxel, cada 2 bytes, de la SDRAM y los escriba en un archivo en formato binario. La función devuelve un 0 si el número de bytes

escritos en la tarjeta es el mismo que de píxeles, para comprobar si ha habido algún error.

```
static void saveFile(uint16_t * ptr, uint32_t length){
    FIL file;
    uint32_t byteswriten = 0;
    f_open(&file, "test.hex", FA_CREATE_ALWAYS | FA_WRITE);
    f_write(&file, ptr, sizeof(uint16_t)*length, (void *) &byteswriten);
    f_close(&file);
    return (byteswriten == sizeof(uint16_t)*length);
}
```

6.5. OBTENCIÓN DE LAS IMÁGENES DE TEST

Una forma de comprobar que la configuración de los periféricos (DCIM y DMA), I2C y SDRAM funciona es probar a guardar las imágenes de prueba que lleva el sensor. Estas imágenes sustituyen a los valores del ADC que el sensor nos manda a través del bus.

Primero se resetea el sensor, poniendo a nivel bajo el pin de *reset*, para que los registros vuelvan a los valores por defectos. Se ajustan los distintos pines a los niveles necesarios para el funcionamiento y se activa el reloj. En los registros se configura que la imagen comience en la fila y columna 0, al tamaño como se especifica en la documentación se le resta 1 [73], se activa que la foto comience cuando el pin de *trigger* o disparo cambie de nivel y que solo haga una, *snapshot mode*, se desactiva el balance de negros para evitar problemas [74] y se activa el patrón de test.

```
static void dcmiPhoto(uint16_t height, uint16_t width, uint8_t pll){
    /* Reset sensor*/
    HAL_GPIO_WritePin(DCMI_RESET_GPIO_Port, DCMI_RESET_Pin, 0);
    HAL_Delay(100);

    /* Reset and address pin*/
    HAL_GPIO_WritePin(DCMI_RESET_GPIO_Port, DCMI_RESET_Pin, 1);
    HAL_GPIO_WritePin(DCMI_ADDR_GPIO_Port, DCMI_ADDR_Pin, 1);
    HAL_GPIO_WritePin(DCMI_STANDBY_GPIO_Port, DCMI_STANDBY_Pin, 1);
    HAL_GPIO_WritePin(DCMI_OE_GPIO_Port, DCMI_OE_Pin, 0);
```

```
    /* Clock must be on*/
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

    /* Size properties*/
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_ROW_START, 0);
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_START, 0);
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_ROW_SIZE, height-1);
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_SIZE, width-1);

    /* Snapshot mode*/
    HAL_GPIO_WritePin(DCMI_TRIGGER_GPIO_Port, DCMI_TRIGGER_Pin, 1);
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_READ_MODE_1, 0);

    /* Disable Row BLC */
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_READ_MODE_2, 0);

    /* Set Test Image*/
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_TEST_PATTERN,
        MT9P031_TEST_PATTERN_VER_COLOR | MT9P031_TEST_PATTERN_ENABLE);

    /* Resume DCIM at frame 0*/
    MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_FRAME_RESTART,
        MT9P031_FRAME_RESTART_RESTART);
}
```

En la figura X1, se ve el patrón de colores ya aplicado el *demosaicing*; en la X2, el patrón de líneas monocolor verticales y en la X3 con horizontales.

Figura 27: Test rayas verticales.

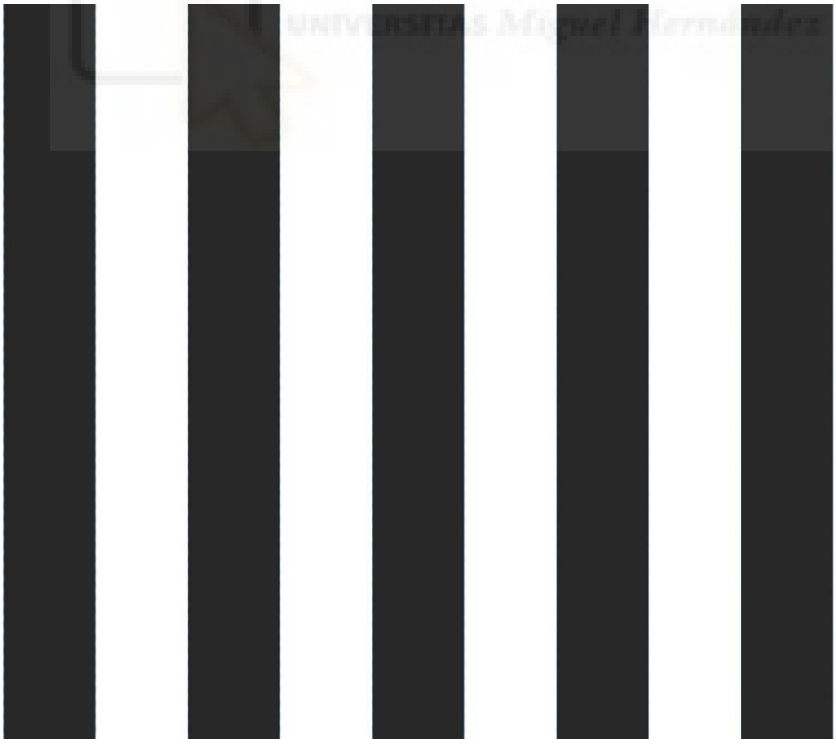


Figura 28: Test líneas monocolor verticales

Figura 29: Test líneas monocolor horizontales

6.6. AUMENTO DE LA FRECUENCIA DE PÍXEL

Una vez conseguido recibir datos con una frecuencia de píxel de 6MHz, elevamos está usando el PLL. En el datasheet [75] indica que la fórmula para calcular la frecuencia de píxel a partir de la frecuencia externa es:

$$f_{PIX} = f_{EXT} \frac{M}{N\ P_1}$$

Con las siguientes consideraciones:

$$16 \leq M \leq 255$$

$$2\ MHz \leq \frac{f_{EXT}}{N} \leq 13.5\ MHz$$

$$180\ MHz \leq f_{EXT} \frac{M}{N} \leq 360\ MHz$$

Además, recomiendan que $\frac{f_{EXT}}{N}$ tenga el valor mayor posible y P_1 sea par.

A continuación, se muestran algunos cálculos que se han utilizado para calcular una frecuencia de píxel de 24 y 80MHz (a partir de $f_{EXT} = 6\ MHz$).

En ambos casos se utiliza $N = 1$ para mantener $\frac{f_{EXT}}{N}$ lo más grande posible.

$$f_{PIX} = 6\ MHz \frac{M}{P_1} \rightarrow M = \frac{f_{PIX}}{6\ MHz} P_1 = M$$

$$180\ MHz \leq 6MHz\ M \leq 360\ MHz\ \rightarrow 180\ MHz \leq f_{PIX}\ P_1 \leq 360\ MHz$$

$$f_{PIX} = 24MHz \rightarrow 180\ MHz \leq 24MHz\ P_1 \leq 360\ MHz\ \rightarrow 7.5\ \leq\ P_1 \leq 15$$

$$f_{PIX} = 80MHz \rightarrow 180\ MHz \leq 80MHz\ P_1 \leq 360\ MHz\ \rightarrow 2.25\ \leq\ P_1 \leq 4.5$$

En el caso de $f_{PIX} = 24\ MHz$ se escoge un valor intermedio para el divisor P_1 de 10, $M = \frac{24\ MHz}{6\ MHz} 10 = 40$.

Para $f_{PIX} = 96\ MHz$ el único valor par de P_1 es 4, $M = \frac{80\ MHz}{6\ MHz} 4 = 53$.

El código para configurar el PLL es el siguiente (atención el reloj externo debe estar funcionando antes de configurarlo y hay que respetar el delay para que el *VLO* se estabilice):

```
MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONTROL,
MT9P031_PLL_CONTROL_PWRON);

MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONFIG_1,

    (M << MT9P031_PLL_CONFIG_1_M_Pos) & MT9P031_PLL_CONFIG_1_M_Msk);

MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONFIG_2, P1-1);

HAL_Delay(1); /* For ensure VCO has locked.*/

MT9P031_I2C_Write(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONTROL,
MT9P031_PLL_CONTROL_USEPLL | MT9P031_PLL_CONTROL_PWRON);
```

6.7. FALLO DE LA SDRAM

Durante las pruebas los valores en los archivos de imagen parecen extraños, son mayores de 4095 (2^12 - 1) bits. Este error parece que solo ocurre para los valores altos, es decir, cuando el sensor recibe mucha luz.

Primero se comprueba que los datos escritos en la tarjeta SD corresponden a los que hay en la SDRAM y se observa que los valores en la RAM también son mayores de 4095. Se descarta cualquier fallo en el periférico SDMMC o en la librería fatFS.

La siguiente prueba es comprobar que lo que hay en el registro *DR* (*Data Register*) del periférico DCIM datos es lo mismo que en la RAM, se configura el sensor para sacar una imagen con una resolución de 2x2 píxeles.

En la figura X se observa como en el registro se encuentra el valor 0xFF0F que corresponde a 4095 (tener en cuenta que este registro también es *little-endianess*, mientras que en las cuatro primeras posiciones el valor es 0xFF1F. Por tanto, el error parece estar en el bit 12 ya sea de la SDRAM, del DMA o de su buffer FIFO.

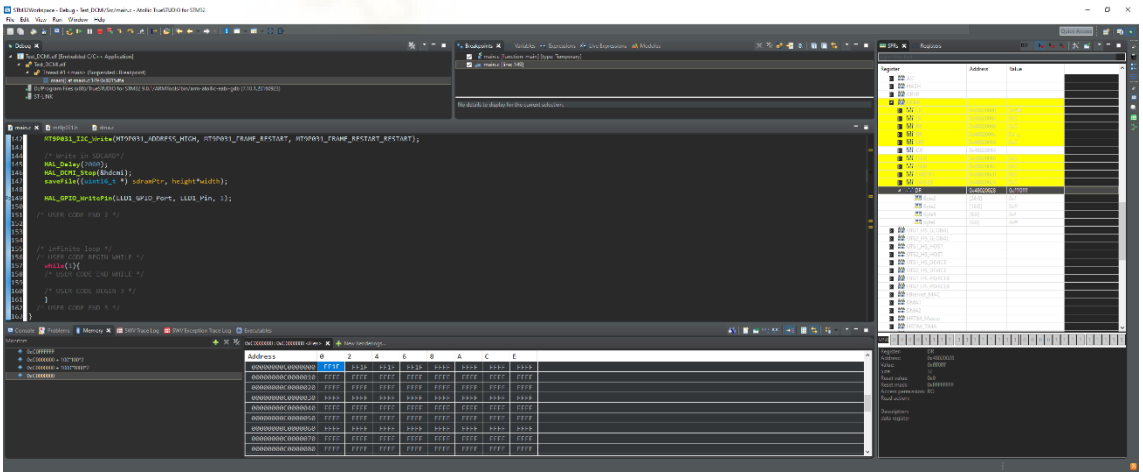


Figura 30: Valores en la SDRAM diferentes que en el registro DCMI.

Lo siguiente es comprobar que la memoria funciona para eso pasamos el mismo test que escribe y lee de forma secuencial que en el apartado 6.3. La prueba no

la consigue pasar con lo que algo relacionado con la memoria SDRAM está mal, parece que es en el bit 12, ya que la corrupción se produce en este (0b1FFF – 0x0FFF = 0x1000 = 1<<12).

Se crea un pequeño programa que escribe en las primeras 15 posiciones de memoria, un 1 desplazado su posición de memoria. De tal forma que en la posición 1 se escribe 0b0000 0001, en la 2 0b0000 0010 así hasta la 15 que corresponde a 0b1000 0000 0000 0000. De esta manera se puede ver que línea o líneas de datos están fallando.

En la Figura X se ve que el fallo está entre los bits 11 y 12, cuando escribimos un 1 en alguna de ellas lo escribe en las dos. Con el multímetro se mide que no haya continuidad entre ningún pista del bus de datos (D0…D15 en el esquemático). Además, en la figura X se ve claramente que ninguna soldadura está cortocircuitada.

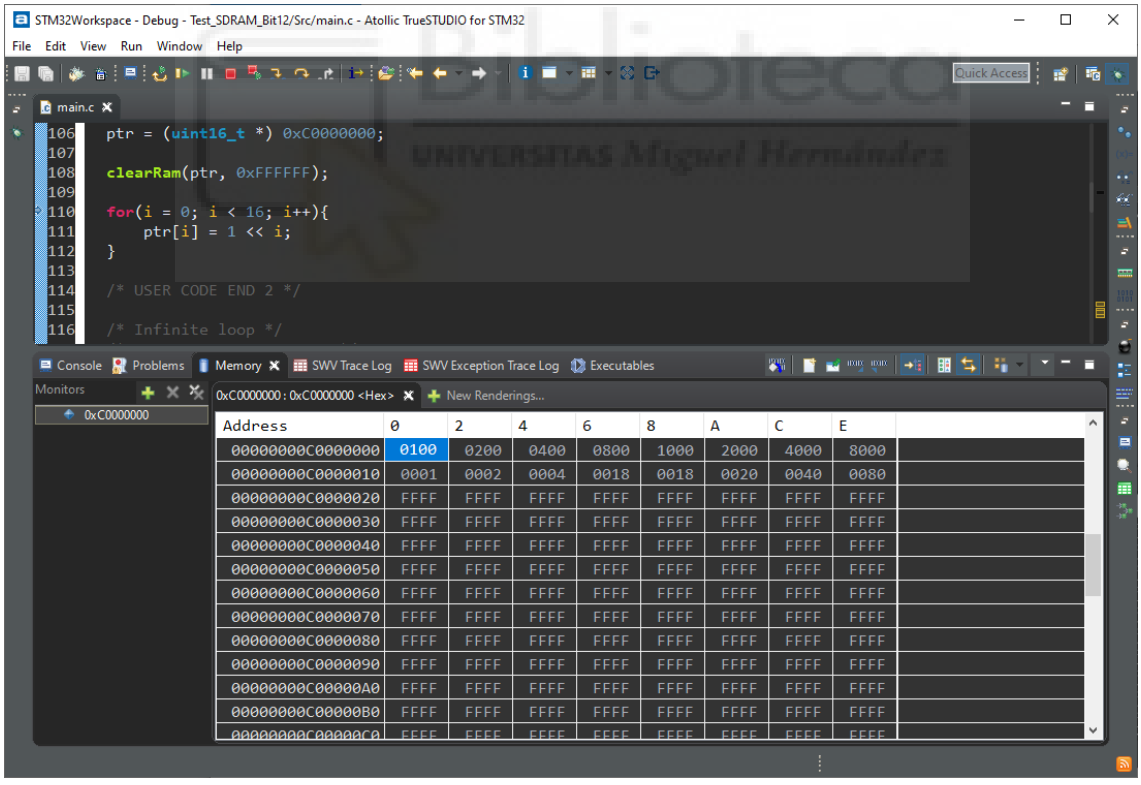


Figura 31: Problema en los datos de la SDRAM.

El componente debe haberse daño internamente, seguramente por alguna descarga electroestática a uno de esos dos pines, ya que el resto de los bits parecen funcionar bien.

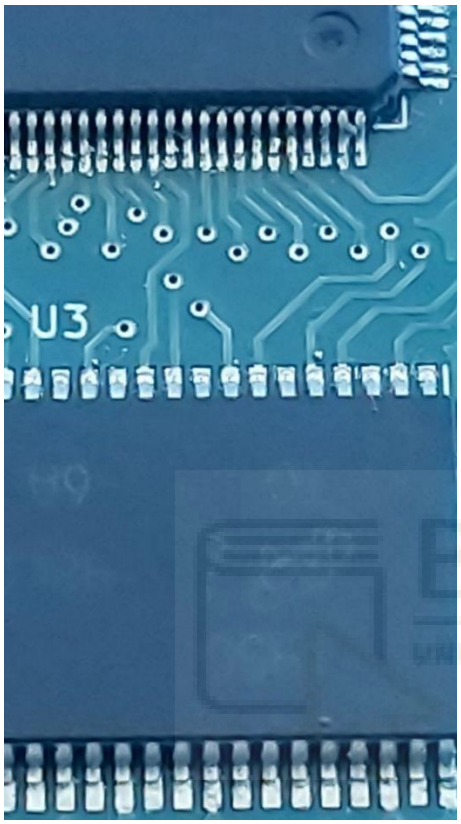


Figura 32: Soldaduras en la SDRAM y en el MCU.

Se pueden recuperar los datos del sensor CMOS de la memoria porque, por suerte, los 12 primeros bits están intactos y aplicando una máscara de 12 bits (está mascara es la operación AND entre el valor de cada píxel y el número 0b1111 1111 1111) se pueden guardar en la tarjeta uSD.

Pero de manera interna en el microcontroladores no se puede aplicar el *demosaicing* ya que los valores RGB-565 son de 16bits ni convertir a JPEG por la misma razón.

7. SOFTWARE

7.1. INTRODUCCIÓN

El software del microcontrolador del proyecto se ha desarrollado pensando en que sea lo más sencillo posible a la par que lo funcional. Se podría hacer más complejo para incluir más características como por ejemplo un procesado de imagen de más calidad, ajustes del brillo, contraste o colores, grabación de vídeo, conversión a otros formatos, modos de ahorro de energía, etc.

La parte desarrollada para las pruebas se ha aprovechado y se le añadido las funcionalidades descritas en el apartado 1.2. La resolución, tiempo de exposición y disparo de la foto se configura mediante botones o a través del puerto serie, e indica el estado con los LEDs. También el software aplica un algoritmo de *demosaicing* y las convierte a JPEG antes de guardarlas en memoria.

7.2. DRIVER DEL SENSOR

Las funciones para controlar el sensor en las pruebas de los apartados anteriores se han agrupado y modificado para simplificar el acceso al sensor desde el programa y mantener el código más ordenado. La inicialización de los pines y registros se realiza mediante la función *mt9p031Init*.

Para cambiar la resolución a la función *mt9p031ChangeResolution* se le pasa como parámetro el ancho y el alto y está se encarga de escribirlos en los registros correspondientes. En la Tabla 5 se muestra las 4 resoluciones entre las que el usuario puede escoger a la hora de hacer una fotografía.

Tabla 5: Resoluciones escogidas.

Ancho (píxeles)	Alto (píxeles)	Resolución aproximada (MP)

2592	1944	5
2048	1536	3.1
1600	1200	1.9
800	600	0.5

El usuario también puede elegir entre 4 tiempos de exposición distintos. El tiempo de 1/125s o 8ms está pensado para fotografías en movimiento como las de deporte, el de 1/60s o 17ms (aprox.) es un tiempo bastante estándar para todo tipo de fotografías, el de 4s se suele utilizar para paisajes en la naturaleza y, por último, un tiempo de larga exposición, 15s, para fotografías en situaciones de poca luminosidad o cuando se quiere conseguir algún tiempo de efecto con el tiempo. Estos tiempos se guardan en la variable *mt9p031ExpTime,* por tanto, para cambiar el tiempo solo hay que modificarla.

El tiempo de exposición lo aplica la función *mt9p031Photo* al pin de *Trigger* manteniéndolo a nivel bajo. A esta función se le pasa como parámetro la dirección donde se guarda la imagen y se encarga de iniciar y parar el periférico encargado de leer los datos del sensor.

Mientras que la variable *mt9p031FrameComplete* es 0, la función *mt9p031Photo* se queda bloqueada esperando a que se complete la transmisión de la imagen. Se ha redefinido la función *HAL_DCMI_FrameEventCallback* para que cuando la complete cambia el valor de *mt9p031FrameComplete* a 1.

7.3. DEMOSAICING Y ESPACIOS DE COLORES

Como se explicó en la sección 2.1 cada píxel del sensor de imagen solo capta un determinado color. Para obtener la información de todos los colores en un píxel, hay que combinar el valor de este píxel con la de los píxeles adyacentes.

Esta información hay que transformarla a un espacio de color que soporte el codificador JPEG del microcontrolador STM32H743.

Un espacio de color se define como un modelo matemático abstracto que describe la forma en que los colores se representan como tuplas de números [76]. Por ejemplo, el espacio de color RGB define un color como los porcentajes de tonos rojos, verdes y azules mezclados [77] o el YCbCr define Y como la componente de luminosidad mientras que las señales C_B y C_R son los componentes de crominancia diferencia de azul y diferencia de rojo, respectivamente [78].

Hay que diferenciar entre la compresión JPEG y el formato de intercambio de archivos TIFF JPEG. Se puede comprimir con el método JPEG distintos espacios de colores, pero el estándar TIFF JPEG define que el espacio de color ha de ser obligatoriamente YCbCr [79].

La documentación del codificador del micro STM32H743 indica que puede codificar (o comprimir) una imagen a partir de cualquier de los espacios de colores RGB, YCbCr, YCMK o BW (colores grises), además de generar la cabecera del archivo para dicha imagen [80]. Como es lógico, se espera que independientemente del espacio de color, ya que el codificador está pensado para escribir directamente en un archivo, se convierta mediante al hardware al espacio YCbCr. A pesar de todo esto resulta que la conversión no se realiza mediante hardware, si no por software, suponemos que es debido a la alta flexibilidad de conversión de espacio de colores que ofrece realizarlo por software.

Por tanto, hay que transformar “manualmente” la información del sensor a YCbCr antes de mandarla al codificador. La forma más sencilla es realizarlo en dos pasos: primero se convierte la imagen del sensor a RGB565 o RGB888 y después a YCbCr. La transformación de RGB a YCbCr según el standard JIFF se realiza de la siguiente forma [81]:

$$Y = 0.299R + 0.587G + 0.114B$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 128$$

$$Cr = 0.5R - 0.4187G - 0.0813B + 128$$

STMicroelectronics para facilitar aún más esta tarea proporciona una librería en C que permite transformar de un espacio de color RGB888, RGB565 o ARGB8888 a YCbCr [82].

La otra forma es convertir la imagen del sensor directamente a YCbCr. Según el *paper* de C. Doutre *A Fast Demosaicking Method Directly Producing YCbCr 4:2:0 Output*, el algoritmo está basado en obtener el gradiente del color verde, filtrar con un filtro paso-bajo los rojos y azules para reducir el número de muestras y convertir los filtros YCbCr con las ecuaciones que se han mostrado más arriba. Según comenta el autor, la ventaja de este método en comparación a hacerlo en dos pasos es que se reduce a la mitad el número de multiplicaciones (de 9 a 5.75) y de desplazamiento de bits (de 6 a 3.5) necesarias por píxel sin aumentar el ruido en la imagen.

El reducir el número de operaciones supone una gran ventaja en el caso de usar procesadores que no tiene una gran capacidad de cálculo o a la hora de procesar muchas imágenes como por ejemplo a la hora de grabar vídeo. En esta aplicación, siendo un trabajo académico, no merece la pena complicarse con implementar un algoritmo tan complejo, además el STM32H743 incluye una FPU (Unidad de procesado de flotantes) que puede realizar multiplicaciones y demás operaciones con números flotantes con una sola instrucción, con lo que la conversión de espacio de color se hará en dos pasos.

El sensor escogido, MT9P031, utiliza como CFA (*color filter array*) un filtro de Bayer. Este una filtro o máscara está formado por un 50 % de filtros que dejan pasar el color verde, un 25 % de rojos y un 25 % de azules, dispuestos en una

fila alternativamente azul y verde y en la siguiente fila verde y roja, como se muestra en la figura X.

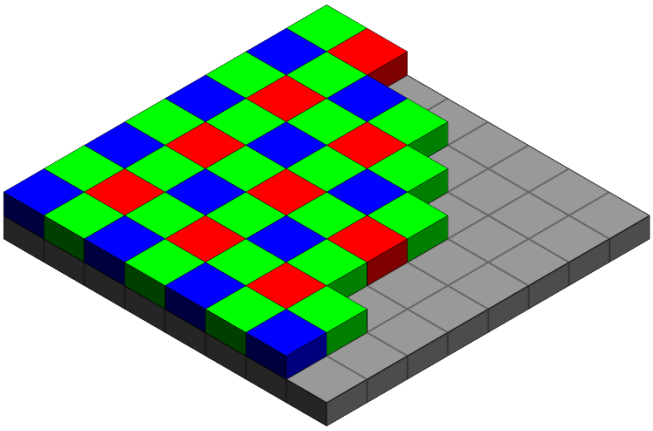


Figura 33: Máscara de Bayer. [F3]

La forma más sencilla de implementar el algoritmo de *demosaicing* es procesar cada píxel del sensor de 4 en 4, es decir agrupar en celdas dos píxeles verdes, uno rojo y uno azul en una celda. Cada celda es equivalente a un píxel de la imagen después del algoritmo, por tanto, el valor de R resultante es el valor del píxel rojo de la celda, el valor de B el del píxel azul y el de G es la media entre los dos píxeles verdes. Al procesarse de 4 en 4, este algoritmo genera una imagen de un cuarto de tamaño de la original y requiere que el ancho y el alto de la imagen sean múltiplos de 4.

Otra forma un poco más compleja para generar una imagen con las dimensiones de la original es procesar cada píxel individualmente usando los píxeles adyacentes para obtener la información de los colores distintos a ese color. Por ejemplo, en el caso de procesar un píxel de color rojo, se obtiene la información de R de este píxel, la de G haciendo la media de los píxeles verdes adyacentes al rojo (es decir, el de arriba, abajo, izquierda y derecha) y la de B igual que para el verde, pero utilizando los píxeles azules de la esquina. La dificultad de este algoritmo es que hay que evaluar individualmente cada caso dependiendo del color y sobre todo en la primera y última fila o columna y en las esquinas de la imagen.

Como se ha comentado en el apartado anterior se busca hacer el software lo más sencillo posible, con lo que se va a aplicar el primer algoritmo para convertir la imagen a RGB888.

En RGB888 como indica su nombre, cada color ocupa 8 bits (1 byte), por tanto, cada píxel ocupa 24 bits (3 bytes). Cada píxel de un color de la fotografía original ocupa 12 bits por lo que hay que reducir su tamaño a 8, la forma más eficiente es desplazando el valor 4 bits para la derecha. Con el color verde además hay que hacer la media entre los 2 píxeles, para ello, se suma el valor de los dos píxeles y para dividir entre 2 se aplica un desplazamiento hacia la derecha, entonces la operación, se resume a una suma y un desplazamiento de 5 bits hacia la derecha.

El código para todo este proceso se ha basado en la función de *demosaicing* del proyecto Siril [83]:

```
void demosaicingRGB888(uint16_t *buf, uint8_t *newbuf, uint16_t width,
uint16_t height) {
    uint32_t i = 0;
    for(int row = 0; row < height - 1; row += 2){
        for(int col = 0; col < width - 1; col += 2){
            newbuf[i + 0] = (uint8_t) ((buf[col + row * width] >> 4) & 0xFF);
            tmp = buf[1 + col + row * width];
            tmp += buf[col + (1 + row) * width];
            newbuf[i + 1] = (uint8_t) ((buf[1 + col + row * width] + buf[col +
            (1 + row) * width] >> 5) & 0xFF);
            newbuf[i + 2] = (uint8_t) ((buf[1 + col + (1 + row) * width] >> 4)
            & 0xFF);
        }
    }
```

El tamaño máximo de la fotografía original antes del *demosaicing* es de 10.077.696 bytes (2592 columnas x 1944 filas x 2 bytes). Si se guarda al principio de la memoria SDRAM, posición 0xC0000000, llega hasta la posición 0xC099C600 (10.077.696 = 0x0099C600). La imagen de después del demosaicing ocupa mucho menos, 3.779.136 bytes ((2592 / 2) columnas x (1944

/ 2) filas x 3 bytes). Se ha escogido que se guarde en el espacio entre las direcciones 0xC099C700 y 0xC0D7140 (0xC099C700 + 3.779.136). En la SDRAM queda un espacio libre de 0xC0FFFFF - 0xC0D37140 = 2.920.127 bytes.

7.4. CONVERSIÓN A JPEG

El periférico para codificar en JPEG está pensado para trabajar la imagen en bloques mediante un buffer de entrada y otro de salida. Esto permite utilizarlo en aplicaciones con poca memoria RAM o donde la velocidad de llegada o salida de los datos es distinta ya que los dos buffers no tienen por qué ser del mismo tamaño.

El buffer de entrada se utiliza para guardar la imagen una vez convertida de RGB888 a YCbCr420 mientras que el de salida está pensado para guardar la imagen en la RAM antes de guardarla en la tarjeta uSD. En una aplicación más avanza se podría hacer directamente sin buffer y que guardará en la memoria uSD, pero habría que modificar la implementación del módulo *FatFS*.

En nuestra aplicación, se ha decidido aprovechar el espacio de la SDRAM donde se han guardado los datos directos del sensor (dirección 0xC0000000) para ubicar los buffers. Con un tamaño de buffer de entrada de 3.779.136 bytes debe ser suficiente, e incluso menos para el de salida ya que está comprimida la imagen, pero se ha decidido, ya que nos sobra espacio, asignar 5 MB al de entrada y 5MB al de salida. Por tanto, el buffer de entrada empieza en la dirección 0xC0000000 y termina en la 0xC04C4B40 y el de salida desde la 0xC04C4C00 a la 0xC0989740.

Lo primero es indicarle los parámetros de la imagen, espacio de color que como se ha explicado es el YCbCr, el subsampleo que se ha escogido el 420 por ser el que menos espacio ocupa, el ancho, alto y la calidad de imagen en porcentaje, se ha elegido un 75%. Estos parámetros se pasan con la función *HAL_JPEG_ConfigEncoding*.

La función *JPEG_GetEncodeColorConvertFunc* nos devuelve el número de MCU (unidad mínima codificada como se indica en la especificación JPEG) y la función *RGBToYCbCr* para convertir de RGB a YCbCr a partir de los parámetros de la imagen. Con se ha indicado antes, los datos convertidos por esta función se guardan en el buffer de entrada.

Se inicia la conversión con la función *HAL_JPEG_Encode*, en nuestra aplicación, para simplificar la implementación, se ha configurado para que funcione de manera bloqueante (sin interrupciones ni DMA) con un tiempo máximo (*time out*) de 60 segundos. El periférico utiliza *callbacks* para indicar si ha habido un error en la conversión, si el buffer de entrada o de salida están llenos y para informar del final de la conversión.

El *callback HAL_JPEG_GetDataCallback* nos indica que el buffer de entrada está vacío y debemos comprobar si todos los datos se han pasado al periférico y si no pasarlos. *HAL_JPEG_DataReadyCallback* es igual, pero para el buffer de salida, aquí solo debemos indicar donde seguir guardando los datos [84]. El *callback HAL_JPEG_EncodeCpltCallback* es el que nos indica el final de la conversión y desde este se llama a la función que escribe el archivo. En la figura X se muestra un diagrama de flujo para ver de forma gráfica todo este proceso:

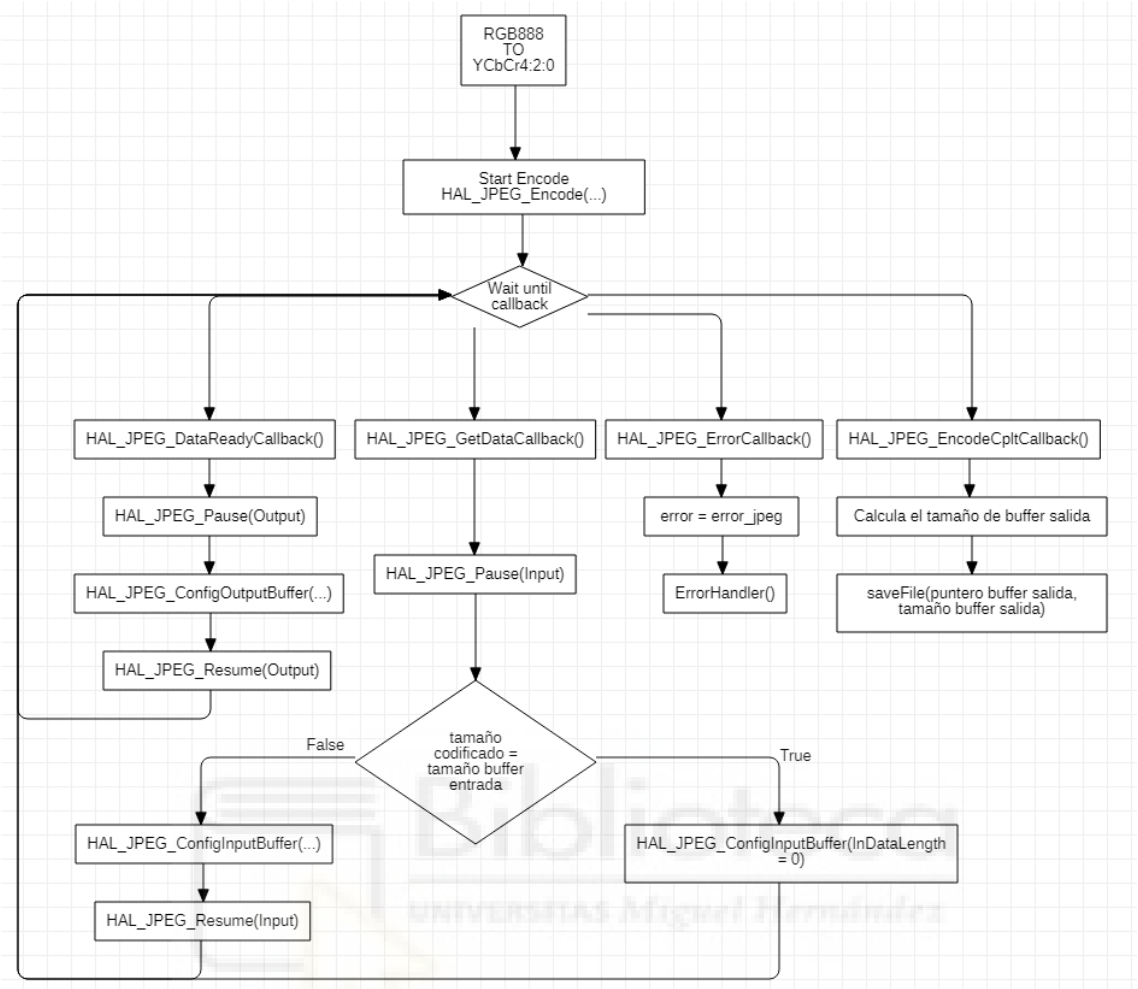


Figura 34: Diagrama de flujo del codificador de JPEG

7.5. GESTIÓN DE ARCHIVOS

Todos los archivos se van a guardar bajo el nombre “imageX.jpeg” siendo “X” “001” para la primera imagen, “002” para la segunda y así sucesivamente. Se ha creado la función *searchFile* para que busque el primer índice libre.

La función *saveFile* se llama cuando se completa la codificación JPEG para crear un nuevo archivo JPEG y guardar los datos. La otra función sendFile abre la última fotografía tomada y la manda por el puerto serie, para eso abre el archivo y va leyendo el archivo de 80 en 80 bytes hasta completar la transferencia. Se

ha decidido hacerlo de esta forma para evitar tener que guardar todo el archivo otra vez en la SDRAM.

7.7. BOTONES Y LEDS

El botón 1 se encarga de tomar la foto. Para ello primero llama a la función *mt9p031Photo*, luego a la función para aplicar el *demosaicing* y por último a la que codifica y guarda el archivo.

Los botones 2 y 3 utilizan las funciones *mt9p031ChangeResolution* y *mt9p031ChangeExpTime* pasando como argumento un "0" para cambiar la resolución y el tiempo de exposición, respectivamente. El botón 4 en esta aplicación se ha decidido no utilizar.

El LED 1 encendido indica que la fotografía ha comenzado y no se apaga hasta que el archivo se ha guardado. El LED 2 mediante parpadeos muestra la resolución seleccionada, lo mismo ocurre para el LED 3 con el tiempo de exposición y con el LED 4 para los errores. En la siguiente tabla se muestra el significado de los parpadeos.

LED	Número de parpadeos rápidos			
	1	2	3	4
2	Resolución 5MP	Resolución 3.1MP	Resolución 1.9MP	Resolución 0.5MP
3	Tiempo de exposición 1/125s	Tiempo de exposición 1/60s	Tiempo de exposición 4s	Tiempo de exposición 15s
84	Error de la codificación	Error de la memoria uSD	Otros errores	Otros errores

Tabla 6: Significado de los LEDs

7.8. PUERTO SERIE

Como se ha comentado anteriormente el puerto serie está pensado para cambiar la resolución y tiempo de exposición, hacer una fotografía, recibir la última fotografía en formato JPEG y ver el estado de los LEDs.

Se ha decidido que los comandos que se envían ocupen dos caracteres. Los comandos del “R1” al “R4” cambian la resolución, del “T1” a “T4” el tiempo de exposición y el comando “PH” toma una fotografía. Con “GP” envía la última fotografía tomada al ordenador, primero manda el tamaño en bytes del archivo y luego empieza a mandar el archivo, así la aplicación que se ejecuta en el ordenador sabe cuántos bytes tiene que guardar. Por último, el comando “RS” envía en el primer byte la resolución escogida (del 1 al 4) y en el segundo el tiempo de exposición (también del 1 al 4).

En la siguiente tabla se muestra un resumen de los comandos.

Comando	Explicación
R1	*Resolution 1.* Cambia la resolución a 5MP.
R2	*Resolution 2.* Cambia la resolución a 3.1MP.
R3	*Resolution 3.* Cambia la resolución a 1.9MP.
R4	*Resolution 4.* Cambia la resolución a 0.5MP.
T1	*Time exposure 1.* Cambia el tiempo de exposición a 1/125s.
T2	*Time exposure 2.* Cambia el tiempo de exposición a 1/60s.
T3	*Time exposure 3.* Cambia el tiempo de exposición a 4s.
T4	*Time exposure 4.* Cambia el tiempo de exposición a 15s.
PH	*Photo*. Toma una fotografía.
GP	*Get Photo.* Envía la última fotografía tomada.
RS	*Read Status.* Lee el estado de la cámara.

Tabla 7: Comandos del puerto serie

A la hora de recibir los comandos se utiliza el modo no bloqueante con interrupciones. De esta manera la aplicación se ejecuta de forma normal hasta que el *callback* de la interrupción indica que se ha recibido algún comando.

7.9. DIAGRAMA DE FLUJO

Para concluir el apartado de software, se expone el diagrama de flujo para mostrar la conexión entre las diferentes secciones del software.

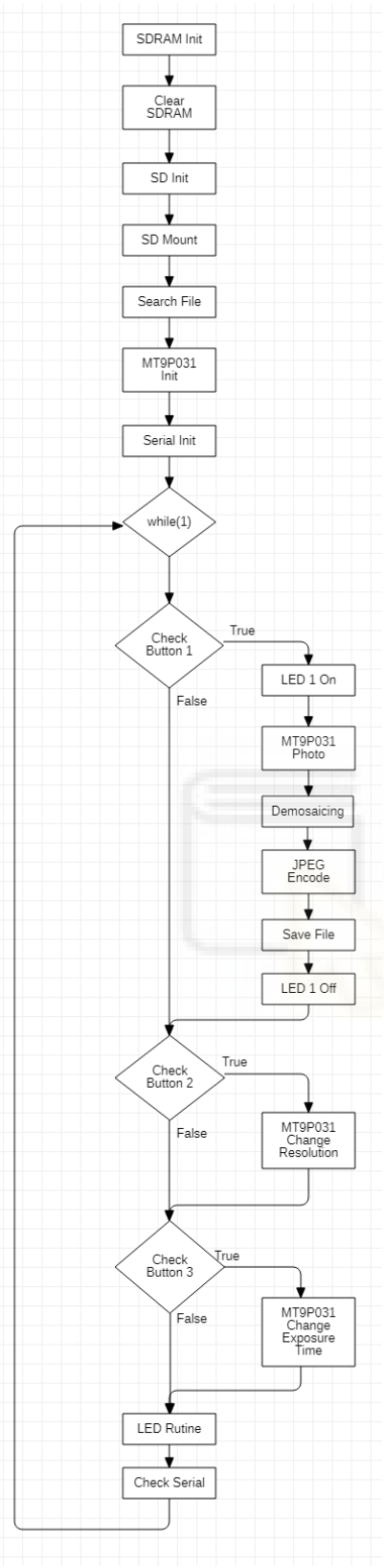


Figura 35: Diagrama de flujo del programa

8. CONCLUSIONES Y TRABAJOS FUTUROS

8.1. CONCLUSIONES

Terminado el proyecto, por el fallo que hubo en la memoria SDRAM el principal objetivo de una cámara de captación de imágenes que es hacer fotografías no se ha podida completar. De todas formas, se ha conseguido desarrollar un hardware que, siendo la primera vez que se desarrollaba, sin aplicar ninguna revisión ni habiendo utilizado antes electrónica tan rápida y crítica (memoria SDRAM a 166MHz, un sensor de imagen a 80MHz, LDOs de bajo ruido, etc.), ha sido capaz de funcionar en su totalidad.

Antes de que ocurriera el fallo, el sensor CMOS consiguió enviar la imagen de prueba al microcontrolador, demostrando así que las conexiones entre ellos eran correctas. Si el componente se hubiera roto antes y no viviéramos la situación de paralización tan atípica que ha ocurrido por el COVID-19, se podría haber desoldado y sustituido.

Desde el punto de vista del hardware, además ha quedado demostrado en la sección 6.1. que la memoria SDRAM, el sensor CMOS y el microcontrolador son capaces de soportar las velocidades necesarias para grabar vídeo con una resolución de 1080x1920 a 30fps.

En el desarrollo del software se han tenido en cuenta todos los objetivos requeridos para la interacción, por parte del usuario a través de los botones y LEDs y mediante la comunicación UART con el ordenador. La parte de *demosaicing* y codificación JPEG no ha podido ser probada por el fallo mencionado anteriormente, aun así, el algoritmo como se comentó en la sección 7.3. lo utiliza el proyecto *Siril* que es un software libre para el procesamiento de imágenes astronómicas y la codificación JPEG se ha desarrollado desde cero, pero de una forma muy similar a la utilizada en los ejemplos descritos en la nota de aplicación de *ST* AN4996 para la placa *STM32H753I-EVAL* [85].

Por último, hay que añadir que ha sido un proyecto muy completo y del que no puedo estar más que contento por tener la oportunidad de aplicar los conocimientos recibidos durante el grado en él.

He desarrollado desde cero una cámara digital que incluía muchas tecnologías y herramientas desconocidas. He aprendido mucho sobre el funcionamiento de los sensores de imagen CMOS, nunca había utilizado *KiCad* para diseñar una de PCB y menos una de 4 capas que incluye señales analógicas y digitales muy sensibles al ruido. También se ha reforzado los conocimientos teóricos en electrónica digital sobre los tipos de memoria y su funcionamiento a la hora de seleccionar y utilizar el módulo RAM y la memoria uSD.

Tanto las clases de programación en C que recibí durante el grado como las de microcontroladores *PSoC4* (*ARM Cortex-M0*) que se impartían en la especialidad de electrónica se han utilizado para aprender a programar una familia completamente nueva de microcontroladores (*STM32*) con un procesador, *ARM Cortex-M7*, muchísimo más potente y avanzado respecto al *PSoC*, utilizando un entorno de programación, *Atollic TrueStudio*, completamente nuevo. Además, los conocimientos de *Matlab* que se han dado a lo largo de muchas asignaturas han resultado críticos a la hora de utilizar las librerías *Numpy*, *MatplotLib* y *OpenCV* de Python durante las pruebas.

8.2. TRABAJOS FUTUROS

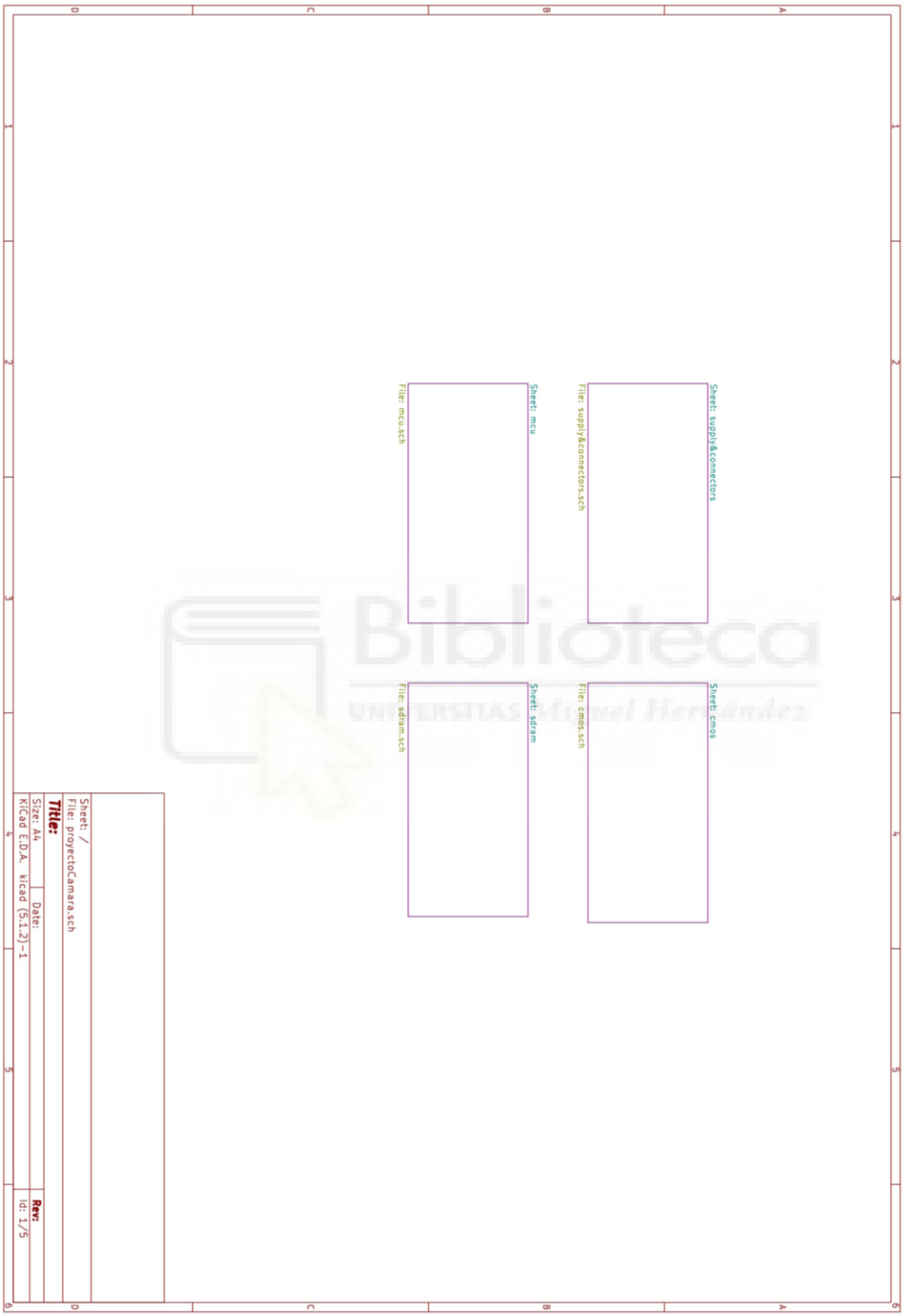
Como se ha comentado a lo largo del proyecto un par de mejoras que se le podrían hacer al software son:

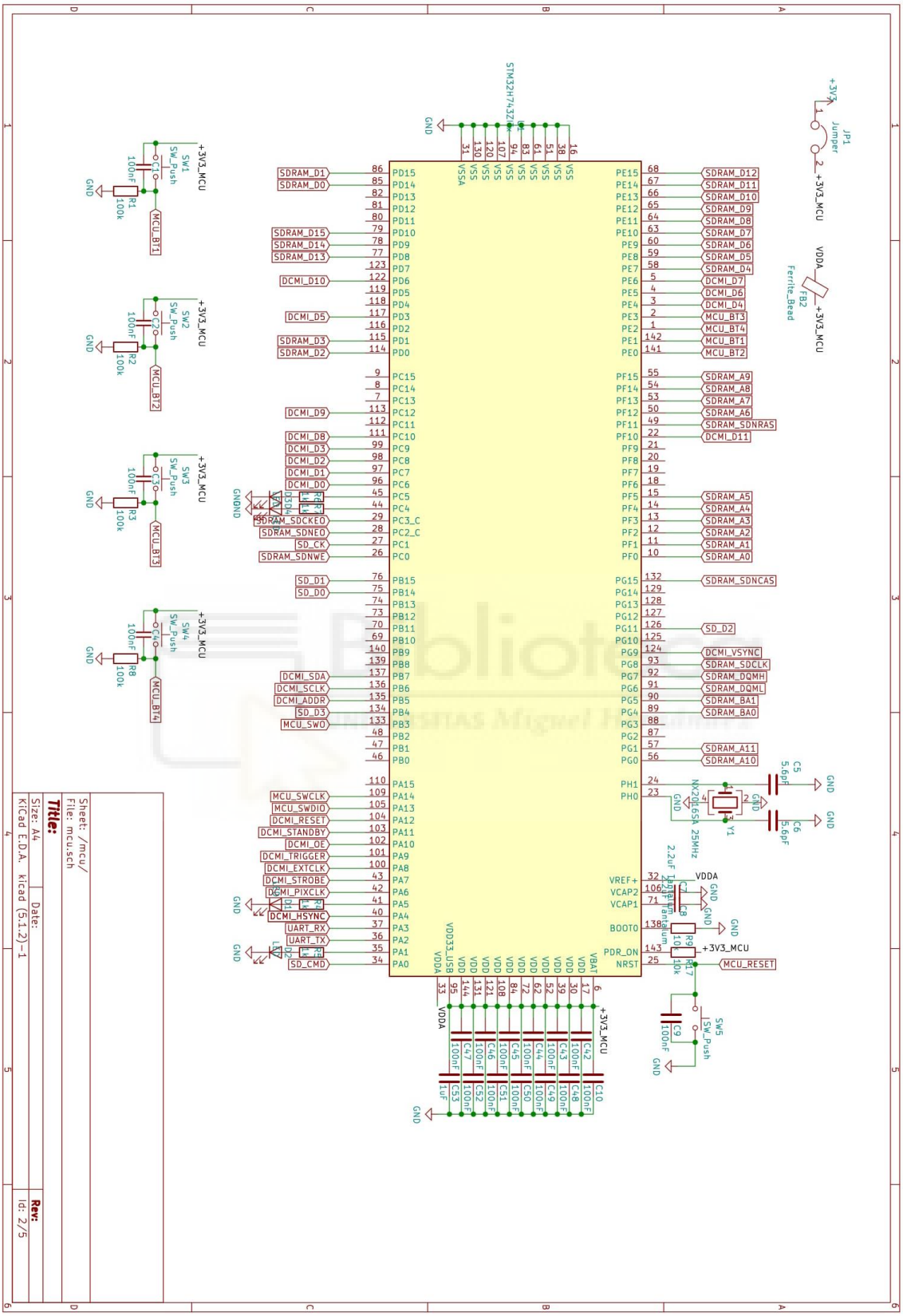
- Añadir grabación de vídeo en 1080p.
- Mejorar el procesado de imagen. Permitir ajustes del brillo, contraste o colores.
- Poder convertir las imágenes a otros formatos.
- Añadir modos de ahorro de energía.
- Utilizar el RTC en el nombre de los archivos de las imágenes de la tarjeta microSD.

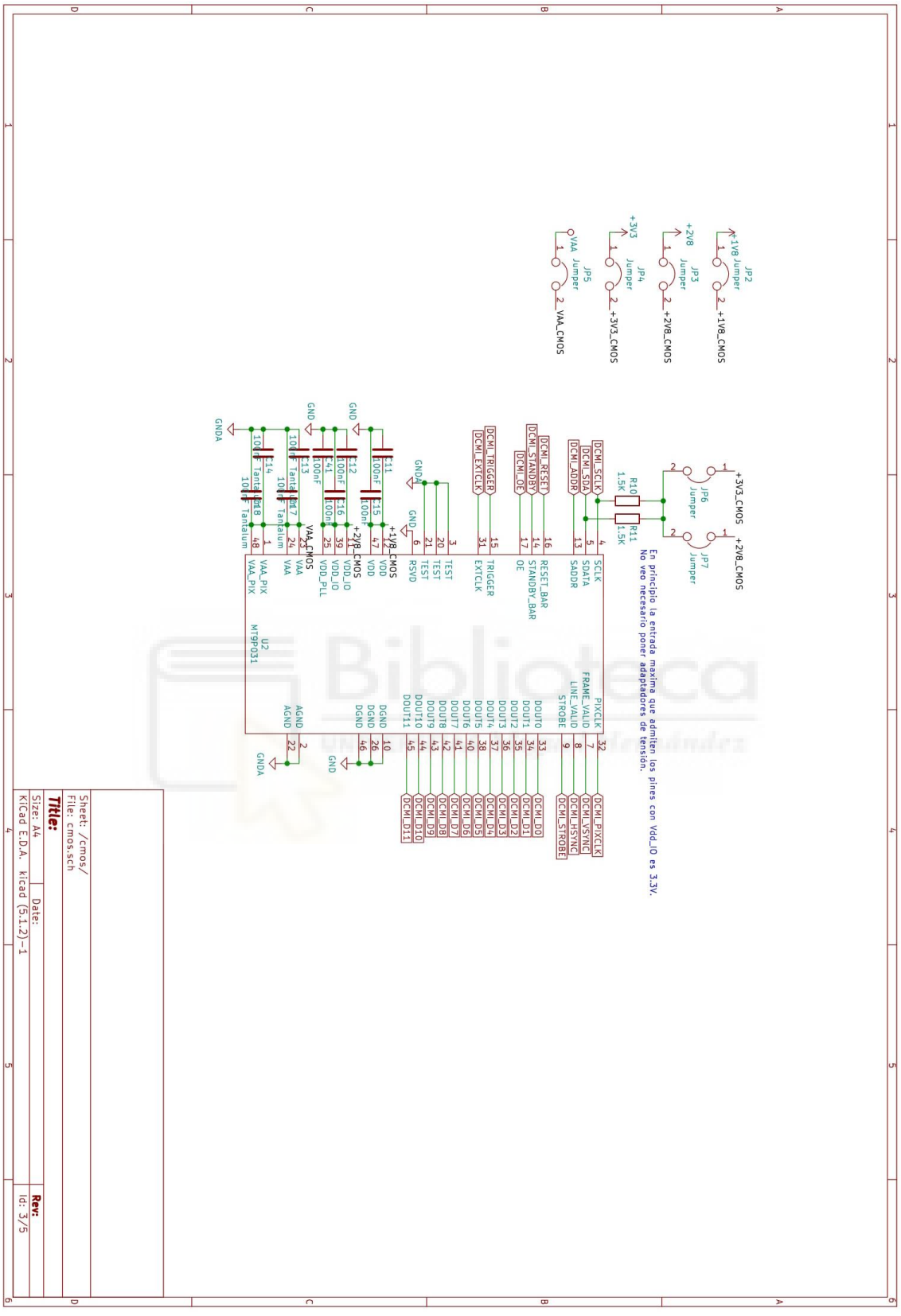
Para el hardware también se pueden añadir en el futuro cosas interesantes:

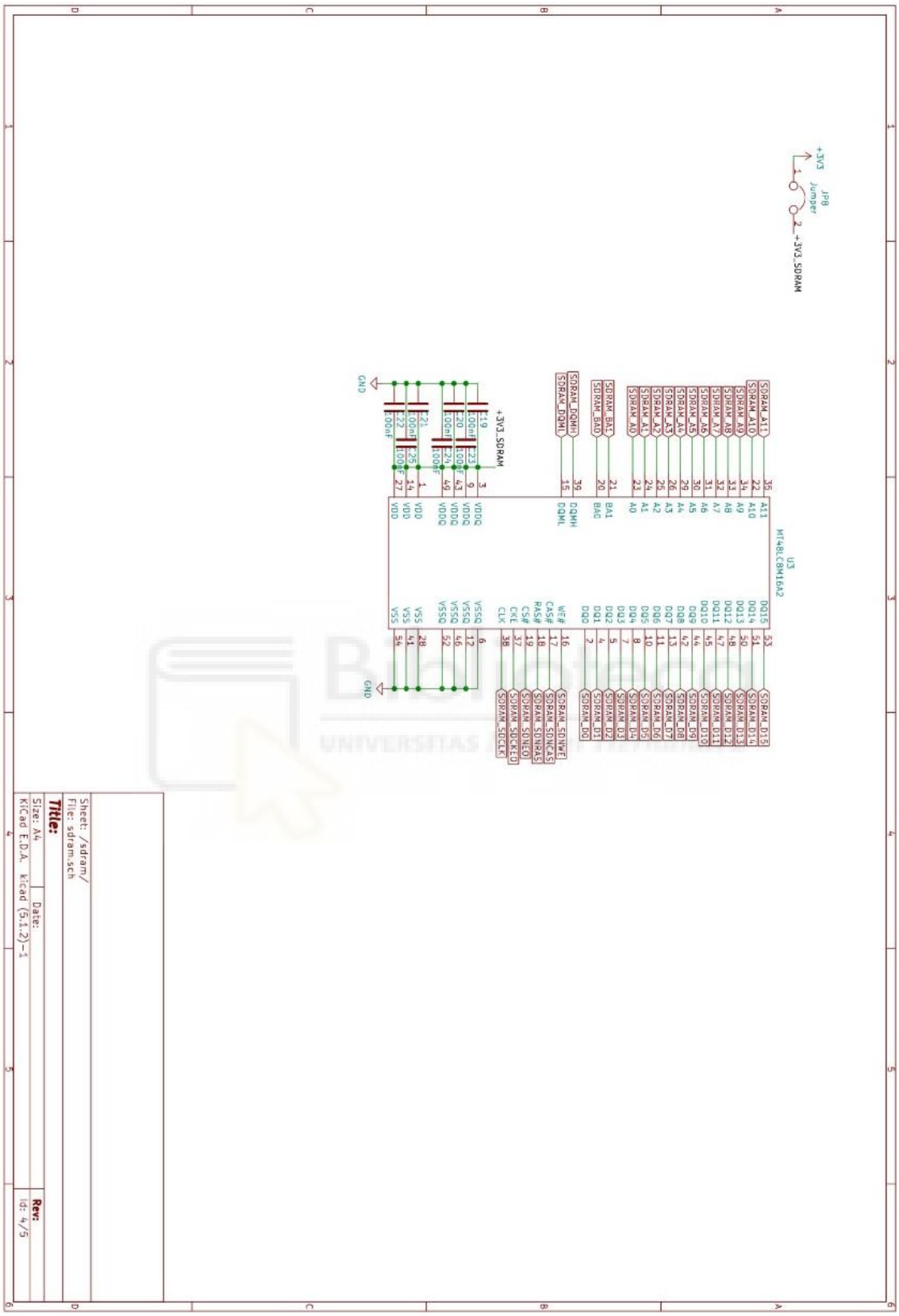
- Incluir una pantalla SPI para mostrar información y las imágenes captadas por el sensor.
- Montar una batería de ION-Litio con un cargador como puede ser el *BQ24092* de *Texas Instruments*.
- Ya que el *STM32H743* incluye un puerto USB, utilizarlo para la conexión a un ordenador.
- Si no se utilizará el puerto USB del microcontrolador, se puede añadir un conversor USB-UART *FTDI FT232H*.
- Diseñar una nueva PCB con otra distribución de los componentes y fabricar una carcasa de plástico con una impresora 3D.

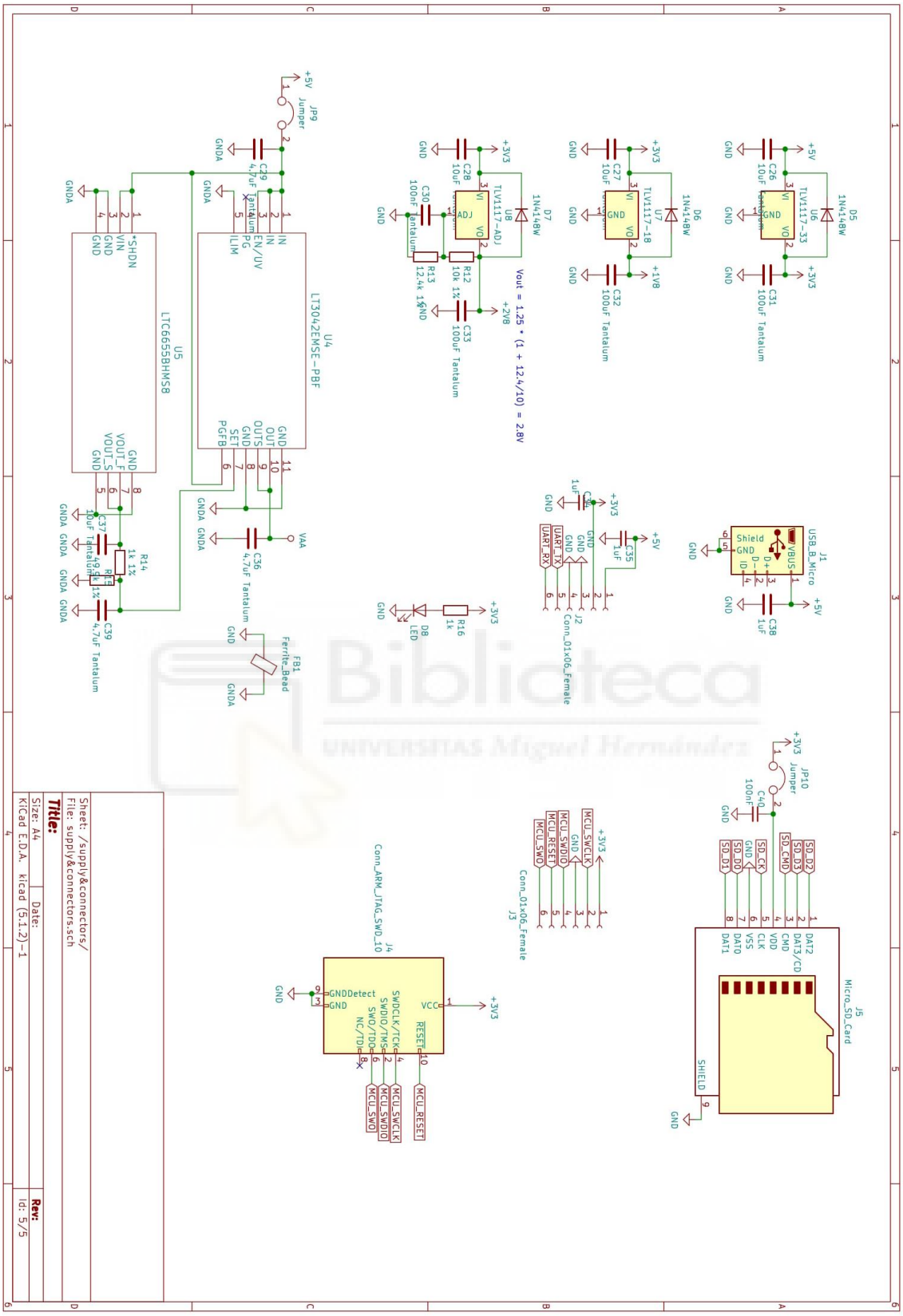
ANEXO A: ESQUEMÁTICO



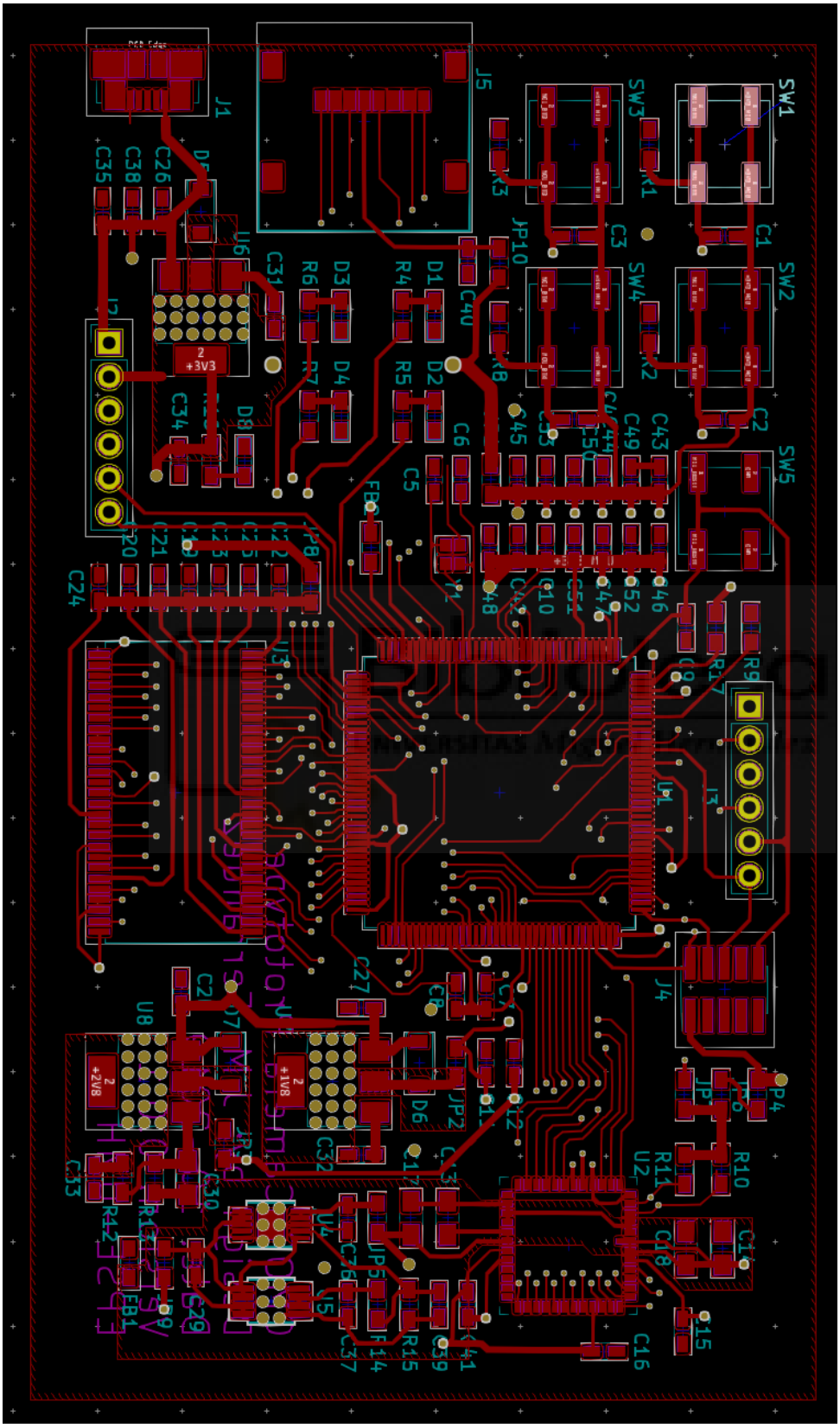

STM32H743ZIT
SDRAM_D1
SDRAM_D0
SDRAM_D15
SDRAM_D14
SDRAM_D13
DCMI_D10
DCMI_D5
SDRAM_D3
SDRAM_D2
DCMI_D9
DCMI_D8
DCMI_D3
DCMI_D2
DCMI_D1
DCMI_D0
SDRAM_SDCKE0
SDRAM_SDNE0
SD_CK
SDRAM_SDNWE
SD_D1
SD_D0
DCMI_SDA
DCMI_SCLK
DCMI_ADDR
SD_D3
MCU_SWO
MCU_SWCLK
MCU_SWDIO
DCMI_RESET
DCMI_STANDBY
DCMI_OE
DCMI_TRIGGER
DCMI_EXTCLK
DCMI_STROBE
DCMI_PIXCLK
DCMI_HSYNC
UART_RX
UART_TX
SD_CMD
SDRAM_D12
SDRAM_D11
SDRAM_D10
SDRAM_D9
SDRAM_D8
SDRAM_D7
SDRAM_D6
SDRAM_D5
SDRAM_D4
DCMI_D7
DCMI_D6
DCMI_D4
MCU_BT3
MCU_BT4
MCU_BT1
MCU_BT2
SDRAM_A9
SDRAM_A8
SDRAM_A7
SDRAM_A6
SDRAM_SDNRAS
DCMI_D11
SDRAM_A5
SDRAM_A4
SDRAM_A3
SDRAM_A2
SDRAM_A1
SDRAM_A0
SDRAM_SDNCAS
SD_D2
DCMI_VSYNC
SDRAM_SDCLK
SDRAM_DQMH
SDRAM_DQML
SDRAM_BA1
SDRAM_BA0
SDRAM_A11
SDRAM_A10
MCU_RESET
+3V3_MCU
VDDA
GND
JP1
Jumper
FB2
Ferrite_Bead
SW1
SW2
SW3
SW4
SW5
SW_Push
Y1
NX2016SA 25MHz
Sheet: /mcu/
File: mcu.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (5.1.2)-1
Rev:
Id: 2/5

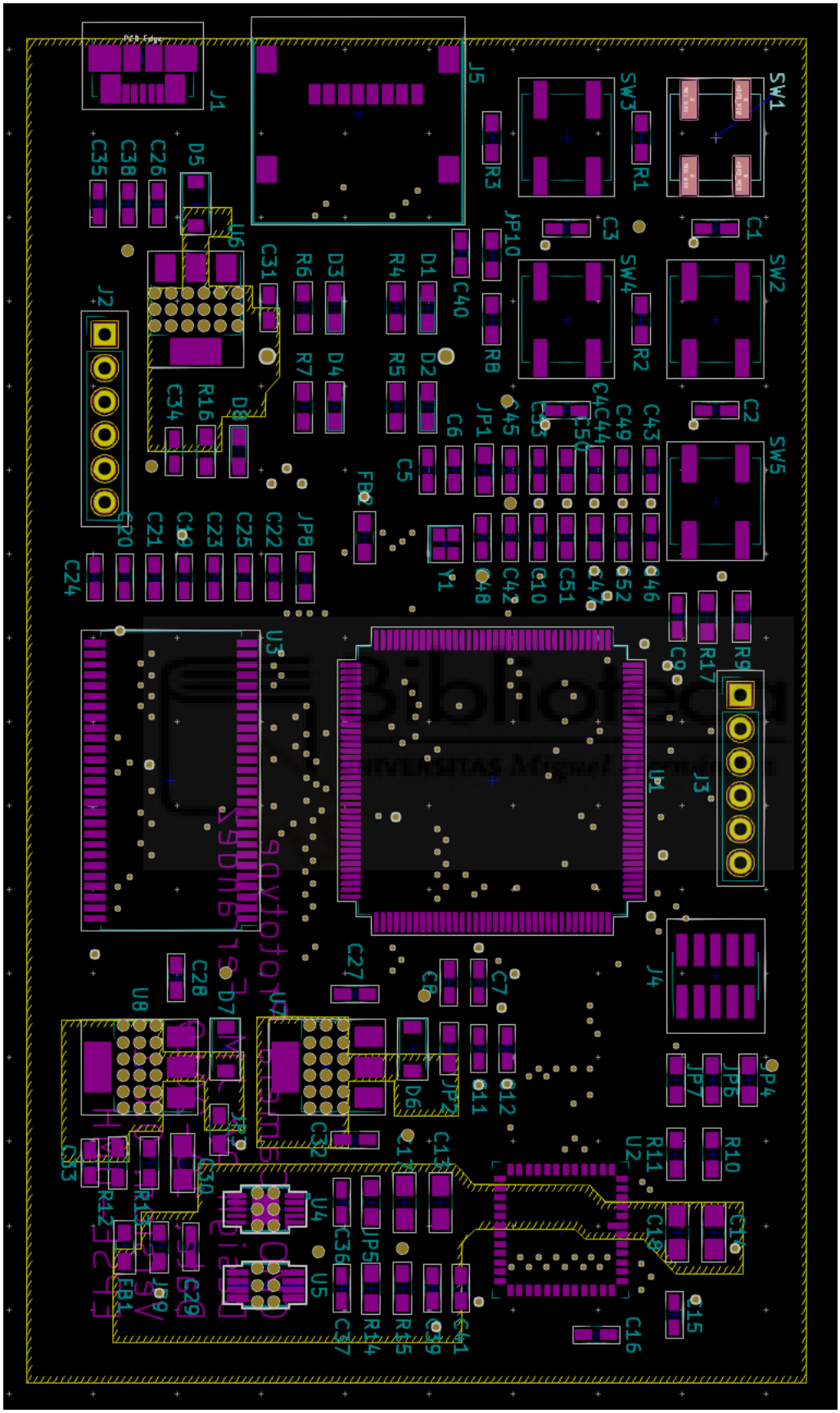

JP2
+1V8 Jumper
2 +1V8_CMOS
JP3
+2V8
Jumper
2 +2V8_CMOS
JP4
+3V3
Jumper
2 +3V3_CMOS
JP5
VAA Jumper
2 VAA_CMOS
JP6
Jumper
JP7
Jumper
+3V3_CMOS
+2V8_CMOS
R10
1.5K
R11
1.5K
En principio la entrada maxima que admiten los pines con Vdd_IO es 3.3V.
No veo necesario poner adaptadores de tensión.
DCMI_SCLK
DCMI_SDA
DCMI_SADDR
DCMI_RESET
DCMI_STANDBY
DCMI_OE
DCMI_TRIGGER
DCMI_EXTCLK
SCLK
SDATA
SADDR
RESET_BAR
STANDBY_BAR
OE
TRIGGER
EXTCLK
PIXCLK
FRAME_VALID
LINE_VALID
STROBE
DOUT0
DOUT1
DOUT2
DOUT3
DOUT4
DOUT5
DOUT6
DOUT7
DOUT8
DOUT9
DOUT10
DOUT11
DCMI_PIXCLK
DCMI_VSYNC
DCMI_HSYNC
DCMI_STROBE
DCMI_D0
DCMI_D1
DCMI_D2
DCMI_D3
DCMI_D4
DCMI_D5
DCMI_D6
DCMI_D7
DCMI_D8
DCMI_D9
DCMI_D10
DCMI_D11
TEST
TEST
TEST
RSVD
VDD
VDD
VDD
VDD_IO
VDD_PLL
VAA
VAA
VAA
VAA_PIX
VAA_PIX
DGND
DGND
DGND
AGND
AGND
GND
GNDA
+1V8_CMOS
+2V8_CMOS
VAA_CMOS
100nF
10uF Tantalum
U2
MT9P031
Sheet: /cmos/
File: cmos.sch
Title:
Size: A4
Date:
KiCad E.D.A. kicad (5.1.2)-1
Rev:
Id: 3/5

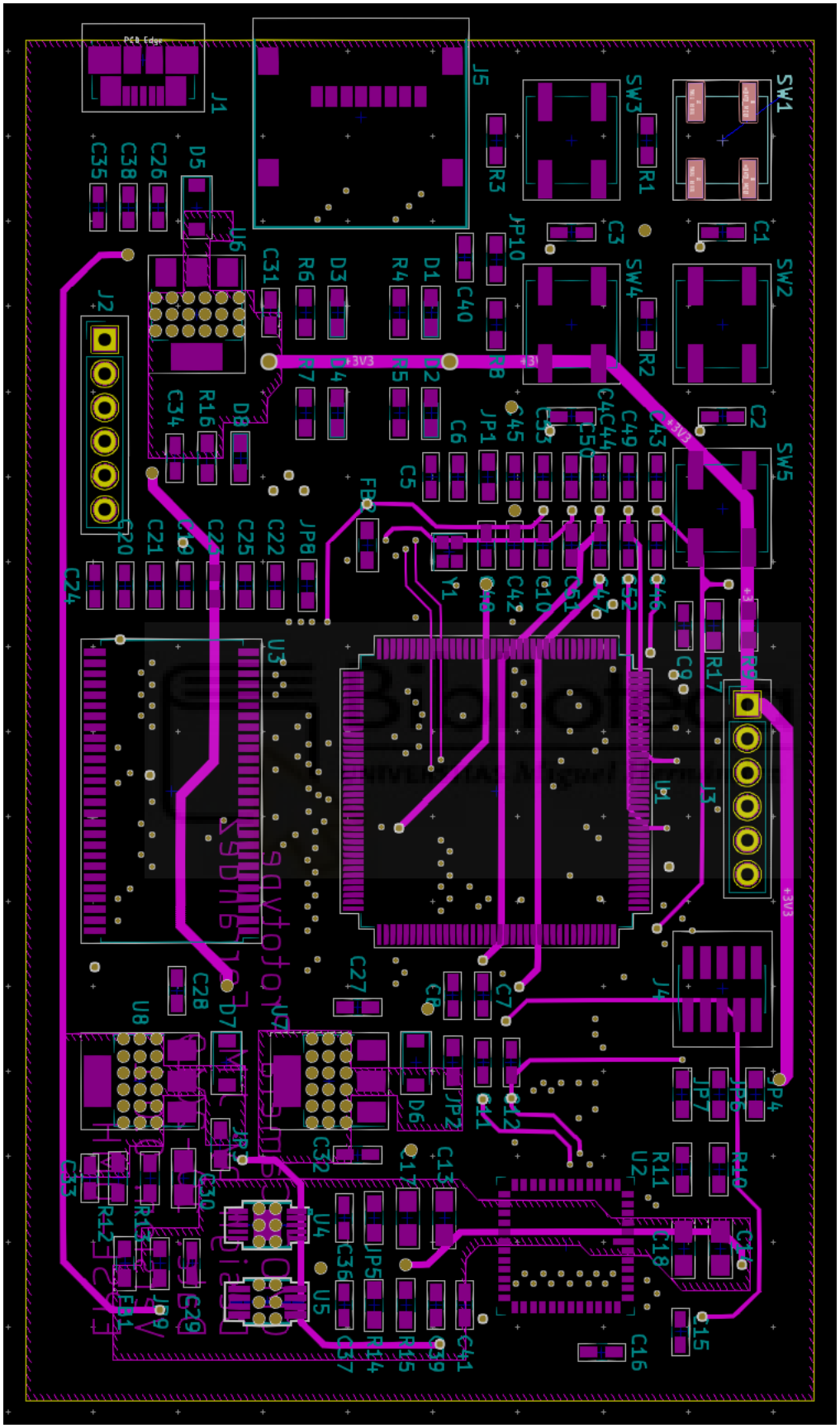

JP8
Jumper
+3V3
+3V3_SDRAM
U3
MT48LC8M16A2
SDRAM_A11
SDRAM_A10
SDRAM_A9
SDRAM_A8
SDRAM_A7
SDRAM_A6
SDRAM_A5
SDRAM_A4
SDRAM_A3
SDRAM_A2
SDRAM_A1
SDRAM_A0
SDRAM_BA1
SDRAM_BA0
SDRAM_DQMH
SDRAM_DQML
SDRAM_D15
SDRAM_D14
SDRAM_D13
SDRAM_D12
SDRAM_D11
SDRAM_D10
SDRAM_D9
SDRAM_D8
SDRAM_D7
SDRAM_D6
SDRAM_D5
SDRAM_D4
SDRAM_D3
SDRAM_D2
SDRAM_D1
SDRAM_D0
SDRAM_SDWE
SDRAM_SDNCAS
SDRAM_SDNRAS
SDRAM_SDNEO
SDRAM_SDCKE0
SDRAM_SDCLK
100nF
GND
Sheet: /sdram/
File: sdram.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (5.1.2)-1
Id: 4/5

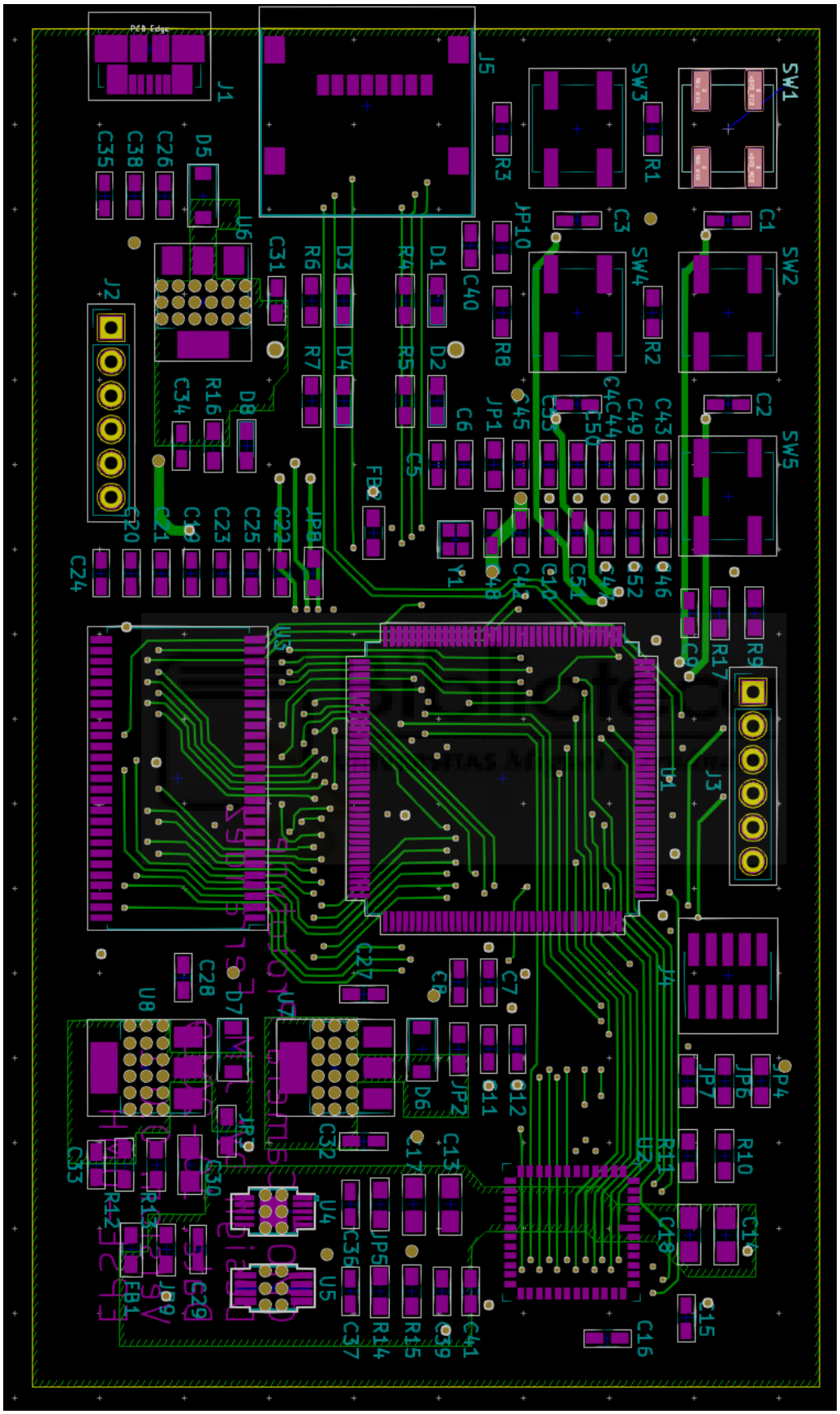


ANEXO B: PCB






PCB_Edge
J1
J5
SW1
SW2
SW3
SW4
SW5
R1
R2
R3
C1
C2
C3
JP10
C40
D1
D2
D3
D4
D5
R4
R5
R6
R7
R8
C31
C34
C35
C38
C26
U6
J2
D8
R16
C20
C21
C24
C25
C22
JP8
C6
C5
JP1
C45
C53
C50
C44
C49
C43
C42
C10
C51
C52
C46
C48
FB3
Y1
U3
U1
J3
R17
R9
C9
C27
C28
C8
C7
U8
D7
D6
JP2
C11
C12
J4
JP7
JP6
JP4
C32
C17
C13
U2
R11
R10
C18
C14
C33
C30
U4
U5
C36
JP5
R14
R15
C37
C39
C41
C16
C15
R12
R13
J9
FB1
C29
+3V3


PCB Edge
J1
J5
SW1
SW2
SW3
SW4
SW5
R1
R2
R3
R8
C1
C2
C3
JP10
D5
U6
J2
C35
C38
C26
C31
R6
D3
R4
D1
C40
R7
D4
R5
D2
C6
C5
JP1
C45
C39
C50
C44
C49
C43
C34
R16
D8
FB2
C20
C21
C19
C23
C25
C22
JP8
C24
Y1
C48
C42
C10
C51
C47
C52
C46
C9
R17
R9
U3
U1
J3
C27
C8
C7
C28
U8
D7
U7
D6
JP2
C11
C12
J4
JP7
JP6
P4
C32
C13
C17
R11
U2
R10
C33
C30
R12
U4
C36
JP5
C18
C14
J9
C29
U5
C37
R14
R15
C39
C41
C16
C15

ANEXO C: CÓDIGO FUENTE

Fichero main.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *                        opensource.org/licenses/BSD-3-Clause
 *
 ******************************************************************************
 */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "dcmi.h"
#include "dma.h"
#include "fatfs.h"
#include "i2c.h"
#include "jpeg.h"
#include "sdmmc.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "fmc.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
#include "mt9p031.h"
#include "buttonsLeds.h"
#include "filesystem.h"
#include "serial.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
static void bspSDRAMInit(SDRAM_HandleTypeDef *hsdram);
static void clearRam(uint16_t *pBuffer, uint32_t length);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
```

```
    * @brief  The application entry point.
    * @retval int
    */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration--------------------------------------------------------*/

    /* Reset of all peripherals, Initializes the Flash interface and the
    Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */
    HAL_I2C_DeInit(&hi2c1);
    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_FMC_Init();
    MX_DCMI_Init();
    MX_I2C1_Init();
    MX_TIM1_Init();
    MX_SDMMC2_SD_Init();
    MX_FATFS_Init();
    MX_JPEG_Init();
    MX_USART2_UART_Init();

    /* USER CODE BEGIN 2 */
```

```
/* Init SDRAM */
bspSDRAMInit(&hsdram1);
HAL_SDRAM_WriteProtection_Disable(&hsdram1);
clearRam((uint16_t*) 0xC0000000, 0xFFFFFF);

/* Init SDCARD*/
BSP_SD_Init();
f_mount(&SDFatFS, (TCHAR const*)SDPath, 0);

/* Search file*/
searchFile();

/* Init sensor*/
mt9p031Init();

/* Init serial data reception*/
initReception();

/* USER CODE END 2 */


/* Infinite loop */
/* USER CODE BEGIN WHILE */
while(1){
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    /* Button 1 used for take a photo */
    checkButton1();

    /* Button 2 used for change resolution */
    checkButton2();

    /* Button 3 used for change exposure time */
    checkButton3();

    /* LED rutine*/
    led();
```

```
        /* Check serial*/
        checkSerial();
    }
    /* USER CODE END 3 */
}

/**
    * @brief System Clock Configuration
    * @retval None
    */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
    RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

    /** Supply configuration update enable
    */
    HAL_PWREx_ConfigSupply(PWR_LDO_SUPPLY);
    /** Configure the main internal regulator output voltage
    */
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE0);

    while(!__HAL_PWR_GET_FLAG(PWR_FLAG_VOSRDY)) {}
    /** Macro to configure the PLL clock source
    */
    __HAL_RCC_PLL_PLLSOURCE_CONFIG(RCC_PLLSOURCE_HSE);
    /** Initializes the CPU, AHB and APB busses clocks
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 5;
    RCC_OscInitStruct.PLL.PLLN = 192;
    RCC_OscInitStruct.PLL.PLLP = 2;
    RCC_OscInitStruct.PLL.PLLQ = 20;
    RCC_OscInitStruct.PLL.PLLR = 2;
    RCC_OscInitStruct.PLL.PLLRGE = RCC_PLL1VCIRANGE_2;
```

```
RCC_OscInitStruct.PLL.PLLVCOSEL = RCC_PLL1VCOWIDE;
RCC_OscInitStruct.PLL.PLLFRACN = 0;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}
/** Initializes the CPU, AHB and APB busses clocks
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                            |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
                            |RCC_CLOCKTYPE_D3PCLK1|RCC_CLOCKTYPE_D1PCLK1;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.SYSCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB3CLKDivider = RCC_APB3_DIV2;
RCC_ClkInitStruct.APB1CLKDivider = RCC_APB1_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_APB2_DIV2;
RCC_ClkInitStruct.APB4CLKDivider = RCC_APB4_DIV2;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
{
    Error_Handler();
}
PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_USART2
                            |RCC_PERIPHCLK_SDMMC|RCC_PERIPHCLK_I2C1
                            |RCC_PERIPHCLK_FMC;
PeriphClkInitStruct.PLL2.PLL2M = 5;
PeriphClkInitStruct.PLL2.PLL2N = 166;
PeriphClkInitStruct.PLL2.PLL2P = 2;
PeriphClkInitStruct.PLL2.PLL2Q = 2;
PeriphClkInitStruct.PLL2.PLL2R = 5;
PeriphClkInitStruct.PLL2.PLL2RGE = RCC_PLL2VCIRANGE_2;
PeriphClkInitStruct.PLL2.PLL2VCOSEL = RCC_PLL2VCOWIDE;
PeriphClkInitStruct.PLL2.PLL2FRACN = 0;
PeriphClkInitStruct.FmcClockSelection = RCC_FMCCLKSOURCE_PLL2;
PeriphClkInitStruct.SdmmcClockSelection = RCC_SDMMCCLKSOURCE_PLL;
PeriphClkInitStruct.Usart234578ClockSelection =
                            RCC_USART234578CLKSOURCE_D2PCLK1;
PeriphClkInitStruct.I2c123ClockSelection = RCC_I2C123CLKSOURCE_D2PCLK1;
```

```
    if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
}

/* USER CODE BEGIN 4 */


/**
 * @brief  Perform the SDRAM external memory initialization sequence
 * @param  hsdram: SDRAM handle
 * @param  Command: Pointer to SDRAM command structure
 * @retval None
 */
static void bspSDRAMInit(SDRAM_HandleTypeDef *hsdram){
    __IO uint32_t tmpmrd = 0;
    FMC_SDRAM_CommandTypeDef command;
    /* Step 3:  Configure a clock configuration enable command */
    command.CommandMode = FMC_SDRAM_CMD_CLK_ENABLE;
    command.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
    command.AutoRefreshNumber = 1;
    command.ModeRegisterDefinition = 0;

    /* Send the command */
    if(HAL_SDRAM_SendCommand(hsdram, &command, SDRAM_TIMEOUT) == HAL_ERROR){
        Error_Handler();
    }

    /* Step 4: Insert 100 us minimum delay */
    /* Inserted delay is equal to 1 ms due to systick time base unit (ms) */
    HAL_Delay(1);

    /* Step 5: Configure a PALL (precharge all) command */
    command.CommandMode = FMC_SDRAM_CMD_PALL;
    command.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
    command.AutoRefreshNumber = 1;
    command.ModeRegisterDefinition = 0;
```

```
    /* Send the command */
    if(HAL_SDRAM_SendCommand(hsdram, &command, SDRAM_TIMEOUT) == HAL_ERROR){
        Error_Handler();
    }

    /* Step 6 : Configure a Auto-Refresh command */
    command.CommandMode = FMC_SDRAM_CMD_AUTOREFRESH_MODE;
    command.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
    command.AutoRefreshNumber = 8;
    command.ModeRegisterDefinition = 0;

    /* Send the command */
    if(HAL_SDRAM_SendCommand(hsdram, &command, SDRAM_TIMEOUT) == HAL_ERROR){
        Error_Handler();
    }

    /* Step 7: Program the external memory mode register */
    tmpmrd = (uint32_t) SDRAM_MODEREG_BURST_LENGTH_1 |
    SDRAM_MODEREG_BURST_TYPE_SEQUENTIAL |
    SDRAM_MODEREG_CAS_LATENCY_2 |
    SDRAM_MODEREG_OPERATING_MODE_STANDARD |
    SDRAM_MODEREG_WRITEBURST_MODE_SINGLE;

    command.CommandMode = FMC_SDRAM_CMD_LOAD_MODE;
    command.CommandTarget = FMC_SDRAM_CMD_TARGET_BANK1;
    command.AutoRefreshNumber = 1;
    command.ModeRegisterDefinition = tmpmrd;

    /* Send the command */
    if(HAL_SDRAM_SendCommand(hsdram, &command, SDRAM_TIMEOUT) == HAL_ERROR){
        Error_Handler();
    }

    /* Step 8: Set the refresh rate counter */
    /* (15.62 us x Freq) - 20 */
    /* Set the device refresh counter */
    hsdram->Instance->SDRTR |= ((uint32_t) ((1292) << 1));
}
```

```
static void clearRam(uint16_t *pBuffer, uint32_t length){
    uint32_t tmpIndex = 0;
    for(tmpIndex = 0; tmpIndex < (length / 2); tmpIndex++){
        pBuffer[tmpIndex] = 0xFFFF;
    }
}

/* USER CODE END 4 */

/**
    * @brief  This function is executed in case of error occurrence.
    * @retval None
    */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return
    state */
    while(1){
        errorLed();
    }
    /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
    * @brief  Reports the name of the source file and the source line number
    *         where the assert_param error has occurred.
    * @param  file: pointer to the source file name
    * @param  line: assert_param error line source number
    * @retval None
    */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line
     number, tex: printf("Wrong parameters value: file %s on line %d\r\n",
     file, line) */
    /* USER CODE END 6 */
```

```
}
#endif /* USE_FULL_ASSERT */

/********************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Fichero main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the
  *                   application.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
```

```
#include "stm32h7xx_hal.h"
#include "stm32h7xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define BT4_Pin GPIO_PIN_2
#define BT4_GPIO_Port GPIOE
#define BT3_Pin GPIO_PIN_3
#define BT3_GPIO_Port GPIOE
#define LED2_Pin GPIO_PIN_1
#define LED2_GPIO_Port GPIOA
#define LED1_Pin GPIO_PIN_5
#define LED1_GPIO_Port GPIOA
```

```
#define DCMI_STROBE_Pin GPIO_PIN_7
#define DCMI_STROBE_GPIO_Port GPIOA
#define LED4_Pin GPIO_PIN_4
#define LED4_GPIO_Port GPIOC
#define LED3_Pin GPIO_PIN_5
#define LED3_GPIO_Port GPIOC
#define FMC_DQML_Pin GPIO_PIN_6
#define FMC_DQML_GPIO_Port GPIOG
#define FMC_DQMH_Pin GPIO_PIN_7
#define FMC_DQMH_GPIO_Port GPIOG
#define DCMI_TRIGGER_Pin GPIO_PIN_9
#define DCMI_TRIGGER_GPIO_Port GPIOA
#define DCMI_OE_Pin GPIO_PIN_10
#define DCMI_OE_GPIO_Port GPIOA
#define DCMI_STANDBY_Pin GPIO_PIN_11
#define DCMI_STANDBY_GPIO_Port GPIOA
#define DCMI_RESET_Pin GPIO_PIN_12
#define DCMI_RESET_GPIO_Port GPIOA
#define DCMI_ADDR_Pin GPIO_PIN_5
#define DCMI_ADDR_GPIO_Port GPIOB
#define BT2_Pin GPIO_PIN_0
#define BT2_GPIO_Port GPIOE
#define BT1_Pin GPIO_PIN_1
#define BT1_GPIO_Port GPIOE
/* USER CODE BEGIN Private defines */
#define SDRAM_TIMEOUT     ((uint32_t)0xFFFF)

#define SDRAM_MODEREG_BURST_LENGTH_1             ((uint16_t)0x0000)
#define SDRAM_MODEREG_BURST_LENGTH_2             ((uint16_t)0x0001)
#define SDRAM_MODEREG_BURST_LENGTH_4             ((uint16_t)0x0002)
#define SDRAM_MODEREG_BURST_LENGTH_8             ((uint16_t)0x0004)
#define SDRAM_MODEREG_BURST_TYPE_SEQUENTIAL      ((uint16_t)0x0000)
#define SDRAM_MODEREG_BURST_TYPE_INTERLEAVED     ((uint16_t)0x0008)
#define SDRAM_MODEREG_CAS_LATENCY_2              ((uint16_t)0x0020)
#define SDRAM_MODEREG_CAS_LATENCY_3              ((uint16_t)0x0030)
#define SDRAM_MODEREG_OPERATING_MODE_STANDARD    ((uint16_t)0x0000)
#define SDRAM_MODEREG_WRITEBURST_MODE_PROGRAMMED ((uint16_t)0x0000)
#define SDRAM_MODEREG_WRITEBURST_MODE_SINGLE     ((uint16_t)0x0200)
/* USER CODE END Private defines */
```

```
#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

/********************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Fichero mt9p031.h

```
/*******************************************************************************
*
* mt9p031.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: MT9P031 drivers and access function. Requires STM32 Hal
*              drivers.
*
*******************************************************************************/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef MT9P031_H_
#define MT9P031_H_

/* Includes ------------------------------------------------------------------*/
/* System includes*/
#include <stdio.h>

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/
```

```
/* Type definitions ------------------------------------------------------*/
enum{
	MT9P031_RES_1 = 1,
	MT9P031_RES_2,
	MT9P031_RES_3,
	MT9P031_RES_4
};

enum{
	MT9P031_EXP_1 = 1,
	MT9P031_EXP_2,
	MT9P031_EXP_3,
	MT9P031_EXP_4
};

/* Variable declarations -------------------------------------------------*/
uint8_t mt9p031Exposure;
uint8_t mt9p031Resolution;
uint16_t mt9p031Width;
uint16_t mt9p031Height;

/* Function declarations -------------------------------------------------*/
void mt9p031Init();
void mt9p031ChangeResolution(uint8_t resolution);
void mt9p031ChangeExpTime(uint8_t expTime);
void mt9p031Photo(uint32_t dst);

/* Function definitions --------------------------------------------------*/


#endif /* MT9P031_H_ */
```

Fichero mt9p031.c

```
/*****************************************************************************
*
```

```
* mt9p031.c
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: MT9P031 drivers and access function. Requires STM32 Hal
*              drivers.
*
******************************************************************************/

/* Includes ------------------------------------------------------------------*/
/* Own include*/
#include "mt9p031.h"
/* Project includes*/
#include "main.h"
#include "tim.h"
#include "dcmi.h"
#include "i2c.h"
/* System includes*/
#include <stdio.h>
#include <string.h>

/* Defines -------------------------------------------------------------------*/
/* Maximum resolution*/
#define MT9P031_PIXEL_ARRAY_WIDTH           2752
#define MT9P031_PIXEL_ARRAY_HEIGHT          2004

/* Register definitions*/
#define MT9P031_ADDRESS_LOW                 0x90
#define MT9P031_ADDRESS_HIGH                0xBA
#define MT9P031_CHIP_VERSION                0x00
#define     MT9P031_CHIP_VERSION_VALUE      0x1801
#define MT9P031_ROW_START                   0x01
#define     MT9P031_ROW_START_MIN           0
#define     MT9P031_ROW_START_MAX           2004
#define     MT9P031_ROW_START_DEF           54
```

```
#define MT9P031_COLUMN_START                0x02
#define     MT9P031_COLUMN_START_MIN        0
#define     MT9P031_COLUMN_START_MAX        2750
#define     MT9P031_COLUMN_START_DEF        16
#define MT9P031_ROW_SIZE                    0x03
#define     MT9P031_ROW_SIZE_MIN            1
#define     MT9P031_ROW_SIZE_MAX            2005
#define     MT9P031_ROW_SIZE_DEF            1944
#define MT9P031_COLUMN_SIZE                 0x04
#define     MT9P031_COLUMN_SIZE_MIN         1
#define     MT9P031_COLUMN_SIZE_MAX         2751
#define     MT9P031_COLUMN_SIZE_DEF         2592
#define MT9P031_HORIZONTAL_BLANK            0x05
#define     MT9P031_HORIZONTAL_BLANK_MIN    0
#define     MT9P031_HORIZONTAL_BLANK_MAX    4095
#define MT9P031_VERTICAL_BLANK              0x06
#define     MT9P031_VERTICAL_BLANK_MIN      8
#define     MT9P031_VERTICAL_BLANK_MAX      2047
#define     MT9P031_VERTICAL_BLANK_DEF      26
#define MT9P031_OUTPUT_CONTROL              0x07
#define     MT9P031_OUTPUT_CONTROL_CEN      (1 << 1)
#define     MT9P031_OUTPUT_CONTROL_SYN      1
#define     MT9P031_OUTPUT_CONTROL_DEF      0x1f82
#define MT9P031_SHUTTER_WIDTH_UPPER         0x08
#define MT9P031_SHUTTER_WIDTH_LOWER         0x09
#define     MT9P031_SHUTTER_WIDTH_MIN       1
#define     MT9P031_SHUTTER_WIDTH_MAX       1048575
#define     MT9P031_SHUTTER_WIDTH_DEF       1943
#define MT9P031_PIXEL_CLOCK_CONTROL         0x0a
#define     MT9P031_PIXEL_CLOCK_INVERT      (1 << 15)
#define     MT9P031_PIXEL_CLOCK_SHIFT(n)    ((n) << 8)
#define     MT9P031_PIXEL_CLOCK_DIVIDE(n)   ((n) << 0)
#define MT9P031_FRAME_RESTART               0x0b
#define     MT9P031_FRAME_RESTART_TRIGGER   (1 << 2)
#define     MT9P031_FRAME_RESTART_PAUSE     (1 << 1)
#define     MT9P031_FRAME_RESTART_RESTART   (1 << 0)
#define MT9P031_SHUTTER_DELAY               0x0c
#define MT9P031_RST                         0x0d
#define     MT9P031_RST_ENABLE              1
```

```
#define     MT9P031_RST_DISABLE             0
#define MT9P031_PLL_CONTROL                 0x10
#define     MT9P031_PLL_CONTROL_PWROFF      0x0050
#define     MT9P031_PLL_CONTROL_PWRON       0x0051
#define     MT9P031_PLL_CONTROL_USEPLL      0x0052
#define MT9P031_PLL_CONFIG_1                0x11
#define     MT9P031_PLL_CONFIG_1_M_Pos      (8U)
#define     MT9P031_PLL_CONFIG_1_M_Msk      \
    (0xFFU << MT9P031_PLL_CONFIG_1_M_Pos)
#define     MT9P031_PLL_CONFIG_1_N_Pos      (0U)
#define     MT9P031_PLL_CONFIG_1_N_Msk      \
    (0x1FU << MT9P031_PLL_CONFIG_1_N_Pos)
#define MT9P031_PLL_CONFIG_2                0x12
#define     MT9P031_PLL_CONFIG_2_P1_Pos     (0U)
#define     MT9P031_PLL_CONFIG_2_P1_Msk     \
    (0x0FU << MT9P031_PLL_CONFIG_2_P1_Pos)
#define MT9P031_READ_MODE_1                 0x1e
#define     MT9P031_READ_MODE_1_XOR_LINE    (1 << 11)
#define     MT9P031_READ_MODE_1_CONT_LINE   (1 << 10)
#define     MT9P031_READ_MODE_1_INV_TRIGGER (1 << 9)
#define     MT9P031_READ_MODE_1_SNAPSHOT    (1 << 8)
#define     MT9P031_READ_MODE_1_GRR         (1 << 7)
#define     MT9P031_READ_MODE_1_BULB        (1 << 6)
#define     MT9P031_READ_MODE_1_INV_STROBE  (1 << 5)
#define     MT9P031_READ_MODE_1_STROBE_EN   (1 << 4)
#define MT9P031_READ_MODE_2                 0x20
#define     MT9P031_READ_MODE_2_ROW_MIR     (1 << 15)
#define     MT9P031_READ_MODE_2_COL_MIR     (1 << 14)
#define     MT9P031_READ_MODE_2_SHOW_DARK_COL (1 << 12)
#define     MT9P031_READ_MODE_2_SHOW_DARK_ROW (1 << 11)
#define     MT9P031_READ_MODE_2_ROW_BLC     (1 << 6)
#define     MT9P031_READ_MODE_2_COL_SUM     (1 << 5)
#define MT9P031_ROW_ADDRESS_MODE            0x22
#define MT9P031_COLUMN_ADDRESS_MODE         0x23
#define MT9P031_GLOBAL_GAIN                 0x35
#define     MT9P031_GLOBAL_GAIN_MIN         8
#define     MT9P031_GLOBAL_GAIN_MAX         1024
#define     MT9P031_GLOBAL_GAIN_DEF         8
#define     MT9P031_GLOBAL_GAIN_MULT        (1 << 6)
```

```
#define MT9P031_ROW_BLACK_TARGET            0x49
#define MT9P031_ROW_BLACK_DEF_OFFSET        0x4b
#define MT9P031_GREEN1_OFFSET               0x60
#define MT9P031_GREEN2_OFFSET               0x61
#define MT9P031_BLACK_LEVEL_CALIBRATION     0x62
#define     MT9P031_BLC_MANUAL_BLC          (1 << 0)
#define MT9P031_RED_OFFSET                  0x63
#define MT9P031_BLUE_OFFSET                 0x64
#define MT9P031_TEST_PATTERN                0xa0
#define     MT9P031_TEST_PATTERN_COLOR      0x00
#define     MT9P031_TEST_PATTERN_HORIZONTAL 0x08
#define     MT9P031_TEST_PATTERN_VERTICAL   0x10
#define     MT9P031_TEST_PATTERN_DIAGONAL   0x18
#define     MT9P031_TEST_PATTERN_CLASSIC    0x20
#define     MT9P031_TEST_PATTERN_WALKING    0x28
#define     MT9P031_TEST_PATTERN_MONO_HOR   0x30
#define     MT9P031_TEST_PATTERN_MONO_VER   0x38
#define     MT9P031_TEST_PATTERN_VER_COLOR  0x40
#define     MT9P031_TEST_PATTERN_ENABLE     (1 << 0)
#define     MT9P031_TEST_PATTERN_DISABLE    (0 << 0)
#define MT9P031_TEST_PATTERN_GREEN          0xa1
#define MT9P031_TEST_PATTERN_RED            0xa2
#define MT9P031_TEST_PATTERN_BLUE           0xa3
#define MT9P031_TEST_PATTERN_BAR_WIDTH      0xa4

/* Exposure time in ms*/
#define MT9P031_EXP_TIME_1_125 8
#define MT9P031_EXP_TIME_1_60  17
#define MT9P031_EXP_TIME_4     4000
#define MT9P031_EXP_TIME_15    15000

/* Resolution*/
#define MT9P031_WIDTH_1  2592
#define MT9P031_WIDTH_2  2048
#define MT9P031_WIDTH_3  1600
#define MT9P031_WIDTH_4  800
#define MT9P031_HEIGHT_1 1944
#define MT9P031_HEIGHT_2 1536
#define MT9P031_HEIGHT_3 1200
```

```
#define MT9P031_HEIGHT_4 600


/* Macro defines ------------------------------------------------------------*/


/* Variable declarations ----------------------------------------------------*/
static uint8_t mt9p031FrameComplete = 0;
uint32_t mt9p031ExpTime = MT9P031_EXP_TIME_1_125;
uint16_t mt9p031Width = MT9P031_WIDTH_1;
uint16_t mt9p031Height = MT9P031_HEIGHT_1;

/* Variable definitions -----------------------------------------------------*/
uint8_t mt9p031Exposure = MT9P031_EXP_1;
uint8_t mt9p031Resolution = MT9P031_RES_1;

/* Static function declarations ---------------------------------------------*/
static void mt9p031I2CWrite(uint8_t address, uint16_t reg, uint16_t data);
static void mt0p031I2CRead(uint8_t address, uint16_t reg, uint16_t* data);

/* Function definitions -----------------------------------------------------*/
void mt9p031Init(){
    /* Reset sensor*/
    HAL_GPIO_WritePin(DCMI_RESET_GPIO_Port, DCMI_RESET_Pin, 0);
    HAL_Delay(100);

    /* Reset and address pin*/
    HAL_GPIO_WritePin(DCMI_RESET_GPIO_Port, DCMI_RESET_Pin, 1);
    HAL_GPIO_WritePin(DCMI_ADDR_GPIO_Port, DCMI_ADDR_Pin, 1);
    HAL_GPIO_WritePin(DCMI_STANDBY_GPIO_Port, DCMI_STANDBY_Pin, 1);
    HAL_GPIO_WritePin(DCMI_OE_GPIO_Port, DCMI_OE_Pin, 0);

    /* Clock must be on*/
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

    /* Set chip disable*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_OUTPUT_CONTROL, 0);

    /* Size properties*/
```

```
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_ROW_START, 0);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_START, 0);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_ROW_SIZE, mt9p031Height-1);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_SIZE,
        mt9p031Width-1);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_HORIZONTAL_BLANK, 0);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_VERTICAL_BLANK, 0);

    /* PLL Settings
         m = 40
         n = 0
         p1 = 10 */
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONTROL,
        MT9P031_PLL_CONTROL_PWRON);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONFIG_1,
        (39 << MT9P031_PLL_CONFIG_1_M_Pos) & MT9P031_PLL_CONFIG_1_M_Msk);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONFIG_2, 9);
    HAL_Delay(1); /* For ensure VCO has locked.*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_PLL_CONTROL,
        MT9P031_PLL_CONTROL_USEPLL | MT9P031_PLL_CONTROL_PWRON);

    /* Snapshot mode*/
    HAL_GPIO_WritePin(DCMI_TRIGGER_GPIO_Port, DCMI_TRIGGER_Pin, 1);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_READ_MODE_1,
        MT9P031_READ_MODE_1_SNAPSHOT|MT9P031_READ_MODE_1_BULB);

    /* Roow BLC */
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_READ_MODE_2,
        MT9P031_READ_MODE_2_ROW_BLC);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_BLACK_LEVEL_CALIBRATION, 0);

    /* Column bin & column skip disable*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_ROW_ADDRESS_MODE, 0);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_ADDRESS_MODE, 0);

    /* Disable Test Image*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_TEST_PATTERN, 0);
}
```

```
void mt9p031ChangeResolution(uint8_t resolution){
    if(0 == resolution){
        mt9p031Resolution++;
        if(MT9P031_RES_4 < mt9p031Resolution){
            mt9p031Resolution = MT9P031_RES_1;
        }
    }

    switch(mt9p031Resolution){
        case MT9P031_RES_1:
            mt9p031Width = MT9P031_WIDTH_1;
            mt9p031Height = MT9P031_HEIGHT_1;
        break;
        case MT9P031_RES_2:
            mt9p031Width = MT9P031_WIDTH_2;
            mt9p031Height = MT9P031_HEIGHT_2;
        break;
        case MT9P031_RES_3:
            mt9p031Width = MT9P031_WIDTH_3;
            mt9p031Height = MT9P031_HEIGHT_3;
        break;
        case MT9P031_RES_4:
            mt9p031Width = MT9P031_WIDTH_4;
            mt9p031Height = MT9P031_HEIGHT_4;
        break;
        default:
            mt9p031Width = MT9P031_WIDTH_1;
            mt9p031Height = MT9P031_HEIGHT_1;
        break;
    }

    /* Reset sensor*/
    HAL_GPIO_WritePin(DCMI_RESET_GPIO_Port, DCMI_RESET_Pin, 0);
    HAL_Delay(100);

    /* Size properties*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_ROW_SIZE, mt9p031Height-1);
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_COLUMN_SIZE,
        mt9p031Width-1);
```

```
    /* Enable chip*/
    mt9p031I2CWrite(MT9P031_ADDRESS_HIGH, MT9P031_OUTPUT_CONTROL, 1);
}

void mt9p031ChangeExpTime(uint8_t expTime){
    if(0 == expTime){
        mt9p031Exposure++;
        if(MT9P031_EXP_4 < mt9p031Exposure){
            mt9p031Exposure = MT9P031_EXP_1;
        }
    }

    switch(mt9p031Exposure){
        case MT9P031_EXP_1:
            mt9p031ExpTime = MT9P031_EXP_TIME_1_125;
        break;
        case MT9P031_EXP_2:
            mt9p031ExpTime = MT9P031_EXP_TIME_1_60;
        break;
        case MT9P031_EXP_3:
            mt9p031ExpTime = MT9P031_EXP_TIME_4;
        break;
        case MT9P031_EXP_4:
            mt9p031ExpTime = MT9P031_EXP_TIME_15;
        break;
        default:
            mt9p031ExpTime = MT9P031_EXP_TIME_1_125;
        break;
    }
}

void mt9p031Photo(uint32_t dst){
    /* Start DMA*/
    HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, dst,
        mt9p031Height * mt9p031Width / 2);

    /* Exposure Time*/
    HAL_GPIO_WritePin(DCMI_TRIGGER_GPIO_Port, DCMI_TRIGGER_Pin, 0);
```

```
    HAL_Delay(mt9p031ExpTime);
    HAL_GPIO_WritePin(DCMI_TRIGGER_GPIO_Port, DCMI_TRIGGER_Pin, 1);

    /* Wait until frame complete*/
    while(mt9p031FrameComplete == 0);

    /* Reset Frame Complete*/
    mt9p031FrameComplete = 0;

    /* Stop DMA*/
    HAL_DCMI_Stop(&hdcmi);
}

static void mt9p031I2CWrite(uint8_t address, uint16_t reg, uint16_t data){
    data = __REV16(data);
    HAL_I2C_Mem_Write(&hi2c1, address, reg, I2C_MEMADD_SIZE_8BIT,
        (uint8_t *) &data, 2, 1000);
}

static void mt0p031I2CRead(uint8_t address, uint16_t reg, uint16_t* data){
    HAL_I2C_Mem_Read(&hi2c1, address, reg, I2C_MEMADD_SIZE_8BIT,
        (uint8_t *) data, 2, 1000);
    *data = __REV16(*data);
}

void HAL_DCMI_FrameEventCallback(DCMI_HandleTypeDef *hdcmi){
    mt9p031FrameComplete = 1;
}
```

Fichero buttonsLeds.h

```
/*******************************************************************************
*
* buttonsLeds.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
```

```
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Buttons and LEDs functions. Requires STM32 Hal drivers.
*
******************************************************************************/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef BUTTONSLEDS_H
#define BUTTONSLEDS_H

/* Includes ------------------------------------------------------------------*/
/* System includes*/
#include <stdio.h>
/* Project includes*/
#include "jpeg.h"

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Type definitions ----------------------------------------------------------*/
typedef enum{
       ERROR_CODIFICATION = 1,
       ERROR_MEMORY = 2,
       ERROR_OTHER = 3
} errorDcmi_t;

/* Variable declarations -----------------------------------------------------*/
errorDcmi_t errorDcmi;

/* Function declarations -----------------------------------------------------*/
void checkButton1();
/* Button 2 used for change resolution */
void checkButton2();
/* Button 3 used for change exposure time */
void checkButton3();
```

```
/* Error Led used in errorHandler */
void errorLed();
/* LED rutine*/
void led();
/* Take a photo*/
void takePhoto();

/* Function definitions -----------------------------------------------*/


#endif /* BUTTONSLEDS_H */
```

Fichero buttonsLeds.c

```
/*******************************************************************************
*
* buttonsLeds.c
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Buttons and LEDs functions. Requires STM32 Hal drivers.
*
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
/* Own include*/
#include "buttonsLeds.h"
/* Project includes*/
#include "jpeg_utils.h"
#include "filesystem.h"
#include "mt9p031.h"
#include "demosaicing.h"
#include "encode.h"
```

```
/* Defines -----------------------------------------------------------------*/
#define TIME_DEBOUNCE    150
#define TIME_LED_ON_FAST 500
#define TIME_LED_ON_SLOW 1000
#define TIME_LED_OFF     1000

/* Macro defines -----------------------------------------------------------*/


/* Variable typedef --------------------------------------------------------*/


/* Variable declarations ---------------------------------------------------*/
uint32_t photoPtr = 0xC0000000;
uint32_t rgbPtr   = 0xC099C700;

/* Variable definitions ----------------------------------------------------*/
errorDcmi_t errorDcmi = ERROR_OTHER;

/* Static function declarations --------------------------------------------*/


/* Function definitions ----------------------------------------------------*/
/* Button 1 used for take a photo */
void checkButton1(){
    if(HAL_GPIO_ReadPin(BT1_GPIO_Port, BT1_Pin)){
        HAL_Delay(TIME_DEBOUNCE);
        if(HAL_GPIO_ReadPin(BT1_GPIO_Port, BT1_Pin)){
            takePhoto();
        }
    }
}

/* Button 2 used for change resolution */
void checkButton2(){
    if(HAL_GPIO_ReadPin(BT2_GPIO_Port, BT2_Pin)){
        HAL_Delay(TIME_DEBOUNCE);
        if(HAL_GPIO_ReadPin(BT2_GPIO_Port, BT2_Pin)){
            mt9p031ChangeResolution(0);
```

```
        }
    }
}

/* Button 3 used for change exposure time */
void checkButton3(){
    if(HAL_GPIO_ReadPin(BT3_GPIO_Port, BT3_Pin)){
        HAL_Delay(TIME_DEBOUNCE);
        if(HAL_GPIO_ReadPin(BT3_GPIO_Port, BT3_Pin)){
            mt9p031ChangeExpTime(0);
        }
    }
}

/* Error Led used in errorHandler */
void errorLed(){
    uint8_t blinkNbr = 1;
    uint8_t ledStatus = 0;
    /* Other LEDs off*/
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    while(1){
        /* LED OFF*/
        if(0 == ledStatus){
            HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
            HAL_Delay(TIME_LED_OFF);
            ledStatus = 1;
        /* LED ON*/
        } else{
            if(blinkNbr <= errorDcmi){
                HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 1);
                HAL_Delay(TIME_LED_ON_FAST);
                HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
                HAL_Delay(TIME_LED_ON_SLOW - TIME_LED_ON_FAST);
            } else{
                HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 1);
                HAL_Delay(TIME_LED_ON_SLOW);
            }
```

```
            ledStatus = 0;
            blinkNbr++;
            if(blinkNbr > 4){
                blinkNbr = 1;
            }
        }
    }
}

/* LED rutine*/
void led(){
    static uint8_t blinkNbr = 1;
    static uint8_t ledStatus = 0;
    static uint32_t timeLedLast = 0;
    uint32_t timeLed = HAL_GetTick();

    /* LED OFF*/
    if(0 == ledStatus){
        /*LED 2 y 3*/
        HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
        HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
        /* Time check*/
        if(timeLed > timeLedLast + TIME_LED_OFF){
            timeLedLast = timeLed;
            ledStatus = 1;
        }
    /* LED ON*/
    } else{
        /* LED 2*/
        if(blinkNbr <= mt9p031Resolution){
            if(timeLed < timeLedLast + TIME_LED_ON_FAST){
                HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
            } else{
                HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
            }
        } else{
            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 1);
        }
        /* LED 3*/
```

```
        if(blinkNbr <= mt9p031Exposure){
            if(timeLed < timeLedLast + TIME_LED_ON_FAST){
                HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
            } else{
                HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
            }
        } else{
            HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 1);
        }
        /* Time check*/
        if(timeLed > timeLedLast + TIME_LED_ON_SLOW){
            ledStatus = 0;
            blinkNbr++;
            if(blinkNbr > 4){
                blinkNbr = 1;
            }
            timeLedLast = timeLed;
        }
    }
}

/* Take a photo*/
void takePhoto(){
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 1);
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, 0);
    HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, 0);
    HAL_GPIO_WritePin(LED4_GPIO_Port, LED4_Pin, 0);
    mt9p031Photo(photoPtr);
    demosaicingRGB888((uint16_t *) photoPtr, (uint8_t *) rgbPtr,
        mt9p031Width, mt9p031Height, BAYER_FILTER_RGGB);
    jpegEncode((uint8_t *) rgbPtr, mt9p031Width, mt9p031Height);
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, 0);
}
```

Fichero demosaicing.h

```
/*****************************************************************************
*
```

```
* demosaicing.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Demosaicing functions.
*
*********************************************************************************/

/* Define to prevent recursive inclusion -----------------------------------*/
#ifndef DEMOSAICING_H_
#define DEMOSAICING_H_

/* Includes ----------------------------------------------------------------*/
/* System includes*/
#include <stdio.h>

/* Defines -----------------------------------------------------------------*/


/* Macro defines -----------------------------------------------------------*/


/* Type definitions --------------------------------------------------------*/
typedef enum{
       BAYER_FILTER_RGGB,
       BAYER_FILTER_BGGR,
       BAYER_FILTER_GBRG,
       BAYER_FILTER_GRBG
} sensor_pattern;

/* Variable declarations ---------------------------------------------------*/


/* Function declarations ---------------------------------------------------*/
void demosaicingRGB888(uint16_t *buf, uint8_t *newbuf, uint16_t width,
```

```
    uint16_t height, sensor_pattern pattern);

/* Function definitions ---------------------------------------------------*/


#endif /* DEMOSAICING_H_ */
```

Fichero demosaicing.c

```
/*****************************************************************************
*
* demosaicing.c
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Demosaicing functions.
*
*****************************************************************************/

/* Includes ---------------------------------------------------------------*/
/* Own include*/
#include "demosaicing.h"
/* Project includes*/


/* Defines ----------------------------------------------------------------*/


/* Macro defines ----------------------------------------------------------*/


/* Variable declarations --------------------------------------------------*/
```

```
/* Variable definitions ------------------------------------------------------*/


/* Static function declarations ----------------------------------------------*/


/* Function definitions ------------------------------------------------------*/
/* Width and height are sizes of the original image */
void demosaicingRGB888(uint16_t *buf, uint8_t *newbuf, uint16_t width,
    uint16_t height, sensor_pattern pattern) {
    long i = 0;
    uint16_t tmp = 0;

    for (int row = 0; row < height - 1; row += 2) {
        for (int col = 0; col < width - 1; col += 2) {
            switch (pattern) {
            default:
            case BAYER_FILTER_RGGB:
                newbuf[i + 0] = (uint8_t) ((buf[col + row * width] >> 4)
                    & 0xFF);
                tmp = buf[1 + col + row * width];
                tmp += buf[col + (1 + row) * width];
                newbuf[i + 1] = (uint8_t) ((tmp >> 5) & 0xFF);
                newbuf[i + 2] = (uint8_t) ((buf[1 + col + (1 + row) * width]
                    >> 4) & 0xFF);
            break;
            case BAYER_FILTER_BGGR:
                newbuf[i + 2] = (uint8_t) ((buf[col + row * width] >> 4)
                    & 0xFF);
                tmp = buf[1 + col + row * width];
                tmp += buf[(col + row * width) + width];
                newbuf[i + 1] = (uint8_t) ((tmp >> 5) & 0xFF);
                newbuf[i + 0] = (uint8_t)
                    ((buf[(1 + col + row * width) + width] >> 4) & 0xFF);
            break;
            case BAYER_FILTER_GBRG:
                newbuf[i + 2] = (uint8_t) ((buf[1 + col + row * width] >> 4)
                    & 0xFF);
                newbuf[i + 0] = (uint8_t) ((buf[col + row * width] >> 4)
```

```
                    & 0xFF);
                tmp = buf[col + row * width];
                tmp += buf[(1 + col + row * width) + width];
                newbuf[i + 1] = (uint8_t) ((tmp >> 5) & 0xFF);
            break;
            case BAYER_FILTER_GRBG:
                newbuf[i + 0] = (uint8_t) ((buf[1 + col + row * width] >> 4)
                    & 0xFF);
                newbuf[i + 2] = (uint8_t) ((buf[col + row * width] >> 4)
                    & 0xFF);
                tmp = buf[col + row * width];
                tmp += buf[(1 + col + row * width) + width];
                newbuf[i + 1] = (uint8_t) ((tmp >> 5) & 0xFF);
            break;
            }
            i += 3;
        }
    }
}
```


Fichero encode.h

```
/*******************************************************************************
*
* encode.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: JPEG Encode functions. Requires STM32 Hal drivers.
*
*******************************************************************************/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef ENCODE_H
```

```
#define ENCODE_H

/* Includes ------------------------------------------------------------------*/
/* System includes*/
#include <stdio.h>
/* Project includes*/
#include "jpeg.h"

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Type definitions ----------------------------------------------------------*/


/* Variable declarations -----------------------------------------------------*/


/* Function declarations -----------------------------------------------------*/
void jpegEncode(uint8_t * rgbData, uint16_t width, uint16_t height);

/* Function definitions ------------------------------------------------------*/


#endif /* ENCODE_H */
```

Fichero encode.c

```
/*******************************************************************************
*
* encode.c
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
```

```
*     Company: UMH
*          Web:
* Description: JPEG Encode functions. Requires STM32 Hal drivers.
*
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
/* Own include*/
#include "encode.h"
/* Project includes*/
#include "jpeg.h"
#include "jpeg_utils.h"
#include "filesystem.h"
#include "buttonsLeds.h"

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Variable typedef ----------------------------------------------------------*/
typedef struct{
    uint8_t *dataBuffer;
    uint32_t dataBufferSize;
}JpegDataBufferTypeDef;

/* Variable declarations -----------------------------------------------------*/
uint8_t * bufferInPtr = (uint8_t *) 0xC0000000;
uint8_t * bufferOutPtr = (uint8_t *) 0xC04C4C00;
uint32_t bufferInMaxSize = 5000000;
uint32_t bufferOutMaxSize = 5000000;
uint32_t dataBufferSize = 0;

JpegDataBufferTypeDef jpegOutBuffer;
JpegDataBufferTypeDef jpegInBuffer;

/* Variable definitions ------------------------------------------------------*/
```

```
/* Static function declarations ---------------------------------------------*/


/* Function definitions -----------------------------------------------------*/
void jpegEncode(uint8_t * rgbData, uint16_t width, uint16_t height){
       uint32_t mcuTotalNbr = 0;
    JPEG_ConfTypeDef jpegConf;
    JPEG_RGBToYCbCr_Convert_Function RGBToYCbCr;

    /* Reset all Global variables */
    dataBufferSize = height * width * 3;
    mcuTotalNbr    = 0;
    jpegInBuffer.dataBuffer = bufferInPtr;
    jpegInBuffer.dataBufferSize = 0;
    jpegOutBuffer.dataBuffer = bufferOutPtr;
    jpegOutBuffer.dataBufferSize = 0;

    /* Encoding parameters */
    jpegConf.ChromaSubsampling = JPEG_420_SUBSAMPLING;
    jpegConf.ColorSpace = JPEG_YCBCR_COLORSPACE;
    jpegConf.ImageQuality = 75;
    jpegConf.ImageWidth = width;
    jpegConf.ImageHeight = height;
    HAL_JPEG_ConfigEncoding(&hjpeg, &jpegConf);

    /* Get MCU Number and RGB Convert Function*/
    JPEG_GetEncodeColorConvertFunc(&jpegConf, &RGBToYCbCr, &mcuTotalNbr);

    /* Conversion from RGB to YCbCr*/
    RGBToYCbCr(rgbData, jpegInBuffer.dataBuffer, 0, dataBufferSize,
        &(jpegInBuffer.dataBufferSize));

    /* Start JPEG encoding with DMA method */
    HAL_JPEG_Encode(&hjpeg, jpegInBuffer.dataBuffer,
        jpegInBuffer.dataBufferSize, jpegOutBuffer.dataBuffer,
        bufferOutMaxSize, 60000);
}
```

```
/**
    * @brief JPEG Get Data callback
    * @param hjpeg: JPEG handle pointer
    * @param NbEncodedData: Number of encoded (consummed) bytes from input
    *                       buffer
    * @retval None
    */
void HAL_JPEG_GetDataCallback(JPEG_HandleTypeDef *hjpeg,
    uint32_t NbEncodedData){
    /* Pause input buffer*/
    HAL_JPEG_Pause(hjpeg, JPEG_PAUSE_RESUME_INPUT);
    /* If encoded data is completed*/
    if(NbEncodedData == jpegInBuffer.dataBufferSize){
        /* Config Input Buffer InDataLength = 0*/
        jpegInBuffer.dataBufferSize = 0;
        HAL_JPEG_ConfigInputBuffer(hjpeg,jpegInBuffer.dataBuffer +
            NbEncodedData, 0);
    /* If encoded data not finish*/
    } else{
        /* New input buffer*/
        HAL_JPEG_ConfigInputBuffer(hjpeg, jpegInBuffer.dataBuffer +
            NbEncodedData, jpegInBuffer.dataBufferSize - NbEncodedData);
        /* Resume input buffer*/
        HAL_JPEG_Resume(hjpeg, JPEG_PAUSE_RESUME_INPUT);
    }
}

/**
    * @brief JPEG Data Ready callback
    * @param hjpeg: JPEG handle pointer
    * @param pDataOut: pointer to the output data buffer
    * @param OutDataLength: length of output buffer in bytes
    * @retval None
    */
void HAL_JPEG_DataReadyCallback(JPEG_HandleTypeDef *hjpeg, uint8_t *pDataOut,
    uint32_t OutDataLength){
    jpegOutBuffer.dataBufferSize += OutDataLength;
    HAL_JPEG_Pause(hjpeg, JPEG_PAUSE_RESUME_OUTPUT);
    HAL_JPEG_ConfigOutputBuffer(hjpeg, jpegOutBuffer.dataBuffer +
```

```
        OutDataLength, bufferOutMaxSize - OutDataLength);
    HAL_JPEG_Resume(hjpeg, JPEG_PAUSE_RESUME_OUTPUT);
}

/**
    * @brief  JPEG Error callback
    * @param hjpeg: JPEG handle pointer
    * @retval None
    */
void HAL_JPEG_ErrorCallback(JPEG_HandleTypeDef *hjpeg){
    errorDcmi = ERROR_CODIFICATION;
    Error_Handler();
}

/*
    * @brief JPEG Decode complete callback
    * @param hjpeg: JPEG handle pointer
    * @retval None
    */
void HAL_JPEG_EncodeCpltCallback(JPEG_HandleTypeDef *hjpeg){
    jpegOutBuffer.dataBufferSize += hjpeg->JpegOutCount;
    saveFile(jpegOutBuffer.dataBuffer, jpegOutBuffer.dataBufferSize);
}
```

Fichero filesystem.h

```
/*******************************************************************************
*
* filesystem.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Functions for handling files on the sd card.
*
```

```
*****************************************************************************/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef FILESYSTEM_H
#define FILESYSTEM_H

/* Includes ------------------------------------------------------------------*/
/* System includes*/
#include <stdio.h>

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Type definitions ----------------------------------------------------------*/


/* Variable declarations -----------------------------------------------------*/


/* Function declarations -----------------------------------------------------*/
void searchFile();
void saveFile(uint8_t * ptr, uint32_t length);
void sendFile();

/* Function definitions ------------------------------------------------------*/


#endif /* FILESYSTEM_H */
```

Fichero filesystem.c

```
/*****************************************************************************
*
* filesystem.c
*
```

```
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Functions for handling files on the sd card.
*
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
/* Own include*/
#include "filesystem.h"
/* Project includes*/
#include "fatfs.h"
#include "usart.h"
#include "buttonsLeds.h"

/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Variable typedef ----------------------------------------------------------*/


/* Variable declarations -----------------------------------------------------*/
uint8_t fileIndex = 0;

/* Variable definitions ------------------------------------------------------*/


/* Static function declarations ----------------------------------------------*/
static void uartFile(uint8_t * ptr, uint32_t length);

/* Function definitions ------------------------------------------------------*/
void searchFile(){
    char str[80] = "";
```

```
    FIL file;
    FRESULT result = 0;
    fileIndex = 0;
    do{
        fileIndex++;
        sprintf(str, "Image%.3d.jpeg", fileIndex);
        result = f_open(&file, str, FA_READ);
        f_close(&file);
    } while(FR_OK == result);
}

void saveFile(uint8_t * ptr, uint32_t length){
    uint32_t byteswriten = 0;
    char str[80] = "";
    FIL file;
    sprintf(str, "Image%.3d.jpeg", fileIndex);
    f_open(&file, str, FA_CREATE_ALWAYS | FA_WRITE);
    f_write(&file, ptr, length, (void *) &byteswriten);
    f_close(&file);
    if(byteswriten != length){
        errorDcmi = ERROR_MEMORY;
        Error_Handler();
    }
}

void sendFile(){
    const uint32_t length = 80;
    uint8_t ptr[80];
    uint32_t bytesRead = 0;
    uint32_t size = 0;
    char str[80] = "";
    FIL file;
    sprintf(str, "Image%.3d.jpeg", fileIndex - 1);
    f_open(&file, str, FA_READ);
    size = f_size(&file);
    uartFile((uint8_t *) &size, 4);
    do{
        f_read(&file, ptr, length, (void *) &bytesRead);
        uartFile(ptr, bytesRead);
```

```
    }while(length != bytesRead);
    f_close(&file);
}

static void uartFile(uint8_t * ptr, uint32_t length){
    HAL_UART_Transmit(&huart2, ptr, length, 100000);
}
```

Fichero serial.h

```
/*******************************************************************************
*
* serial.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Serial functions. Requires STM32 Hal drivers.
*
*******************************************************************************/

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef SERIAL_H
#define SERIAL_H

/* Includes ------------------------------------------------------------------*/


/* Defines -------------------------------------------------------------------*/


/* Macro defines -------------------------------------------------------------*/


/* Type definitions ----------------------------------------------------------*/
```

```
/* Variable declarations -----------------------------------------------*/


/* Function declarations -----------------------------------------------*/
void checkSerial();
void initReception();

/* Function definitions -----------------------------------------------*/


#endif /* SERIAL_H */
```

Fichero serial.c

```
/*******************************************************************************
*
* serial.h
*
*     Version: 0.0.1
*  Created on: 15/07/2020
*      Author: Juan Manuel Ferrández Aznar
*       Email: juanma.ferrandez.aznar@gmail.com
*     Company: UMH
*         Web:
* Description: Serial functions. Requires STM32 Hal drivers.
*
*******************************************************************************/


/* Includes ------------------------------------------------------------------*/
/* Own include*/
#include "serial.h"
/* Project includes*/
#include "usart.h"
#include "filesystem.h"
#include "buttonsLeds.h"
```

```
#include "mt9p031.h"
/* System includes*/
#include <string.h>

/* Defines ------------------------------------------------------------------*/


/* Macro defines ------------------------------------------------------------*/


/* Variable typedef ---------------------------------------------------------*/


/* Variable declarations ----------------------------------------------------*/
uint8_t uartReception = 0;
uint8_t uartRxBuffer[2];
uint8_t uartTxBuffer[2];

/* Variable definitions -----------------------------------------------------*/


/* Static function declarations ---------------------------------------------*/


/* Function definitions -----------------------------------------------------*/
void checkSerial(){
    if(1 == uartReception){
        uartReception = 0;
        /* Resolutions*/
        if(0 == strcmp((char *) uartRxBuffer, "R1")){
            mt9p031ChangeResolution(MT9P031_RES_1);
        } else if(0 == strcmp((char *) uartRxBuffer, "R2")){
            mt9p031ChangeResolution(MT9P031_RES_2);
        } else if(0 == strcmp((char *) uartRxBuffer, "R3")){
            mt9p031ChangeResolution(MT9P031_RES_3);
        } else if(0 == strcmp((char *) uartRxBuffer, "R4")){
            mt9p031ChangeResolution(MT9P031_RES_4);
        /* Exposure time*/
        } else if(0 == strcmp((char *) uartRxBuffer, "T1")){
```

```
            mt9p031ChangeExpTime(MT9P031_EXP_1);
        } else if(0 == strcmp((char *) uartRxBuffer, "T2")){
            mt9p031ChangeExpTime(MT9P031_EXP_2);
        } else if(0 == strcmp((char *) uartRxBuffer, "T3")){
            mt9p031ChangeExpTime(MT9P031_EXP_3);
        } else if(0 == strcmp((char *) uartRxBuffer, "T4")){
            mt9p031ChangeExpTime(MT9P031_EXP_4);
        /* Take a photo*/
        } else if(0 == strcmp((char *) uartRxBuffer, "PH")){
            takePhoto();
        /* Get photo*/
        } else if(0 == strcmp((char *) uartRxBuffer, "GP")){
            sendFile();
        /* Read status*/
        } else if(0 == strcmp((char *) uartRxBuffer, "RS")){
            uartTxBuffer[0] = mt9p031Resolution;
            uartTxBuffer[1] = mt9p031Exposure;
            HAL_UART_Transmit(&huart2, uartTxBuffer, 2, 100000);
        }
        /* Request new data*/
        initReception();
    }
}

void initReception(){
    HAL_UART_Receive_IT(&huart2, uartRxBuffer, 2);
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    if(huart == &huart2){
        uartReception = 1;
    }
}
```

ANEXO D: BIBLIOGRAFÍA

[1] Waltham, N. (2010). CCD and CMOS sensors. En A. P. Martin C. E. Huber, *Observing Photons in Space* (págs. 391-408). Didcot, UK: Springer Science & Business Media.

[2] https://www.youtube.com/watch?v=MytCfECfqWc (Consultado en noviembre de 2019)

[3] https://wikivisually.com/wiki/Color_filter_array (Consultado en noviembre de 2019)

[4] https://possibility.teledyneimaging.com/imaging-from-a-to-d/ (Consultado en noviembre de 2019)

[5] https://en.wikipedia.org/wiki/Active-pixel_sensor (Consultado en noviembre de 2019)

[6] https://es.wikipedia.org/wiki/Sensor_CMOS (Consultado en noviembre de 2019)

[7] https://www.anandtech.com/show/6777/understanding-camera-optics-smartphone-camera-trends/4 (Consultado en noviembre de 2019)

[8] https://en.wikipedia.org/wiki/Image_processor (Consultado en noviembre de 2019)

[9] http://www.ti.com/product/TMS320DM355 (Consultado en noviembre de 2019)

[10] https://www.st.com/content/st_com/en/products/automotive-adas-devices/automotive-image-signal-processors/stv0991.html (Consultado en noviembre de 2019)

[11] https://www.albedomedia.com/tecnologia/procesadores-de-imagen-i/ (Consultado en noviembre de 2019)

[12] https://vhdl.es/fpga/ (Consultado en noviembre de 2019)

[13] Accelerated Image Processing on FPGAs.Bruce A. Draper, J. Ross Beveridge, A.P. Willem Böhm, Charles Ross,Monica Chawathe, Jeffrey Hammes, 2003

[14] https://www.xilinx.com/products/silicon-devices/soc.html (Consultado en noviembre de 2019)

[15] https://www.microsemi.com/product-directory/fpga-soc/1639-soc-fpgasv

[16] https://www.xilinx.com/products/intellectual-property/nav-audio-video-and-image-processing/nav-image-processing.html (Consultado en noviembre de 2019)

[17] https://www.intel.com/content/www/us/en/programmable/products/intellectual-property/ip/dsp/m-alt-vipsuite.html (Consultado en noviembre de 2019)

[18] A. Benedetti and P. Perona, "Real-time 2-DFeature Detection on a ReconfigurableComputer," presented at IEEE Conferenceon Computer Vision and PatternRecognition, Santa Barbara, CA, 1998.

[19] J. Woodfill and B. v. Herzen, "Real-TimeStereo Vision on the PARTS ReconfigurableComputer," presented at IEEE Symposiumon Field-Programmable Custom ComputingMachines, Napa, CA, 1997.

[20] D. Benitez and J. Cabrera, "ReactiveComputer Vision System withReconfigurable Architecture," presented atInternational Conference on Vision Systems,Las Palmas de Gran Canaria, 1999.

[21] R.W.Hartenstein,J.Becker,R.Kress,H.Reinig, and K. Schmidt, "A ReconfigurableMachine for Applications in Image andVideo Compression," presented atConference on Compression Technologiesand Standards for Image and VideoCompression, Amsterdam, 1995.

[22] Horowitz, P. and Hill, W. (2015). *The art of Electronics*. 3rd ed. Cambridge: Cambridge Univ. Press, p.1053.

[23] https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers:ARM-MICROCONTROLLERS https://www.ti.com/microcontrollers/simplelink-mcus/overview.html https://www.cypress.com/products/microcontrollers-mcus (Consultado en noviembre de 2019)

[24] Ian Johnson, Supercharging the Embedded Device: ARM Cortex-M7

[25] https://www.st.com/en/microcontrollers-microprocessors/stm32-arm-cortex-mpus.html (Consultado en noviembre de 2019)

[26] https://www.raspberrypi.org/ (Consultado en noviembre de 2019)

[27] https://beagleboard.org/bone (Consultado en noviembre de 2019)

[28] https://www.onsemi.com/products/sensors/image-sensors-processors/image-sensors#products=fjI1MDMxNjh+dmFsdWV+M341fjUuMX41LjN (Consultado en julio de 2020)

[29] https://en.wikipedia.org/wiki/Camera_Serial_Interface (Consultado en julio de 2020)

[30] https://graphics.stanford.edu/projects/camera-2.0/faq.html (Consultado en julio de 2020)

[31] https://frankencamera.wordpress.com/ (Consultado en julio de 2020)

[32] https://www.digikey.es/product-detail/es/xilinx-inc/XC7Z020-1CLG484C/122-1850-ND/3925759 (Consultado en julio de 2020)

[33] https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productTable (Consultado en julio de 2020)

[34] STMicroelectronics (2017). *AN5020 Rev 1. Application note Digital camera interface (DCMI) for STM32 MCUs*. (págs. 13 y 14)

[35] https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus:IMX-RT-SERIES (Consultado en agosto de 2020)

[36] https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html (Consultado en agosto de 2020)

[37] STMicroelectronics (2019). DS12110 STM32H742xI/G STM32H743xI/G Datasheet. (pág. 1)

[38] STMicroelectronics (2017). *AN5020 Rev 1. Application note Digital camera interface (DCMI) for STM32 MCUs*. (pág. 14)

[39] http://www.differencebetween.net/object/difference-between-sram-and-sdram/ (Consultado en julio de 2020)

[40] https://www.jedec.org/standards-documents/dictionary/terms/pseudostatic-random-access-memory-psram (Consultado en julio de 2020)

[41] STMicroelectronics (2015). *MB1191 Rev B-02 STM32F746 Discovery Interconnexion.* (pág. 7)

[42] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 29)

[43] STMicroelectronics (2019). DS12110 STM32H742xI/G STM32H743xI/G Datasheet. (pág. 107)

[44] Micron (2012). *128mb_x4x8x16_ait-aat_sdram.pdf - Rev. D 6/18 EN*. (pág. 25)

[45] Kingston. *Datasheet 4900180-001.A00 microSDHC memory card.* (pág. 14)

[46] Analog Devices. Datasheet *LTC6655/LTC6655LN. Rev. H*. (pág. 4)

[47] Analog Devices. Datasheet *LT3042.* (pág. 1)

[48] https://www.keil.com/support/man/docs/jlink/jLink_connectors.htm (Consultado en marzo de 2019)

[49] https://www.analog.com/en/products/lt3042.html#product-quality (Consultado en marzo de 2019)

[50] https://www.analog.com/en/products/ltc6655.html#product-quality (Consultado en marzo de 2019)

[51] https://www.st.com/en/microcontrollers-microprocessors/stm32h743-753.html#resource (Consultado en marzo de 2019)

[52] STMicroelectronics (2019). DS12110 STM32H742xI/G STM32H743xI/G Datasheet. (pág. 64 - 103)

[53] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 31)

[54] STMicroelectronics (2019). DS12110 STM32H742xI/G STM32H743xI/G Datasheet. (pág. 139)

[55] https://jlcpcb.com/capabilities/Capabilities (Consultado en abril de 2019)

[56] STMicroelectronics (2019). DS12110 STM32H742xI/G STM32H743xI/G Datasheet. (pág. 139)

[57] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 362)

[58] https://elinux.org/RPi_SD_cards (Consultado en mayo de 2020)

[59] Sandisk (2003). *Product Manual Sandisk Secure Digital Card*. Pag.32

[60] https://community.st.com/s/question/0D50X00009XkY9GSAV/bug-in-hal-sd-driver-code (Consultado en mayo de 2020)

[61] https://community.st.com/s/question/0D50X00009Xkfcq/microsd-card-problem-bug-in-the-filestm32f4xxhalsdc- (Consultado en mayo de 2020)

[62] http://elm-chan.org/fsw/ff/00index_e.html (Consultado en mayo de 2020)

[63] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 782)

[64] Micron (2012). *128mb_x4x8x16_ait-aat_sdram.pdf - Rev. D 6/18 EN*. (pág. 8)

[65] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 26)

[66] *Aptina (2010). MT9VP031 Register Reference – Rev. A 5/11 EN. (pág. 5)*

[67] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 27)

[68] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 1168)

[69] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 1163)

[70] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 2000)

[71] https://github.com/torvalds/linux/blob/master/drivers/media/i2c/mt9p031.c (Consultado en mayo de 2020)

[72] https://opencv.org/about/ (Consultado en mayo de 2020)

[73] Aptina (2010). *MT9VP031 Register Reference – Rev. A 5/11 EN*. (pág. 10)

[74] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 23)

[75] Onsemi (January, 2017). *MTP031/D – Rev. 10*. (pág. 14)

[76] https://en.wikipedia.org/wiki/Color_space (Consultado en julio de 2020)

[77] https://es.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html (Consultado en julio de 2020)

[78] https://es.wikipedia.org/wiki/YCbCr (Consultado en julio de 2020)

[79] Hamilton, E. (1 de Septiembre de 1992). JPEG File Interchange Format Version 1.02. 2. Milpitas, CA 95035, USA: C-Cube Microsystems.

[80] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (pág. 1215)

[81] Hamilton, E. (1 de Septiembre de 1992). JPEG File Interchange Format Version 1.02. 4. Milpitas, CA 95035, USA: C-Cube Microsystems.

[82] STMicroelectronics (2018). *AN4496*. (pág. 29)

[83] https://gitlab.com/free-astro/siril (Consultado en julio de 2020)

[84] STMicroelectronics (2018). *RM0433 Rev 5. Reference manual STM32H743/753 and STM32H750 advanced ARM®-based 32-bit MCUs*. (págs. 858 y 859)

[85] STMicroelectronics (2018). *AN4496*.

[F1]. Waltham, N. (2010). CCD and CMOS sensors. En A. P. Martin C. E. Huber, *Observing Photons in Space* (págs. 391-408). Didcot, UK: Springer Science & Business Media.

[F2] https://www.raspberrypi.org/ (Consultado en noviembre de 2019)

[F3] https://wikivisually.com/wiki/Color_filter_array (Consultado en noviembre de 2019)