UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería Electrónica

Diseño de bridge USB3.0-MIPI

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Samper Rodríguez, Arturo

Tutor/a: Toledo Alarcón, José Francisco

Cotutor/a externo: PERINO VICENTINI, IVAN VIRGILIO

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

DISEÑO DE BRIDGE USB3.0-MIPI

Autor: Arturo Samper Rodríguez

Tutor: José Francisco Toledo Alarcón

Tutor de Empresa: Ivan Virgilio Perino

Trabajo Fin de Máster presentado en el Departamento de Ingeniería Electrónica de la Universitat Politècnica de València para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2021-22

Valencia, Julio de 2022

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

CIS CMOS Image Sensor.

DPCM Differential Pulse Code Modulation.

GPIO General Purpose Input/Output.

HAL Hardware Abstraction Layer.

LDO Low Dropout.

MCU Microcontroller Unit.

MIPI Mobile Industry Processor Interface.

PCB Printed Circuit Board.

PLL Phase Locked Loop.

Parte I

Memoria

Capítulo 1

Introducción

1.1. Motivo del Proyecto

El auge de la tecnología de reconstrucción 3D es un hecho, siendo cada vez más segmentos del mercado los que demanda esta técnica. Como por ejemplo:robótica industrial, electrónica de consumo, atención médica, automóvil o aeroespacial y defensa, entre otras.

Es por ello que actualmente en la empresa **photonicSENS©** se trabaja para cubrir esa necesidad de mercado y poder suministrar dispositivos de imagen capaces de obtener tanto el mapa de profundidad, es decir, la imagen 3D, como la imagen 2D. Utilizando para ello un *camera module* y una tecnología propia.

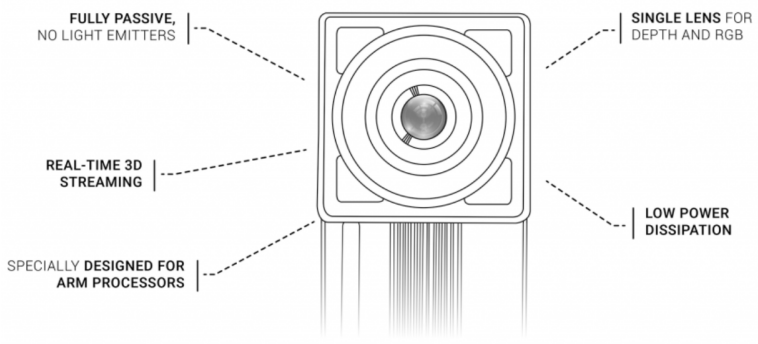


Figura 1.1: apiCAM©[1]

Es cierto que la ventaja competitiva de la compañía reside tanto en los algoritmos como en la capa software que logran implementarlos de forma eficiente en diferentes arquitecturas. Aunque para que esto sea posible, será necesario poder transferir la información de los píxeles que conforman los frames desde el CIS hasta el dispositivo *host*. Esto se realizaba a través de una placa de evaluación propiedad de ON-Semiconductor©, implementando de un driver custom y de una API ad-hoc.

Pero el gran tamaño del hardware sumado a la opacidad del driver propio, supone perder versatilidad en cuanto a diseño. Y sobre todo, permanecer atados a un hardware de terceros para poder mostrar la tecnología en los diferentes sistemas operativos con mayor índice de uso.

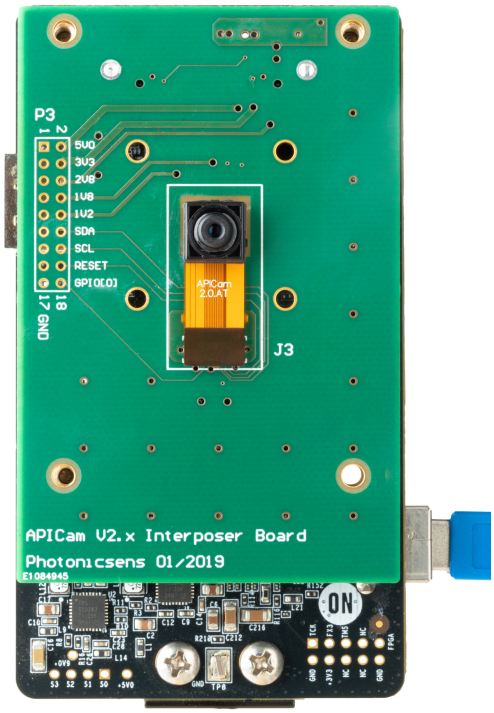


Figura 1.2: Placa de Evaluación [2]

El presente proyecto se basa en diseñar un dispositivo electrónico propio, denominado **Bridge CX3**, capaz de dotar de conectividad USB al CIS actual. Donde La finalidad sea recibir el streaming de datos de imagen provenientes del sensor a través del bus MIPI CSI-2 y pasarlos a USB3.0, siguiendo un formato de dispositivo USB estándar basado en el driver genérico UVC.

Y es que la introducción y adopción de un estándar de mercado, como es USB, hace que cada vez sea más común dotar este tipo de conectividad a los dispositivos electrónicos, quedando demostrado que aumenta las posibilidades de éxito del producto, convirtiéndolo en un dispositivo más versátil.

A lo largo del desarrollo del proyecto se describirán los aspectos fundamentales de los dos grandes bloques de diseño: el desarrollo del hardware y el desarrollo del firmware, continuando con la implementación de una aplicación software, basada en la librería OpenCV, necesaria para verificar el correcto funcionamiento del dispositivo. Finalmente se abordará la puesta a punto y optimización del conjunto CIS-MCU(CX3), con el fin de lograr obtener el mayor throughput posible.

Todo ello permitirá obtener un producto comercial capaz de competir en un mercado que ya es una realidad.

1.2. Objetivos

Antes de entrar en materia, se plantea un pequeño resumen de las tareas que serán realizadas en los diferentes bloques de diseño. Cabe destacar que para la consecución de todos los objetivos, será imprescindible las **directrices del tutor de empresa**.

1.2.1. Hardware

Planteamiento de todos los componentes que requiere el PCB, generar el proyecto en Altium y obtener las librerías de todos los componentes. Determinar el número de capas que tendrá el PCB así como su *stack-up* y definir las directivas de diseño, en función de las reglas marcadas por el fabricante elegido. Por último, se pasará a verificar el dispositivo, generar los ficheros *gerbers* y lanzar tanto la producción como su ensamblado.

Será necesario tener acceso a toda la documentación técnica de los diferentes circuitos integrados, al igual que ser miembro del foro técnico de la empresa ***Cypress***, donde se proporciona infinidad de soluciones a problemas hardware y firmware de sus integrados.

1.2.2. Firmware

Familiarizarse con el IDE y ser capaz de compilar y correr los proyectos de ejemplo en las diferentes placas de evaluación que embeben la MCU. Alcanzar un conocimiento profundo sobre los registros del CIS de apiCAM e inicializarlos a través de la MCU. Estudiar el funcionamiento del driver UVC, plantear la arquitectura de la API que permita un control eficiente del CIS y habilitar diferentes perfiles de visualización.

Para la consecución de este bloque, en primer lugar, será necesaria la consultoría de una empresa externa que asesore sobre la integración del CIS para lograrlo conjuntamente, permitiendo acortar plazos. Posteriormente, el planteamiento de la arquitectura de la API, se diseñará en consenso junto con el departamento de software de **photonicSENS**, ya que dicha API nace con la intención de facilitar el control de la cámara al SDK que corre en el *host*, y deberá amoldarse a sus necesidades.

1.2.3. Software

Crear un proyecto en Microsoft Visual Studio basado en la utilización de la librería **OpenCV**. Ser capaz de comunicarse con la cámara a través del hardware desarrollado y verificar todos los puntos contemplados en la API. La concepción del algoritmo se diseñará en consenso junto al tutor en la empresa.

1.2.4. Producto final

El producto final será la mezcla del dispositivo electrónico desarrollado en este proyecto y del chasis mecánico diseñado íntegramente por el área de desarrollo mecánico de **photonicSENS**.

1.3. Herramientas utilizadas

Para lograr la ejecución del propósito marcado en el proyecto, ha sido necesario hacer uso de las siguientes herramientas software:

1.3.1. Altium Designer©

Herramienta utilizada para el desarrollo del hardware. Tanto en la concepción y diseño del esquemático e integración de componentes, como para la posterior fase de rutado del PCB. Cabe destacar que en esta última fase, será muy importante ajustar bien las reglas de diseño en lo referente a tamaño de pistas, vias, pares diferenciales, etc, adecuándolas al fabricante con el que se materialice el dispositivo.

1.3.2. Hyperlynx©

Herramienta utilizada para plantear el estudio y verificación tanto de **integridad de potencia** de los planos de alimentación existentes en el PCB, así como de **integridad de señal** en aquellas señales críticas.

1.3.3. EZ-USB Suite©

Entorno de desarrollo integrado proporcionado por Cypress para el **diseño del firmware que correrá en la MCU del dispositivo**. El entorno está basado en el **standard Eclipse Kepler IDE for C/C++ Developers**, pero además cuenta con una serie de customizaciones ad-hoc para las diferentes dispositivos de la familia Cypress, contando también con el **GNU ARM tool-chain** para los procesos de compilación y debug.

1.3.4. Visual Studio Code

Editor de código fuente desarrollado por Microsoft© que permite la instalación de multitud de *plugins*, entre ellos **Python extension for Visual Studio Code**, convirtiéndolo en un IDE completo para Python. A lo largo del desarrollo del proyecto será necesario hacer uso de un lenguaje de programación de *scripting*, que facilite la obtención de resultados en diversas tareas, como por ejemplo: Caracterización de funciones matemáticas, estimaciones térmicas de componentes, o cálculos de circuitos R-C.

1.3.5. Visual Studio

Entorno de desarrollo integrado desarrollado por Microsoft©, compatible con el lenguaje C++,que permitirá generar proyectos basado en la librería **OpenCV**, necesaria para implementar un entorno de testing de la estructura del firmware. Este programa se ejecutará en el dispositivo *host* sobre el que se conecte el dispositivo.

1.3.6. Git

Software de control de versiones, necesario para llevar un seguimiento completo de los diferentes subproyectos de código que engloba el trabajo actual. Es decir, el proyecto del firmware,el proyecto de la aplicación de testing, así como los scripts de python. Se trabajará con el cliente **GitLab**, y con **TortoiseGit** como cliente Git GUI,que proporciona un entorno gráfico de manejo en Windows.

Capítulo 2

Diseño del PCB

Durante el presente capítulo, se explicará el proceso de desarrollo que se ha llevado a cabo para diseñar el PCB **Bridge CX3** y que definirá el comportamiento hardware de éste. A modo introductorio se plasmará, en primer lugar, un diagrama de bloques del conjunto para posteriormente explicar, con un mayor grado de detalle, los diferentes bloques que lo componen.

2.1. Block Diagram

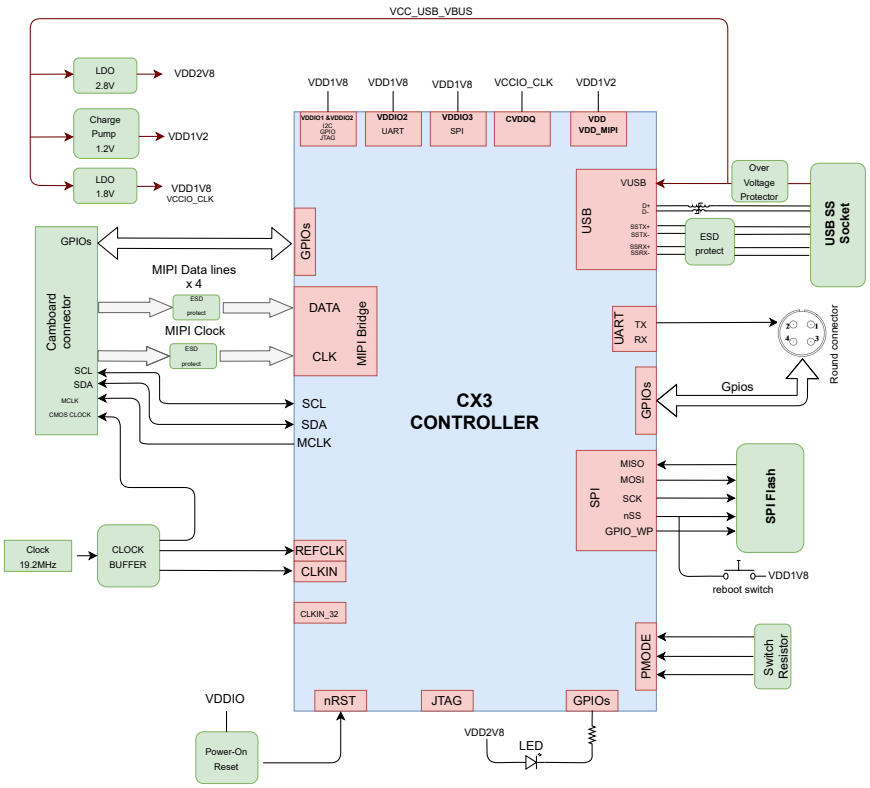


Figura 2.1: Block Diagram Bridge CX3

En el diagrama de bloques resalta como actor principal el integrado **CX3**, que aparece coloreado en **azul**. Los bloques de color **rojo**, no son más que referencias a pines hardware del propio CX3, mientras que en color **verde** se representan el resto de **periféricos**, encargados de hacer trabajar al CX3 en óptimas condiciones y de adaptar el comportamiento del PCB al producto final. Donde la posición del conector USB o el botón de reset de la memoria flash, dependerán de una gran sinergia entre el diseño electrónico y el diseño mecánico.

Para poder implementarlo, ha sido necesario tener acceso a toda la documentación oficial del fabricante del Integrado CX3, ***Datasheet**[3], **CX3 Technical Reference Manual**[4], etc*. Sin esta documentación habría sido una tarea casi imposible. Esto mismo pasará con el resto de integrados, por lo que, una de las premisas más importantes en el diseño hardware es **saber interpretar bien toda la información proporcionada por las hojas de datos de los fabricantes**.Para mayor entendimiento del desarrollo adoptado en las diferentes etapas,se recomienda ***Ir al apéndice A***.

2.2. Conector USB 3.0

Es el primer elemento a plantear en el diseño, siendo la **entrada de alimentación del PCB, establecida en 5V** y la referencia al **plano de masa**.

El conector utilizado se corresponde con un **USB-micro B**. El cual posee los mismo pines del conector habitual USB2.0, es decir, **VBUS**,**D+**,**D-** y **GND**. Y además cuenta con dos pares diferenciales adicionales: **SSTX-**,**SSTX+**,**SSRX-**,**SSRX+**, permitiendo transferir datos en modo *SuperSpeed(SS)*.

El estándar USB contempla una masa adicional destinada al bloque de señal, denominada **GND_DRAIN**. Mientras destina la propia masa **GND** para alimentación. Pero en el presente diseño se unirán y se trabajará sobre un único plano de masa.

Con respecto a las medidas de protección que implementan los diferentes pares diferenciales, se aprecian las siguientes:

- **Par diferencial D** : filtro de modo común, implementado por una bobina de choque y diodos TVS de protección ESD.
- **Par diferencial SSRX,SSTX** : diodos TVS de protección ESD.

El conector cuenta con un pin denominado **SHELL** que puede llevar a confusión teniendo en cuenta que ya posee el pin de **GND**, pero que resultará de ayuda para entender la diferencia entre masa y tierra. **GND** es el potencial eléctrico de referencia del diseño, ajustado en $0V$, proveniente de la fuente de alimentación externa. Mientras que **SHELL** es la conexión a tierra, de ahí que vaya conectada tanto a la carcasa del conector como al chasis del producto final. Será necesario unir ambas a través de dos componentes pasivos:

- **Capacitor de baja capacidad y alta tensión de trabajo**: encargado de filtrar las interferencias de alta frecuencia que puedan llegar al PCB a través de las carcasas metálicas o a través del cable.
- **Resistencia de elevado valor**: encargada de eliminar la estática que pueda tener la masa del PCB con respecto a tierra.

2.3. Etapa de Alimentación

Como la tensión general de 5V vendrá a través del conector USB, será el propio dispositivo *host* el que haga de fuente de alimentación. A partir de esta, habrá que obtener el resto de tensiones de alimentación requeridas por los diferentes componentes. De entre todos ellos, interesan los dos principales:

COMPONENTE	TENSIONES REQUERIDAS
CX3	5V, 1.8V, 1.2V
CIS	1.2V, 1.8V, 2.8V

Tabla 2.1: Tensiones de alimentación requeridas

Factores como el bajo consumo de los integrados que forman el diseño, sumado a la generación de ruido y distorsión armónica que producen los convertidores conmutados, hacen de los LDO el regulador idóneo para el presente proyecto. Concretamente, se selecciona el modelo ***ADP12x*** *de Analog Devices*, ya que se ajusta a las condiciones de tensión y corriente demandadas.

2.3.1. Estimación Térmica

No obstante, al tratarse de la etapa de alimentación, los LDO serán los integrados que mayor potencia disiparán en forma de calor al PCB, suponiendo un peligro en una placa de tan reducidas dimensiones y una alta integridad de componentes.

Es cierto que se podría hacer un modelado térmico complejo, basado en análisis nodal en todos los puntos del PCB, teniendo en cuenta todas las placas de cobre internas, el impacto térmico de las thermal vías o la disipación de calor del PCB al ambiente a través de la convección. Pero debido el reducido tamaño del PCB, se puede simplificar al comportamiento térmico de un solo nodo.

Esto hace que el estudio se reduzca a determinar cual es la temperatura máxima alcanzable en la capa top, que evite superar la ***Junction Temperature(Tj)*** límite del LDO, dando una aproximación a la hora de detectar los integrados que se encuentran en el margen de su funcionamiento nominal, implicando un acortamiento considerable de su vida útil.

Para analizarlo, se hará uso de la siguiente ecuación, que análogamente a la ley de Ohm, determina que la diferencia de temperatura entre dos nodos, dependerá del cociente entre la resistencia térmica entre ellos y la potencia que circula.

$$\Delta T = R_{termica} \cdot P \tag{2.1}$$

Para el caso del LDO:

- ΔT : será el gradiente entre la temperatura en la unión del silicio y la temperatura de la capa donde se coloca el integrado.
- $R_{termica}$: *Junction-to-board characterization parameter (Ψ_{jb})*, obtenida de la hoja de datos del integrado.

– P : será la potencia disipada en forma de calor por cada integrado. Pudiéndose obtener multiplicando el drop de tensión entre la entrada y la salida, por la corriente demandada para ese dominio de tensión.

Será necesario estimar la demanda de corriente de cada dominio de tensión. Esto se consigue sumando los consumos de corriente, en condiciones nominales de funcionamiento, de todos los integrados que cuelgan de cada alimentación, obteniendo el siguiente consumo:

LDO	Demanda de corriente
1.2V	300 mA
1.8V	100 mA
2.8V	75 mA

Tabla 2.2: Demandas de corriente

Observando la hoja de datos del LDO, la temperatura máxima en la unión se establece en 125^oC. Pero en vista a alargar su vida útil, se estima un valor límite de temperatura máxima en la unión de 100 ^oC. Por consiguiente, se procede a obtener la relación entre la *junction Temperature(Tj)* alcanzada en los tres integrados, ante rango de temperaturas del *top layer* comprendidas entre 40^oC a 120^oC.

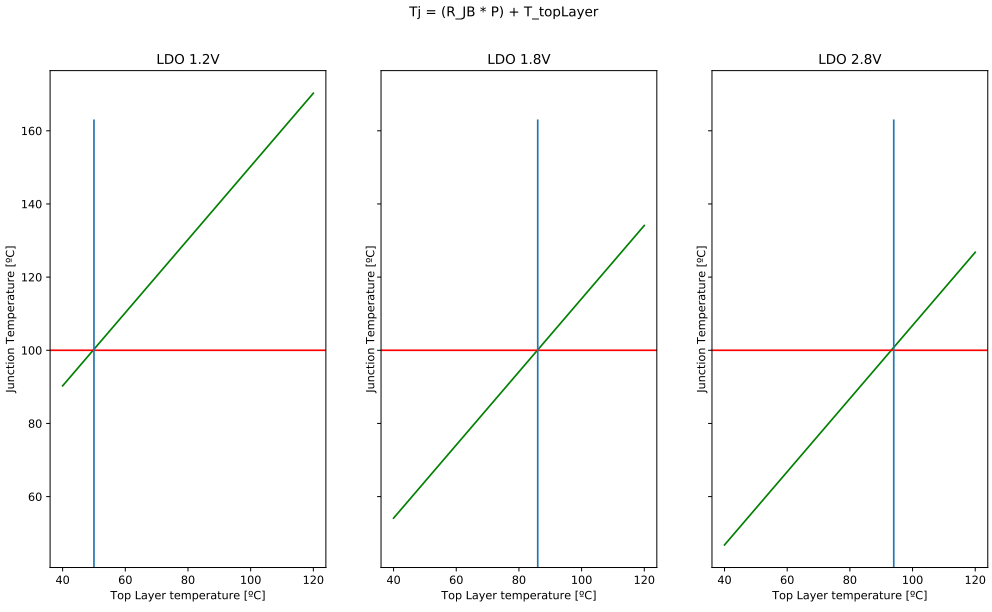


Figura 2.2: Barrido Termico LDOs

```
Potencia consumida por el LDO de 1.2V : 1.14 W
Coeficiente de disipacion por unidad de area : 0.1186 W/mm^2
-------------------------------------------------
Potencia consumida por el LDO de 1.8 : 0.32 W
Coeficiente de disipacion por unidad de area : 0.0333 W/mm^2
-------------------------------------------------
Potencia consumida por el LDO de 2.8 : 0.154 W
Coeficiente de disipacion por unidad de area : 0.016 W/mm^2
-------------------------------------------------


Junction Temperature maxima permitida en los LDO : 100 ºC

Temperatura limite alcanzable por el PCB para el LDO 1.2V : 50 ºC
Temperatura limite alcanzable por el PCB para el LDO 1.8V : 86 ºC
Temperatura limite alcanzable por el PCB para el LDO 2.8V : 94 ºC
```

Es apreciable ver que el cuello de botella en cuanto a diseño térmico lo marca el LDO de $1{,}2V$, ya que posee los dos factores más desfavorables en cuanto a generación de potencia se refiere: mayor drop de tensión y mayor demanda de corriente.

2.3.1.1. Solución adpotada para la tensión de 1.2V

Todo indica que cambiando el regulador lineal de 1.2 V por un dispositivo capaz de realizar la misma tarea, es decir, generar un nivel de tensión estable a partir de una tensión de entrada de diferente nivel de voltaje, se estará reduciendo la disipación de potencia total al PCB, implicando una reducción considerable de la temperatura global de trabajo.

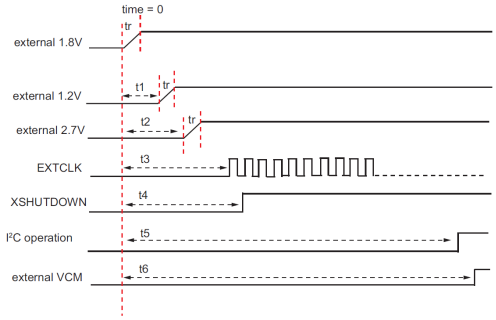
La premisa es clara. Se deberá encontrar un componente que realice la misma función que el actual, pero con un mayor rendimiento. Por lo tanto, la principal opción será encontrar un **DC/DC buck converter**. Aunque el elevado número de componentes externos que requiere, especialmente el inductor, hacen de este tipo de dispositivos una mala elección. Es por ello que se decide introducir un **charge pump step-down DC/DC converter**, concretamente el modelo **LTC3251-1.2** de *Analog Devices*.

Estos charge pump son un tipo de convertidores DC/DC que utilizan capacitores, y no inductores, para el almacenamiento de la carga energética. Se consideran dispositivos que también poseen altos rendimientos y sobre todo presentan tipologías de circuitos más simples que los convertidores buck.

2.3.2. Power-up sequence

Como sucede en multitud de componentes electrónicos, el CIS necesita cumplir una secuencia de establecimiento de sus tensiones durante el proceso de arranque. Es decir, no pueden llegar en cualquier momento por que podría desencadenar multitud de errores funcionales a nivel de silicio.

Para cumplir la secuencia de arranque será necesario seguir lo establecido en la siguiente figura, en donde **primero tiene que establecerse la tensión de 1.8V. A continuación la tensión de 1.2V, y por último la tensión de 2.8V**.



Por un lado, el **charge-pump** que regula la salida de 1.2V posee un tiempo fijo de estabilización de su tensión de 1 ms. Por otro lado, los **LDO** cuentan con un *soft-start* interno cuya constante de tiempo es de 350 us, deduciendo que no hará falta introducir ningún control en estos dos integrados para cumplir la condición de arranque entre las tensiones de 1.8V y 1.2V.

En cambio, hay que lograr que la tensión de 2.8V se establezca 1ms posterior a la estabilización de 1.2V, es decir, a los 2 ms. Para lograr este requisito, los LDO cuentan con una entrada de habilitación, encargada de activar la tensión de salida una vez se supere un *hreshold* de tensión en dicha entrada.

La solución adoptada se basa en colocar un **filtro R-C**, que consiga alcanzar la tensión de *threshold* pasados 2 ms. De esta forma se estará cumpliendo la temporización de las diferentes tensiones de alimentación, y por consiguiente, se efectuará la secuencia de arranque del CIS.

Para calcular los valores del circuito R-C será necesario evaluar la respuesta temporal de la salida, ante una tensión de entrada escalón de $5V$, contando también con las tolerancias de los componentes pasivos, logrando estimar la variabilidad temporal que pueda tener la activación del *threshold*.

$$V_{out} = V_{in} \cdot (1 - e^{-\frac{t}{\tau}}) \qquad siendo \;\; \tau = R \cdot C \tag{2.2}$$

Capacitor	0,1 $\mu F \pm 5\,\%$
Resistencia	100 $k\Omega \pm 1\,\%$
Threshold	0.98 V
iteraciones	1000

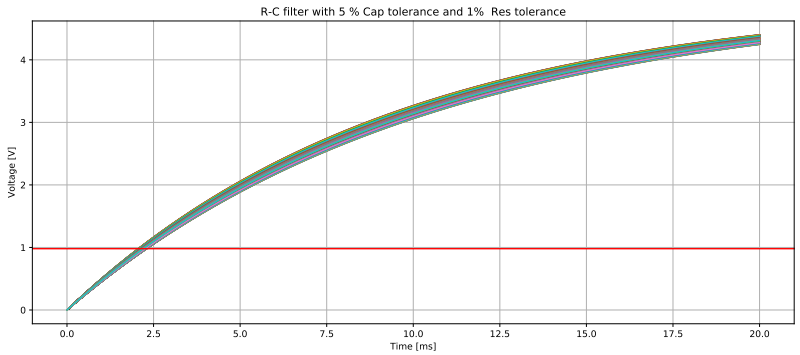
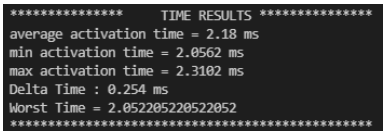


Figura 2.3: filtro R-C



Colocando el filtro en el LDO de 2.8V, se terminará de cumplir la secuencia de arranque del CIS, evitando la aparición de posibles errores de comportamiento hardware derivados de una mala inicialización.

2.4. Clock del sistema

El clock principal del sistema está compuesto por un **oscilador de cuarzo XTAL de 19.2MHz**, alimentado a 1.8V. La salida del XTAL conectará directamente con un ***clock Buffer***, componente de alta importancia que permitirá obtener a su salida un número de señales con muy bajo *skew*, y con flancos mucho más definidos, consiguiendo reducir el jitter del clock de entrada.

Para lograr este comportamiento, el *clock buffer* usa la histéresis con el fin de prevenir el ruido que podría tener la señal de clock original, que por lo general suele generar flancos débiles. Se obtendrán las 3 señales de clock que son necesarias en el sistema:

- **CLK_19.2MHZ_CX3**: Clock del CX3, para propósito general.
- **CLK_19.2MHZ_MIPI**: destinado únicamente al *embedded hard-core* MIPI CSI-2 RX, del CX3. Es decir, para gestionar la recepción de los datos del MIPI.
- **CLK_CMOS**: Clock destinado al CIS.

2.5. Reset del sistema

Se corresponde con el integrado encargado de gestionar la señal de reset del CX3,escogiendo el modelo ***TPS3808G50****, de Texas Instruments*, por su funcionamiento simple y eficiente. La condición para resetear el sistema se basa en la existencia de un drop en la tensión de alimentación general, es decir, el plano de $5V$. El integrado, a través de su salida *open-collector*, reseteará el CX3 cuando en su pin de monitorización **SENSE** se alcance una tensión igual o inferior a $4{,}65V$.

El hecho de habilitar el *footprint* de dos resistencias, a modo de divisor resistivo, hará que sea posible adaptar el nivel de tensión que lee el pin SENSE. Esto habilitará la colocación de otros integrados de la misma familia, que únicamente difieren en el nivel de tensión de activación del pin SENSE.

2.6. Boot Mode

Consta de un conexionado de resistencias en configuración de *pull-up* o *pull-down*, que marcarán los niveles lógicos de tres GPIO del CX3, encargados de configurar el *bootmode* que determina como se cargará la imagen binaria del firmware.

El CX3 da tres opciones posibles: cargar el binario a través de USB desde el ordenador, grabar el binario en una memoria EEPROM-I2C o grabar el binario en una memoria SPI Flash. En el presente diseño se ha optado por embeber la tercera opción, evitando la necesidad de cargar el firmware en cada ciclo de encendido.

BOOT MODE	PMODE0[GPIO]	PMODE1[GPIO]	PMODE2[GPIO]
USB	1	1	sin conectar
I2C	sin conectar	0	sin conectar
SPI	1	sin conectar	0

Tabla 2.3: Configuración Boot Mode

2.7. Conexionado CX3

Se trata del integrado de mayor importancia del PCB, que determina la funcionalidad del mismo. Por consiguiente, habrá que plantear un correcto conexionado tanto de las alimentaciones, como de los diferentes buses de datos. Pero, para lograr un buen funcionamiento del integrado en particular y de todo el PCB en general, se deberán de centrar los esfuerzos en configurar una adecuada red de desacoplo.

2.7.1. Red de desacoplo

Para diseñar la red de desacoplo del PCB fue necesario acceder a la hoja de datos de todos los integrados que forman el diseño.

El motivo de realizar una buena red de desacoplo reside en que, en el mundo real, las pistas y planos que conectan las fuentes de alimentación con los distintos integrados, poseen un cierto valor de resistencia e inductancia. Dicha inductancia aplica una impedancia que es directamente proporcional a la frecuencia. Y como consecuencia, la inductancia impedirá que la fuente de alimentación pueda suministrar los picos de corriente demandados por la conmutación de los transistores internos de cada IC.

El hecho de no colocar una adecuada red de desacoplo implicaría tener un ruido de alta frecuencia en los puntos cercanos a los pines de alimentación de los integrados. Algo totalmente indeseado y que afectaría al correcto funcionamiento del PCB.

2.7.1.1. Impacto del desacoplo

Los condensadores de desacoplo serán fundamentales para suplir dos aspectos de diseño muy importantes:

- Por un lado, lo que se consigue al colocar un condensador de desacoplo lo más cercano posible al pin del integrado, no es otra cosa que tener una segunda fuente de alimentación capaz de suministrar esos picos de corriente de alta frecuencia al IC, eliminando el efecto del retardo en la demanda de corriente producido por la inductancia de las pistas.
- Por otro lado, el condensador de desacoplo se encargará de filtrar el ruido que se produce en los puntos cercanos a los pines de alimentación de los integrados. Normalmente ese ruido vendrá determinado por la frecuencia de conmutación de los ICs.

2.7.1.2. Valor de los condensadores

Tras lo explicado en el punto anterior, se entiende que se deberá colocar un condensador de desacoplo lo más cercano al pin de alimentación de los integrados. Y que además, para que sea efectivo en cuanto a filtrado, dicho condensador muestre su punto de menor impedancia a la frecuencia a la que se produce el ruido.

Estas frecuencias de conmutación no suelen ser datos suministrados por los fabricantes de circuitos integrados, en cambio, si que indicarán el valor recomendado de los condesadores de desacoplo y las características de su encapsulado, ya que a fin de cuentas son los parámetros que definirá su curva de impedancia en función de la frecuencia.

Para entender esto último, será necesario tener en cuenta la curva de impedancia de un modelo de condensador real.

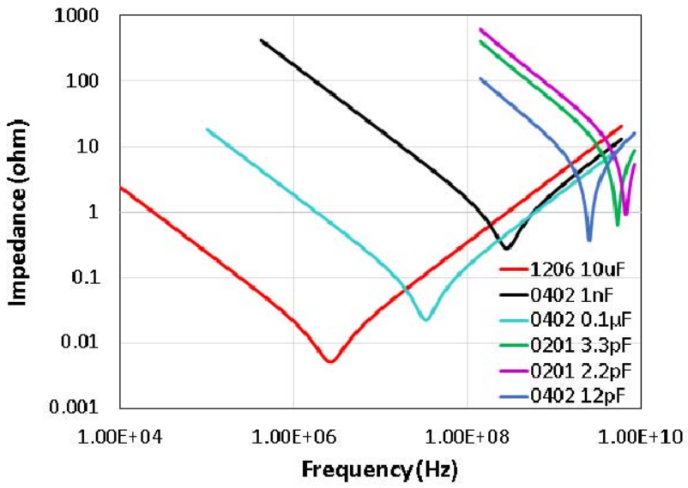


Figura 2.4: Curva Impedancia Condensador Real[5]

De la figura anterior se extrae que el valor y el tamaño del condensador modificarán la frecuencia de resonancia que marca el punto de impedancia mínima. Los condensadores de mayor tamaño

requieren encapsulados más grandes, implicando también el aumento de su inductancia parásita (ESL), lo que conllevará una disminución considerable de su frecuencia de resonancia y por consiguiente, del punto de impedancia mínima.

Es por esto que en el presente diseño se planteará la siguiente distribución:

- **Condensadores de mayor tamaño (10uF-4.7uF):** contando con encapsulados comprendidos entre 0805-0603. Se encontrarán tanto a la entrada de alimentación general para filtrar el posible ruido de la alimentación externa, como a la salida de los LDO, ya que debido a su mayor tamaño implicará tener la capacidad de suministrar mayor corriente. Esto es fundamental en dispositivos como los LDO, ya que poseen un fan-out elevado teniendo que alimentar a muchos integrados.
- **Condensadores de menor tamaño (0.1uF-0.01uF):** contando con encapsulados 0402. Se encontrarán lo más cercano posible a los pines de alimentación de los IC. Al contar con una inductancia parásita pequeña tendrán su punto de impedancia mínima a frecuencias elevadas cercanas a las frecuencias de conmutación de los integrados. Se recomendará colocar los condensadores utilizando la técnica *via on pad* para reducir al máximo la inductancia parásita.

2.8. Momoria SPI Flash

El presente diseño requerirá de memoria adicional para almacenar el binario del firmware. Acudiendo a la documentación del CX3, se decide colocar el modelo de Cypress ***S25FS064S***, que cuenta con 64 Mb alimentada a 1.8V.

En el **apartado 2.6** se configuró el CX3 para que arrancara el firmware desde la presente memoria, evitando tener que ser cargado desde el *host* cada vez que el dispositivo fuera encendido de nuevo. Pero para aquellos momentos en los que haya que actualizar el firmware grabado en la memoria, será necesario diseñar un mecanismo que consiga que el CX3 arranque sin ningún firmware a la espera de ser cargado desde el dispositivo externo. Siendo en ese momento cuando se pueda graba un nuevo firmware en la memoria Flash.

Este mecanismo consistirá en introducir un *switch* que cortocircuitará el pin de *chip-select* de la memoria y lo establecerá a nivel lógico alto. De esta forma, el CX3 será incapaz de activar la memoria, inhabilitando el integrado e impidiendo la transacción del firmware al CX3.

2.9. Conexión del CIS

Para el conexionado del CIS se parte de una premisa de diseño muy importante. El verdadero potencial del producto reside en la obtención del mapa de profundidad de la escena que es capturada por apiCAM©, y no tanto en obtener una calidad extrema de la imagen 2D. Esto deriva en que sea de vital importancia codificar y enviar los datos de los píxeles con los mínimos niveles de ruido electrónico posible.

No es motivo de estudio del presente proyecto, abordar las diferentes fuentes de ruido que afectan a sensores de imagen basados en tecnología CMOS. Pero si cabe destacar que la temperatura es

uno de los principales factores a tener en cuenta. Es decir, que existirá una relación entre el ruido existente en los datos codificados por el sensor y la temperatura de trabajo del mismo.

Por consiguiente, será necesario hacer trabajar al CIS en un rango de temperatura lo más bajo posible que optimice su rendimiento. Esto hace que el conexionado del CIS no sea tan inmediato como el del resto de componentes, ya que, debido a las reducidas dimensiones del PCB y la alta integración de componentes, no será posible embeber el *camera module* sobre el actual PCB Bridge CX3.

2.9.1. Camboard

La solución adoptada ha sido diseñar un PCB adicional, cuya misión sea símplemente conectar **apiCAM©** al **Bridge CX3**, siguiendo el presente diagrama de bloques.***Ir al apéndice C*** para mayor información a cerca del diseño.

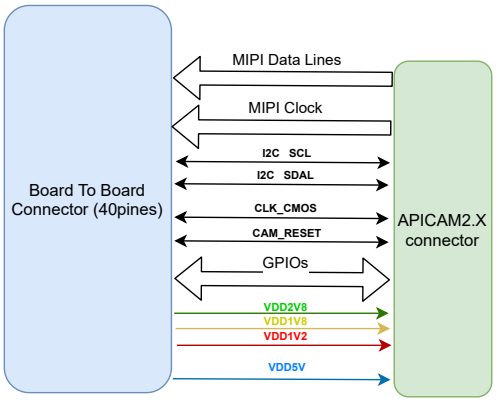
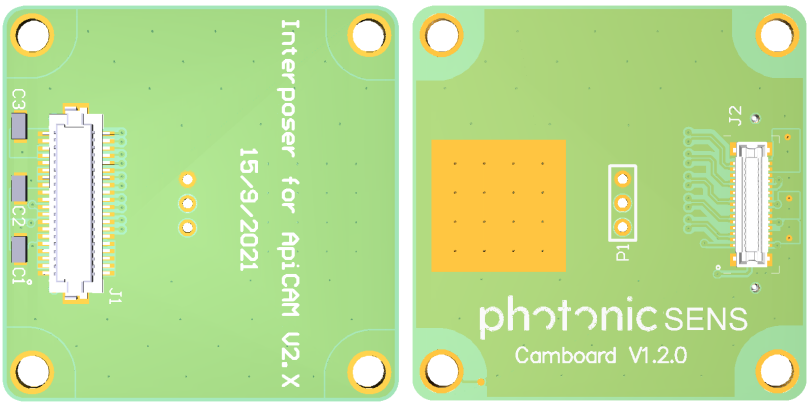


Figura 2.5: Block Diagram Camboard

El PCB deberá tener el mismo factor de forma que el Bridge CX3. Además, al contar con 4 capas y una baja integración de componentes, permitirá introducir planos de masa interconectados a través de vías por todas las capas del PCB, consiguiendo una muy buena disipación térmica del CIS.



2.9.2. Board To Board Connector

Para poder transferir todas las señales necesarias entre el CX3 y el CIS, será necesario añadir un conector con un número elevado de pines. Esto permitirá pasar desde el BridgeCX3, señales adicionales como GPIOs para poder controlar multitud de periféricos que sean añadidos al diseño de la Camboard en futuras etapas.

2.10. Debug Connector

En vistas a poder corregir el firmware, será necesario poder plasmar el valor de las variables almacenadas en memoria a través del puerto serie. Y es que debido al reducido tamaño del PCB será imposible poder introducir un conector JTAG.

A través de este conector de formato circular, será posible establecer el protocolo UART abriendo comunicación con un puerto COM desde el PC.

2.11. Stack-up del PCB

El presente diseño queda definido con la siguiente distribución de capas:

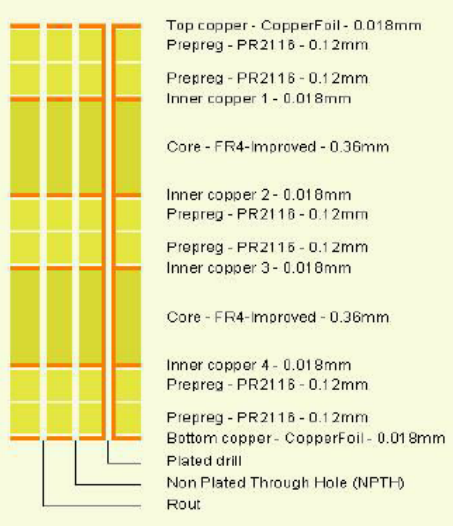


Figura 2.6: Stackup Bridge CX3

Se contará con un PCB de 6 capas, por ello se decide abordar el diseño de forma que las capas de rutado estén lo más próximas posibles a los planos de masa, albergando dos grandes ventajas:

- Crear un apantallamiento entre diferentes pistas de conexión, evitando al máximo de lo posible, fuentes de *crosstalk*.
- Acortar y facilitar los caminos de las corrientes de retorno de las pistas que trabajan a alta velocidad, ya que estas siguen el camino de menor impedancia.

Una de las capas centrales se corresponderá con los diferentes planos de alimentación. El hecho de que se encuentre en la zona central del *stuck-up* conlleva la ventaja de tener a los planos de masa como planos contiguos a los de rutado, aunque la separación existente entre los planos de alimentación y los planos de rutado aumentará considerablemente el número de vias.

2.12. Análisis del diseño

A diferencia de lo ocurrido con el firmware o el software, en el mundo hardware, el hecho de iterar para depurar errores se convierte en un proceso costoso en términos económicos y temporales.

Por ende, será primordial simular y verificar el comportamiento del diseño tanto a nivel de integridad de potencia como ajustar la impedancia de las señales que puedan ser tratadas como líneas de transmisión.

2.12.1. Integridad de Potencia (DC Drop)

El objetivo del análisis de integridad de potencia será asegurar el nivel de voltaje y corriente a todos los integrados que cuelgan de cada línea de alimentación. Para asegurar la integridad de potencia en el PCB será necesario detectar y eliminar los DC drops existentes en cualquiera de los diferentes voltajes que intervienen en el diseño.

Con el presente estudio, será posible detectar las área de los planos de alimentación, en donde la resistencia al paso de la corriente sea mayor, es decir, aquellas zonas con superficies de cobre estrechas.

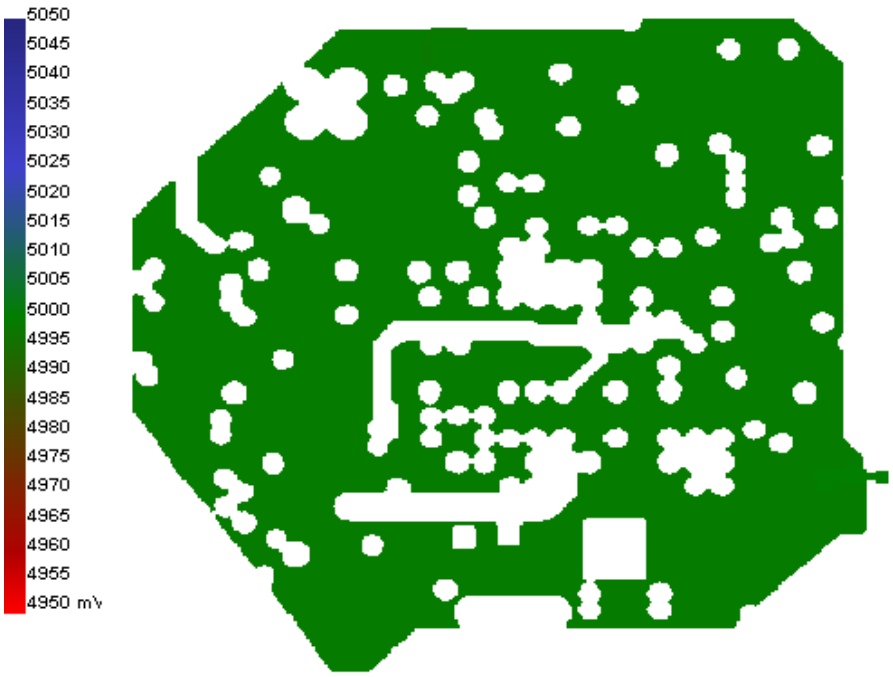


Figura 2.7: Power Plane 5V

Figura 2.8: Power Plane 1.2V

Figura 2.9: Power Plane 1.8V

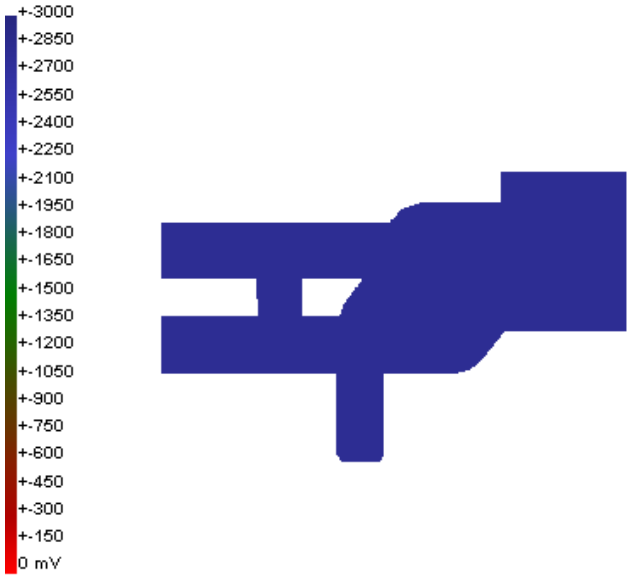


Figura 2.10: Power Plane 2.8V

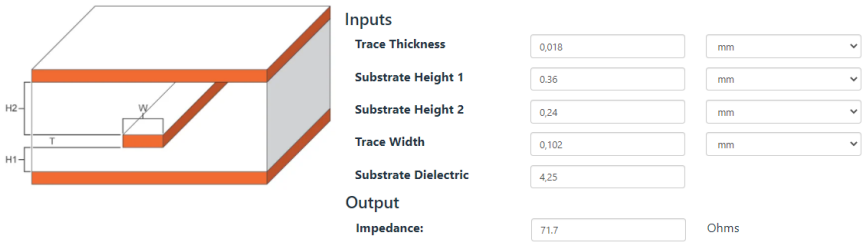
Los 4 planos de alimentación muestran un drop insignificante a lo largo de toda su área. Lo que asegura a los periféricos operar en sus adecuados rangos de tensión de alimentación.

2.12.2. Ajuste de Impedancia en líneas críticas

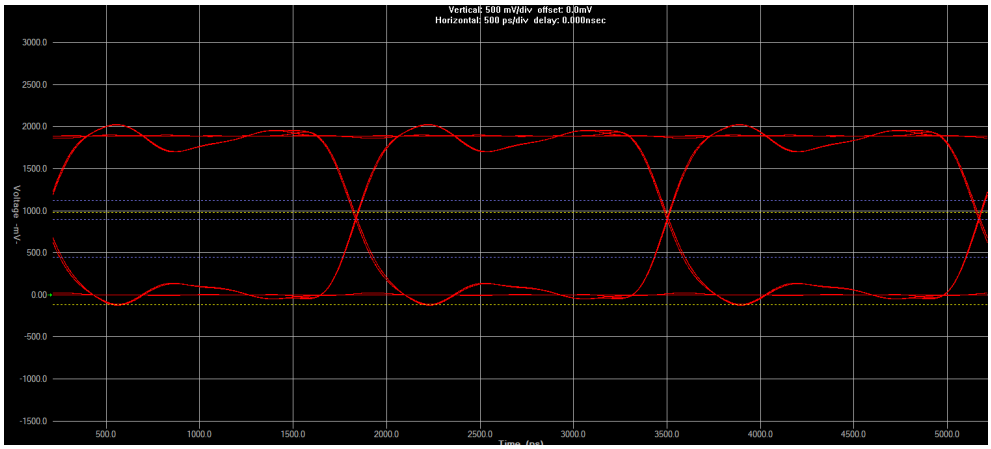
En el presente diseño hay dos factores que facilitan las cosas en términos de integridad de señal, y que aplicando un correcto rutado, evitarán problemas. Por un lado, el reducido tamaño del PCB implica que la mayoría de pistas posean una longitud total muy por debajo de la longitud de onda de las señales que circularán por ellas. Y por otro lado, Aunque los protocolos MIPI y USB3.0 trabajen con elevadas tasas de datos, no es comparable con diseños en los que se alcanzan frecuencias del orden de decenas de GHz.

Bien es cierto que el ancho de banda de las señales no depende del perido que posea la señal, sino de los tiempos de transición entre diferentes estados lógicos. Por consiguiente, será necesario acudir a un simulador para obtener los diagramas de ojos correspondientes y poder asegurar una correcta integridad de señal, antes de lanzar la fabricación del diseño.

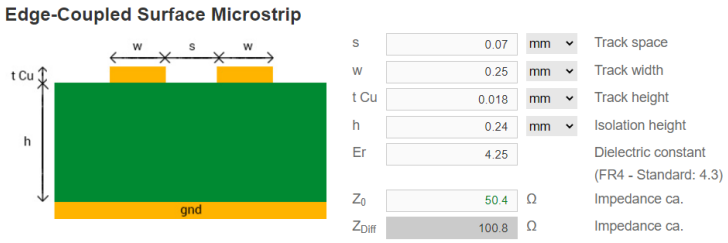
Las señales de clock suministradas por el *Clock Buffer*, se rutarán en una de las capas intermedias, por lo que contarán con las siguientes dimensiones en función del *stack-up*:



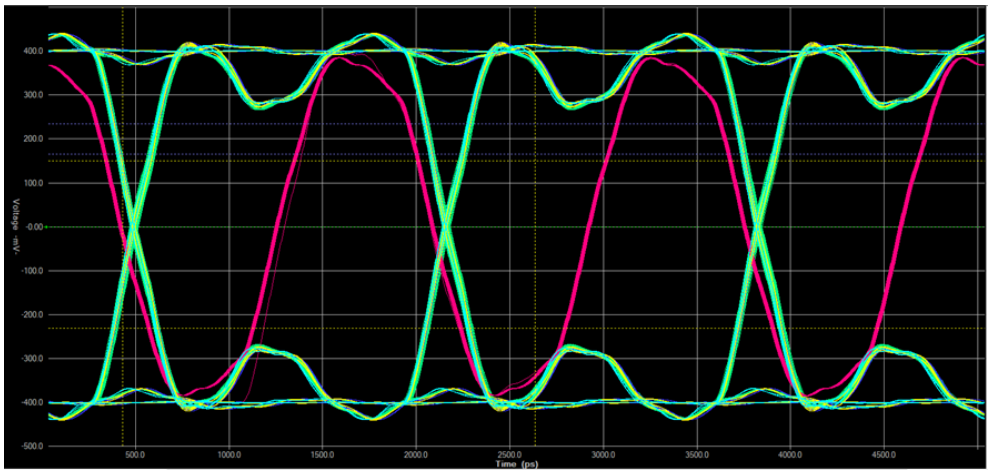
Se lanza la simulación del diagrama de ojo para la señal de clock, tras haber asignado los pines pertinentes de los ficheros IBIS de cada integrado.



Los pares diferenciales tanto del bus MIPI como del USB3.0 serán rutados por las capas exteriores y deberán de seguir dentro de lo posible las siguientes dimensiones:



La simulación del diagrama de ojo de los pares diferenciales del bus MIPI presenta mayor complejidad. Será necesario representar la señal diferencial correspondiente a cada par, y retardar la señal de clock la mitad del ancho de bit, ya que así lo plantea el protocolo.



En la simulación se puede observar como el ojo esta bastante abierto en ambos ejes. En colores claros se representan las señales diferenciales de datos, mientras que la rosa se corresponde con

la señal de clock. Es relevante destacar que no existe excesivo *skew* entre las señales de datos, indicando que los pares diferenciales estan correctamente ecualizados en distancia.

Será primordial **rutar y equidistar** los pares diferenciales pertenecientes a cada bus, **sobre la misma capa**. De no hacerlo, la velocidad de propagación de la señal será diferente, ya que existirá una variación de la constante dieléctrica, tal y como se refleja en la ecuación *2.3*. Esto supondrá la aparición de desfases entre los datos y por consiguiente, una mala transmisión de la información.

$$V_{propagacion} = \frac{C}{\sqrt{\epsilon_r}} \tag{2.3}$$

Bajo los pares diferenciales deberá de existir un plano de cobre limpio, es decir, sin discontinuidades ocasionadas por pistas o vias. De esta forma se reducirán los factores de tener problemas de integridad de señal asociado a las corrientes de retorno.

2.13. Resultado Final

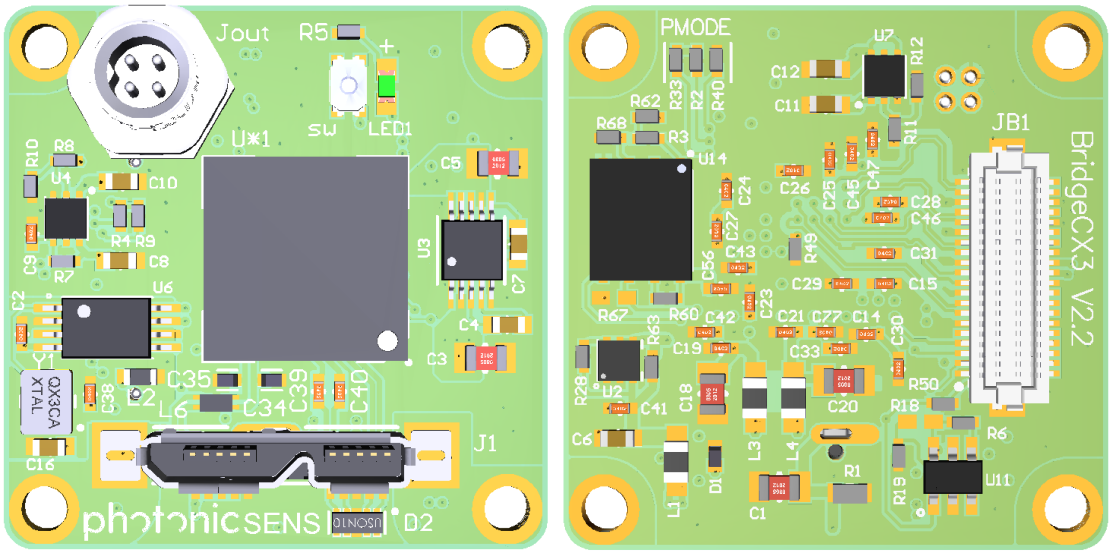


Figura 2.11: BridgeCX3 Top y Bottom

Capítulo 3

Desarrollo del Firmware

Como se ha explicado al inicio del trabajo, el principal objetivo del proyecto es obtener un dispositivo hardware que sea capaz de dos cosas. En primer lugar, poder transferir los píxeles que conforman los *frames* de un CIS a través de USB. Y en segundo lugar, dar la posibilidad a una aplicación host de poder controlar todos los parámetros relevantes de un sensor de imagen en tiempo real, haciendo uso de un driver estándar. A lo largo del capítulo 2 se han explicado los diferentes apartados que conformaban el desarrollo hardware del dispositivo, pero para conseguir el producto final, será necesario embeber un firmware eficiente dentro de la MCU.

El presente apartado tiene como misión, dar una visión global de la estructura del firmware, y sobre todo, explicar con mayor grado de detalle, la arquitectura que conforma el API, que permitirá al dispositivo *host* poder controlar el CIS de una forma robusta, eficiente y alejada de los métodos de control tradicionales adoptados por el driver UVC.

3.1. Arquitectura de Partida

Se parte de un proyecto de ejemplo que proporciona la empresa Cypress, fabricante del CX3, en el que implementan un RTOS que gestiona las diferentes Librerías HAL encargadas de configurar el integrado. Este ejemplo nace con el fin de poder transferir los datos de un sensor de imagen, diferente al utilizado en el presente proyecto. Pero este capítulo no nace con el motivo de explicar estas librerías de alto nivel que Cypress provee para poder programar el CX3, ya que la propia compañía se ha encargado de documentarlas.

Para entender con mayor grado de detalle el desarrollo del firmware, será necesario atender a la siguiente figura.

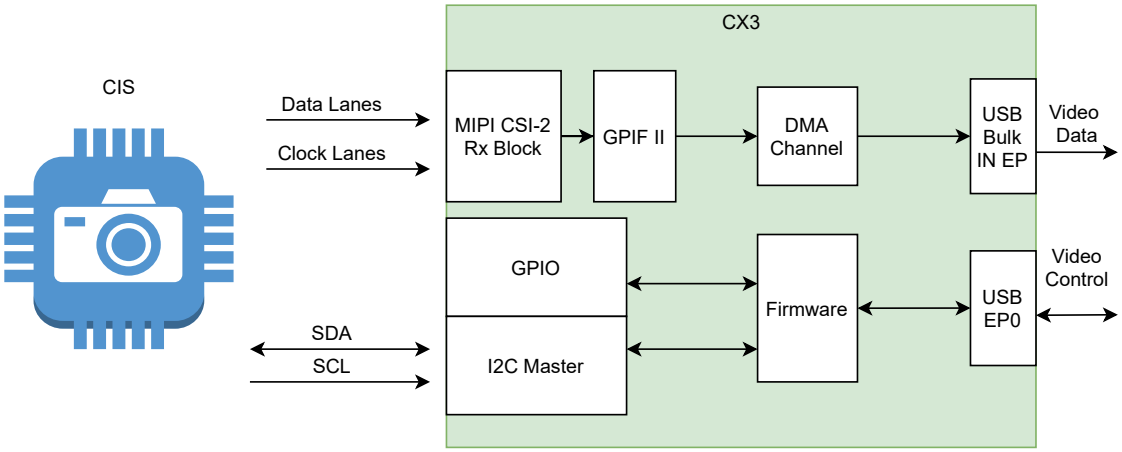


Figura 3.1: Diagrama Interno del CX3

Dentro del bloque verde, están representados en blanco, los diferentes subloques de los que se compone el CX3. Estos subloques son *hard-cores*, es decir, que están implementados en silicio, y que convierten al CX3 en un especie de *ASIC* con la completa funcionalidad de poder convertir los píxeles de un sensor de imagen entre los protocolos MIPI-USB.

Entre los cores más importantes, se encuentran:

- **MIPI CSI-2 Rx Block:** Básicamente, este bloque conforma el PHY del MIPI, siendo totalmente necesario para poder recibir correctamente la información del *frame* proveniente del CIS.
- **GPIF-II** : Es una máquina de estados que se encarga de recibir los datos del frame, en el formato que marca el protocolo MIPI, y que los manipulará para que sigan el protocolo USB y puedan ser enviados a través de éste. Es decir, básicamente se tranta de un traductor entre ambos estándares de comunicación.
- **I2C Master**: Implementa la PHY del bus I2C. Es un bloque imprescindible que servirá para poder inicializar todos los registros del CIS, permitiendo también manipular los diferentes parámetros de control del sensor, haciéndolo trabajar en condiciones óptimas.
- **GPIO**: Como toda MCU, el CX3 también posee pines de entrada y salida de propósito general, que permitirán controlar diferentes periféricos como LEDs infrarrojos, proyectores laser de puntos, etc.
- **Firmware**: No se conforma como un *hard-core*, sino que representa la estructura de programa que se ejecutará en la MCU. Lo más importante de entender, es que el firmware será como un director de orquesta,encargado de inicializar los diferentes *hard-cores* del CX3, configurar el CIS, configurar el descriptor del driver UVC y gestionar todos los requerimientos del protocolo USB.

Tras alcanzar un nivel de entendimiento del código inicial y estudiar el comportamiento de los diferentes bloques internos del CX3, se pasó a implementar todas las modificaciones necesarias, para adaptarlo al CIS de apiCAM©.Los esfuerzos se centraron en dos aspectos fundamentales:

- Transferir al CIS, a través de I2C,el valor de todos los registros necesarios para su configuración inicial.
- Customizar todos los parámetros de diseño del core MIPI CSI-2 RX ad-hoc para el tamaño del sensor, número de bits por pixel, blanking vertical y horizontal, etc.

Conseguido lo anterior, se estará en disposición de poder desarrollar el API.

3.2. Interfaz API

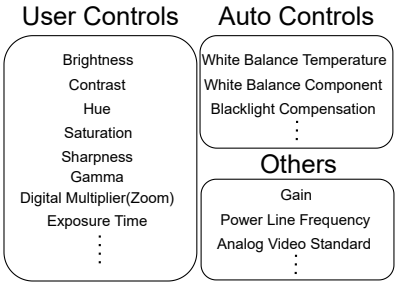
3.2.1. Estándar UVC

Dado que el firmware se basa en el estándar UVC1.1 [6], la interfaz queda definida prácticamente por este. Sin embargo, debido a las características de control que la aplicación software, que se ejecuta desde el dispositivo host requiere hacer sobre el CIS, será necesario implementar un *handshake* propio, a través de una serie de propiedades UVC, en las que el dato a transferir no se corresponderá con la acción a controlar, dada por su propia definición.

Para entender esto, primero será necesario explicar de forma general cómo el estándar permite modificar y controlar los parámetros más comunes de cualquier cámara. UVC permite a los desarrolladores la posibilidad de habilitar lo que denomina **Enumeration Data**, que no son más que una serie de terminales adecuados para representar la mayoría de las funciones de vídeo de los dispositivos actuales y ampliable a dispositivos futuros. En resumidas palabras, unas propiedades que en el presente caso, habilitarán el intercambio de datos entre la aplicación *host* y el **Bridge CX3**, permitiendo el control de los parámetros más relevantes del CIS. El estándar UVC los agrupa en 4 tipos:

- Input Camera Terminal (IT).
- Output Terminal (OT).
- Processing Unit (PU).
- Extension Unit (EU).

Para mayor conocimiento de estos, se recomienda acudir a la documentación oficial **Universal Serial Bus Device Class Definition for Video Devices Revision 1.1**[6]. De los 4 tipos, se centrará la atención sobre las **Processing Unit (PU)** que permiten el soporte de las siguientes funcionalidades de control:



3.2.2. Esquema de control

De entre todas las posibilidades, se han escogido las propiedades **Contrast**,**Saturation** y **Sharpness** para realizar el *handshake* propio. Debido a que son las únicas que permiten el envío y recepción de enteros de 16 bits sin signo, sin que el driver ejerza ninguna modificación sobre el valor, lo que supone un rango útil de valores comprendido entre 0 y 65535. Los valores del resto de las propiedades UVC contempladas, eran manipulado en el driver, convirtiéndolas en opciones imposibles de implementar.

Cabe destacar que será posible utilizar cualquiera de las propiedades UVC, tal y como las contempla el estándar, por lo que, bastará con habilitarla en el firmware del Bridge CX3,e implementar su lógica. De hecho, a modo de ejemplo, en el presente proyecto se habilita la propiedad *Exposure Time*, para que el dispositivo *host* pueda controlarla siguiendo el estándar.

El modelo de control queda con la siguiente estructura:

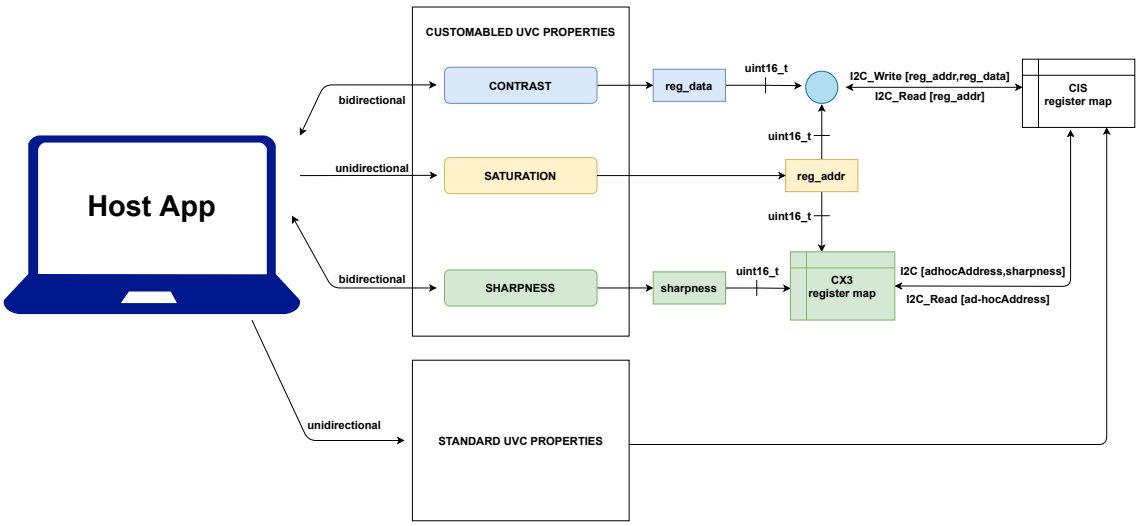


Figura 3.2: Esquema de la estructura de control

Entrando más en detalle en las propiedades UVC customizadas, hay que destacar que, con la actual implementación, la aplicación host podrá acceder directamente a los registros del CIS, tanto para lectura como para escritura, implicando tener la necesidad de conocer su mapa de memoria. Por otro lado, también se le permitirá al host poder acceder a un mapa de registros desarrollado en la MCU, entre los que se contemplan una serie de controles básicos aplicables a cualquier CIS.

Y es que el fin último de este mapa de registros propio del Bridge CX3, es el de dar la posibilidad a la aplicación host de poder controlar los parámetros más relevantes de cualquier CIS, sin necesidad de conocer su propio mapa de registros, que con toda seguridad variará de un CIS a otro. Esto permitirá abstraer al programador de la aplicación host, de tener que implementar todo el control siguiendo el paradigma de programación que marca el estándar, ya que toda la lógica de conversión y adaptación de los controles será llevada a cabo en el propio firmware de la MCU.

Siguiendo el esquema **3.2**, se puede distinguir entre dos grupos de propiedades:

3.2.2.1. Propiedades UVC Customizadas

Son las propiedades UVC utilizadas para la implementación del *handshake* propio.

- **Saturation :** Entero de 16 bits [**reg_addr**] que determina la dirección de memoria.
- **Contrast :** Entero de 16 bits[**reg_data**] que determina el valor a escribir en un registro del CIS, cuya dirección de registro viene determinada por reg_addr.
- **Sharpness :** Entero de 16 bits[**sharpness**] que determina el valor a escribir en el registro del mapa de memoria interno del CX3, cuya dirección de registro viene determinada por reg_addr.

3.2.2.2. Propiedades UVC Estándar

Son las propiedades UVC que siguen el paradigma de control marcado por el estándar UVC.

- **Exposure Time :** El valor de la propiedad especifica la duración de la exposición. Este valor se expresa en base logarítmica de 2. Por lo que para valores inferiores a 0 el tiempo de exposición vendrá determinado por $(1/2^{valor})$ segundos. Mientras que, para valores positivos y cero, el tiempo de exposición será de $(2 \cdot valor)$ segundos.

La idea de partida será que los desarrolladores de la aplicación software, no tengan que hacer uso de las propiedades estándar. Si no que lleven a cabo todos los controles necesarios a través de las propiedades customizadas, haciendo uso del mapa de registros del CX3, abstrayéndose por completo del sensor que albergue la cámara.

3.2.3. Handshake directo con el CIS

Implementando este método, se podrá acceder directamente a la lectura/escritura de cualquiera de los registros del CIS. Para ello, siguiendo el esquema visto en el apartado anterior, se hará uso de las propiedades UVC, Saturation (dirección de memoria del registro del CIS) y Contrast (dato a escribir o leer en la dirección de memoria correspondiente).

Será importante seguir el orden exhaustivo en las transacciones a realizar desde la aplicación *host*. Ya que, en caso de enviar el valor de una propiedad, antes que la otra, implicará estar enviando el valor del dato en el campo de dirección de memoria y viceversa, conllevando una escritura errónea.

3.2.3.1. Proceso de Escritura

Se muestra el *handshake* entre los diferentes agentes que intervienen en el proceso de escritura de un registro cualquiera del CIS:

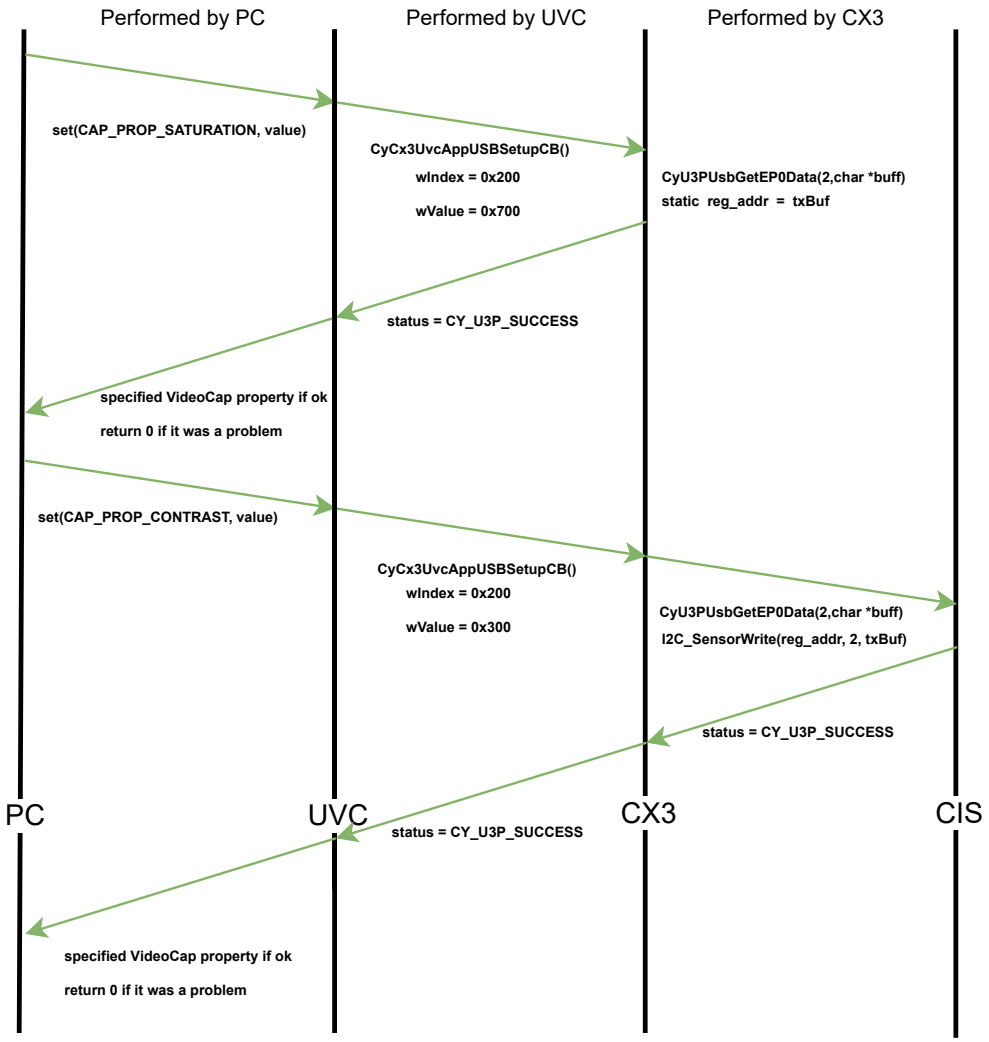


Figura 3.3: Handshake de escritura directa con los registros del CIS

Tanto para el presente diagrama, como en los tres siguientes, las funciones de escritura y lectura realizadas por el *host* pertenecen a la librería **OpenCV**. Todo esto se verá con más detalle en el ***capítulo 4***.

El valor a enviar en ambas propiedades se corresponderá con un entero de 16 bits sin signo, sugiriéndose la recomendación de escribirlos en formato hexadecimal, ya que es así como se representan los registros y los valores que albergan en las hojas de datos del CIS.

3.2.3.2. Proceso de Lectura

Se muestra el handshake entre los diferentes agentes que intervienen en el proceso de lectura de un registro cualquiera del CIS:

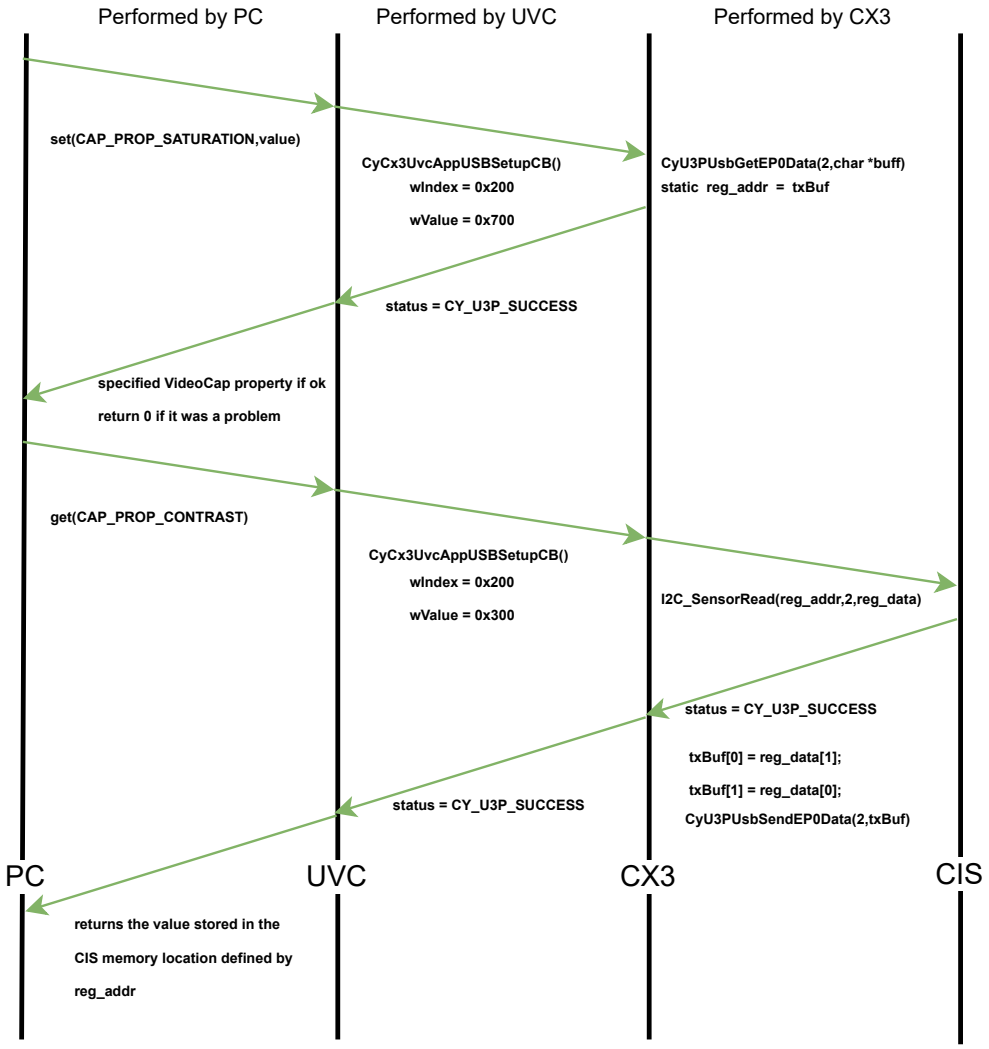


Figura 3.4: Handshake de lectura directa con los registros del CIS

3.2.4. Handshake con el mapa de registros del CX3

Implementando este método, se podrá acceder a la lectura/escritura del mapa de registros propio del CX3, por lo que se podrá manipular los parámetros más importantes de un CIS, sea cual sea este, y con independencia del conocimiento del mapa de registros propio del sensor. Para ello, se hará uso de las propiedades UVC: Saturation (dirección de memoria) y Sharpness (dato a escribir o leer en la dirección de memoria correspondiente).

Al igual que sucede en el apartado anterior, habrá que seguir el orden preciso en las transacciones a realizar desde la aplicación host, ya que, en caso de enviar el valor de una propiedad antes que la

otra, implicará estar enviando el valor del dato en el campo de dirección de memoria y viceversa, conllevando una escritura errónea.

3.2.4.1. Proceso de Escritura

En este apartado, se muestra el handshake entre los diferentes agentes que intervienen en el proceso de lectura de un registro del mapa de memoria de la MCU .

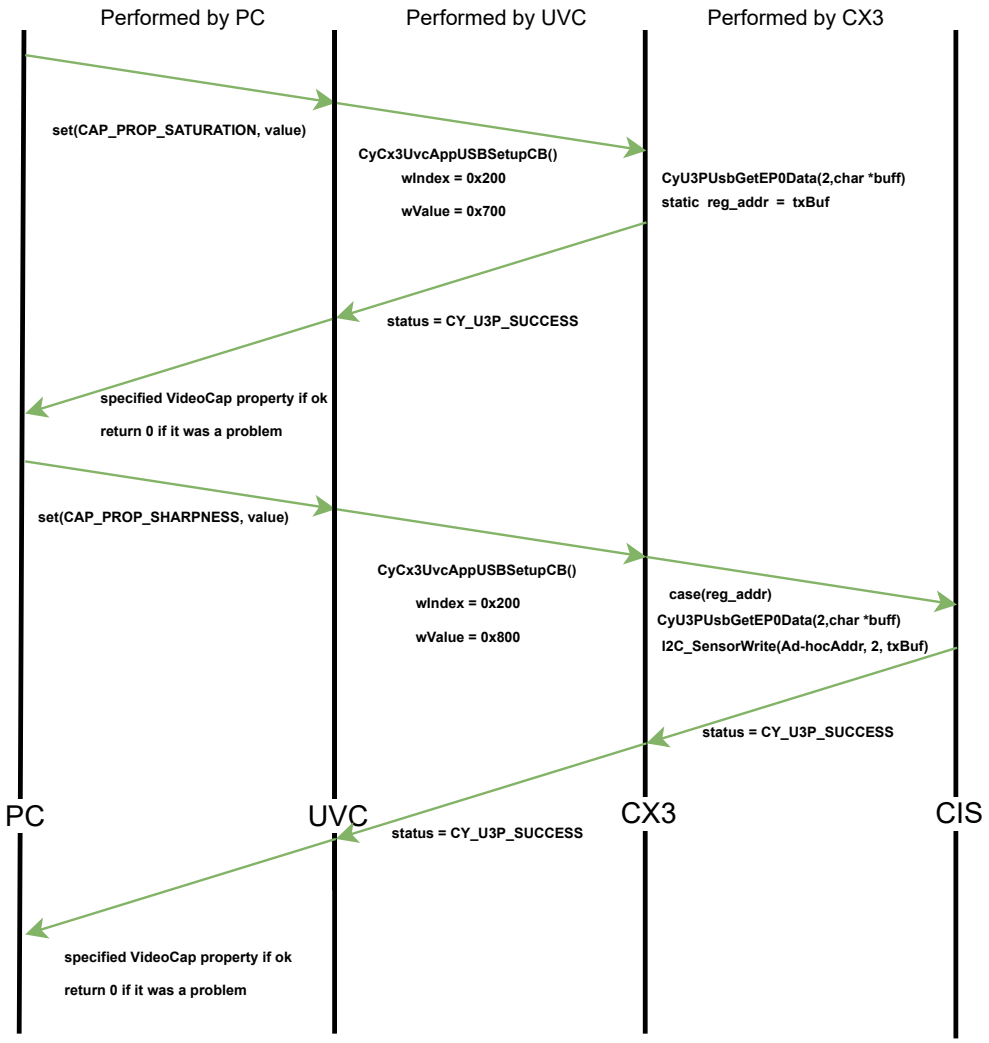


Figura 3.5: Handshake de escritura al mapa de registros del CX3

Observando el diagrama se puede apreciar la gran similitud a la hora de realizar la transacción desde la aplicación host, en el que la única modificación con respecto al proceso de escritura de un registro directo del CIS, es que ahora el valor del dato se enviará a través de la propiedad Sharpness. De nuevo, el valor a enviar será un tipo de dato entero de 16 bits sin signo.

3.2.4.2. Proceso de Lectura

Se muestra el handshake entre los diferentes agentes que intervienen en el proceso de lectura de un registro del mapa de memoria del CX3:

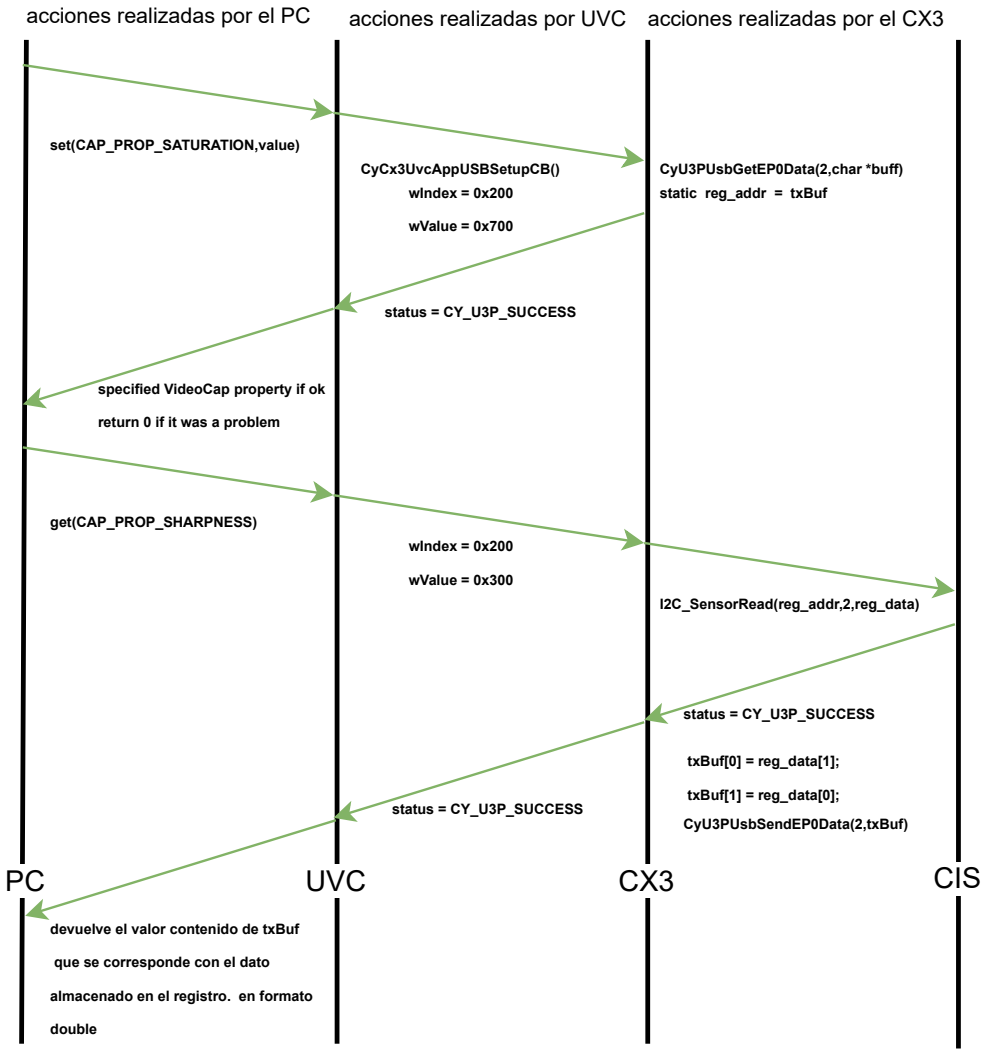


Figura 3.6: Handshake de lectura del mapa de registros del CX3

Observando el diagrama se puede apreciar la gran similitud a la hora de ejecutar la transacción desde la aplicación host, siendo las propiedades utilizadas, la única modificación con respecto al proceso de lectura de un registro directo del CIS.

3.2.5. Mapa de registros de la MCU (CX3)

Como ya se ha recalcado, el mapa de memoria de la MCU nace con el fin de poder controlar, desde la aplicación *host*, los parámetros más importantes de un CIS, sea cual sea este. Es decir, que los desarrolladores del software no requerirán del conocimiento del mapa de registros propio del

sensor. Únicamente bastará con acceder a la dirección del registro de la MCU encargada de cada control.

Además, también existirá la posibilidad de habilitar registros para el control de periféricos, información de la versión firmware que se está ejecutando en la MCU, y multitud de funcionalidades futuras.

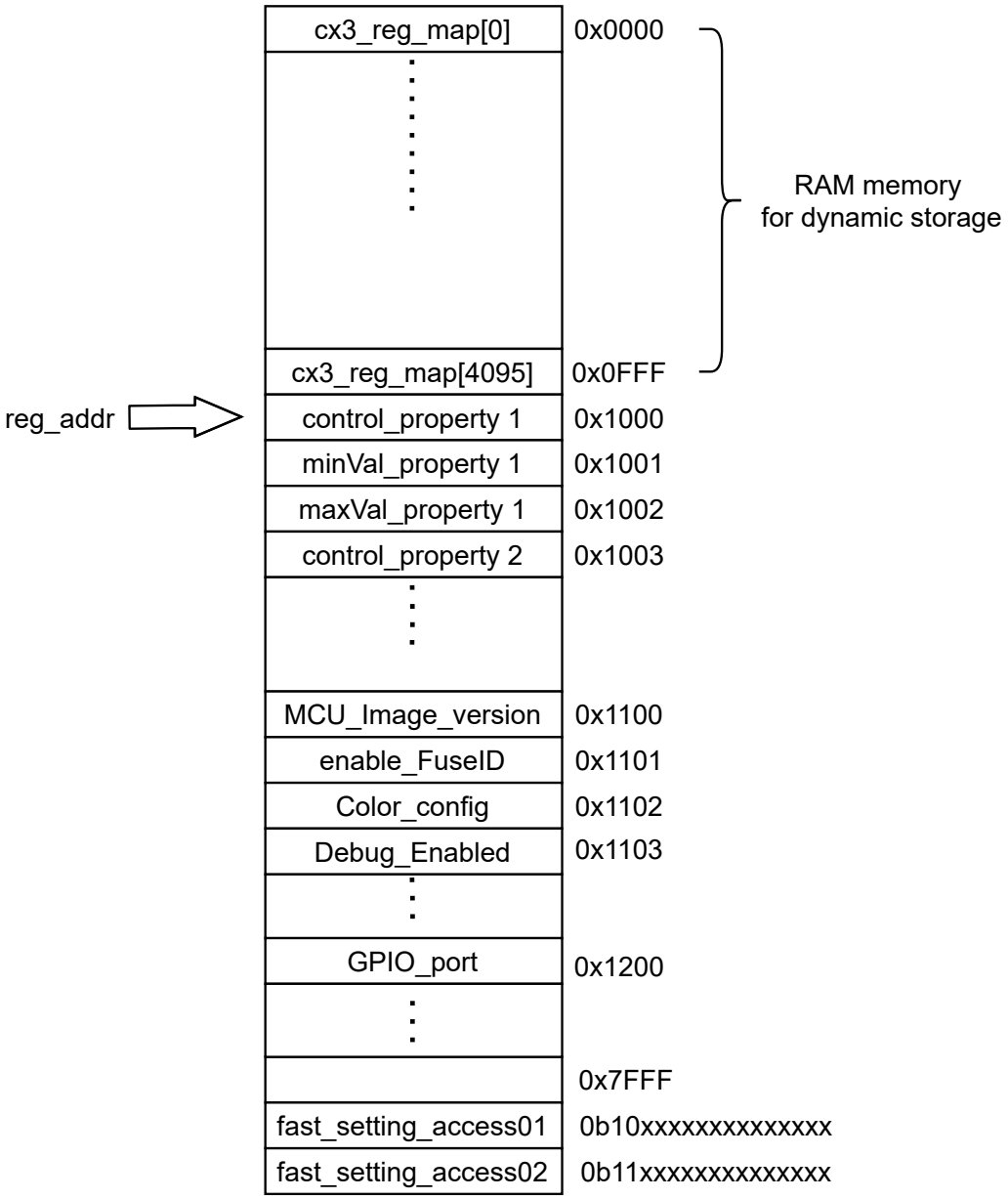


Figura 3.7: Mapa de Memoria de la MCU (CX3)

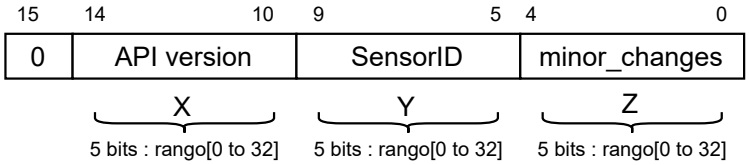
El acceso a los registros de esta memoria vendrá determinado por el valor de reg_addr, el cual es configurado a través de la propiedad UVC Saturation. Dicho valor se corresponde con un tipo de dato uint16_t, por lo que la memoria tendrá un tamaño de $2^{16} - 1$ posiciones y un ancho de palabra de 16 bits.

- Las primeras 4096 posiciones, **[0x0000-0x0FFF]**, se reservan para realizar almacenamiento dinámico. Por ejemplo, el valor del *fuseID* del sensor u otros valores que necesitan ser almacenados dentro de la aplicación. Actualmente cuenta con una capacidad de almacenamiento máximo de 8 KBytes, aunque actualmente solo se habilitan 512 Bytes.

- El rango de direcciones comprendido entre **[0x1000-0x10FF]** se reserva exclusivamente para la manipulación de los controles más importantes del CIS. Además, las dos posiciones de memoria contiguas a cada registro de control determinarán su valor mínimo y su valor máximo permitido. Información que podrá ser consultada por la aplicación software que se ejecuta desde el PC, con el fin de acotar los rangos.

- El registro**0x1100**, denominado **MCU_Image_version**, almacenará el valor de la versión del firmware que en ese momento corre en la MCU. Su valor vendrá determinado por: la versión del API, el CIS que embeba el dispositivo y por los cambios menores que se realicen en la estructura del algoritmo. ***Ir al apéndice B*** para un mayor entendimiento.

  Contando con la siguiente interpretación binaria:



  Actualmente se contemplan los siguientes campos:

API Version	Referencia
1	Interfaz API V.1
-	-

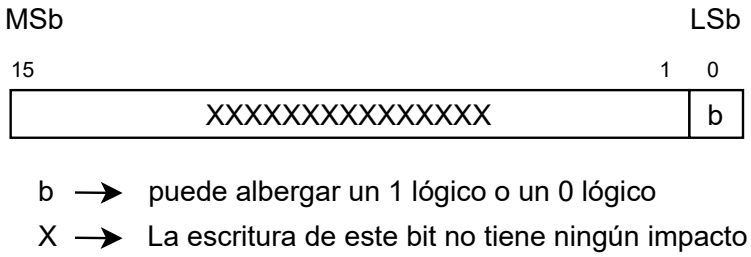
Tabla 3.1: Versiones API

SensorID	Referencia
1	ON-Semi CIS
-	-

Tabla 3.2: Sensores Contemplados

- El registro **0x1101**, denominado **enable_FuseID**, permitirá a la aplicación *host* poder activar la obtención del número identificador del CIS. Es decir, una vez seteado, se cargará el fuseID en la zona del mapa destinada a almacenamiento dinámico, comenzando dicho almacenamiento a partir de la dirección 0x0000 y ocupando tantas celdas de memoria consecutivas como tamaño posea su codificación.

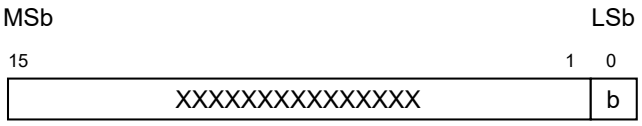
El registro únicamente permite la escritura del LSb, por lo que seteando dicho bit a nivel alto, implicará comenzar la transacción.



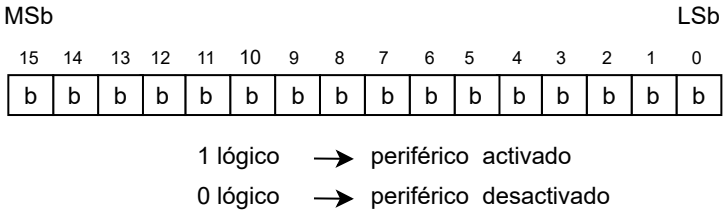
- El registro **0x1102**, denominado **Color_config**, dará la posibilidad al host de conocer si el CIS conectado es monocromático o color.

- El registro **0x1103**, denominado **Debug_Enabled**, dará la posibilidad a la aplicación *host* de poder habilitar o deshabilitar el proceso de *debug*, visualizando todo lo referente al funcionamiento de la API. Una vez habilitado, bastará con conectar el cable de debug del puerto serie a un puerto USB del PC.

  El registro únicamente permite la escritura del LSb. Bastará con establecerlo a nivel lógico alto para habilitar el debug, o establecerlo a nivel lógico bajo para deshabilitarlo.



- El registro **0x1200**, denominado **GPIO_port**, se destinará a manipular parte de los periféricos digitales que embeba la placa Bridge CX3. Al tratarse de un registro con 16 bits de ancho de palabra, y teniendo en cuenta que el control de cada uno de estos dispositivos es booleano, implicará el control de un total de 16 periféricos.



Actualmente, la placa BridgeCX3 cuenta con los siguientes periféricos:

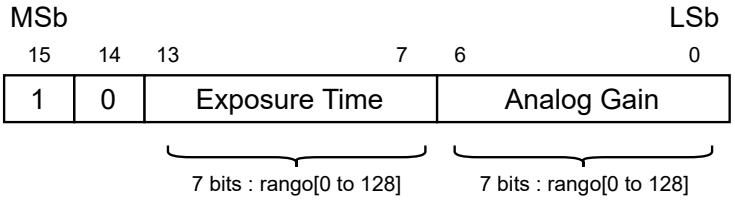
PERIFÉRICO	Bit de control
Embedded Green LED	GPIO_port[0]
-	-

Tabla 3.3: Perifericos disponibles

- Por último, se dota la posibilidad al *host* de poder **setear dos controles del CIS en la misma transacción**. Esto supone doblegar el ***throughput***, siendo únicamente necesario la escritura de una de las propiedades UVC customizadas para llevarse a cabo. Por el contrario, se reducirán los bits de precisión con los que se codifica cada control del CIS.

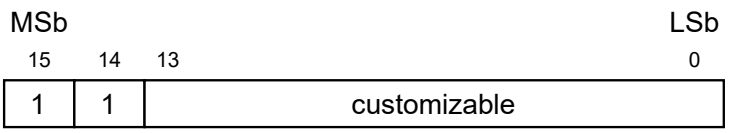
  Para que esto sea posible, se sacrificará la mitad mapa de registros de la MCU, es decir, aquellas direcciones con un 1 en su bit más significativo, a costa de ganar un par de registros que podrán configurar dos controles del CIS en la misma transacción UVC, utilizando la propiedad **Saturation**.

  Registro 0b10xxxxxxxxxxxxx, denominado **fast_setting_access01**. Se encargará de establecer el valor del Exposure Time y de la Ganancia analógica en la misma transacción, y cuenta con la siguiente interpretación binaria:



  Al reducir el número de bits de cuantificación con el que se codifican los valores de cada control, la resolución se verá afectada, es decir, el paso mínimo entre valores codificados contiguos aumentará considerablemente. **Cada CIS determinará unos rangos de valores para cada control, por lo que habrá que estudiar la resolución resultante en cada caso**.

  Registro 0b11xxxxxxxxxxxxx, denominado **fast_setting_access02**. Se reserva con el mismo fin que el registro anterior, aunque actualmente no contempla la configuración de ningún control.



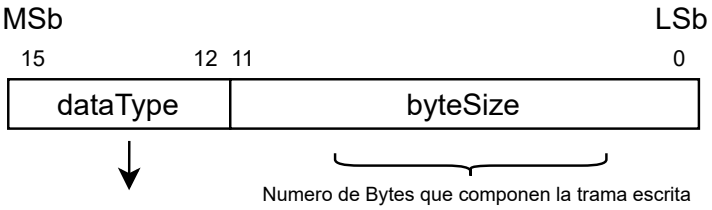
3.2.6. Envío y Recepción de paquetes de datos

En el apartado anterior se explicó que el rango de direcciones comprendido entre **[0x0000-0x0FFF]**, se reserva para almacenamiento dinámico de memoria. De manera que será posible enviar y recibir paquetes de datos entre el *host* y el CX3.

Para poder realizar las transacciones de paquetes a través del driver UVC, habrá que enviar o recibir los datos uno a uno accediendo a las posiciones contiguas de memoria, siguiendo el siguiente protocolo:

3.2.6.1. Escritura

El agente que tenga que realizar la acción de escritura, deberá de informar de la cantidad de Bytes que va a escribir, así como el tipo de dato del que se trata. Esto lo plasmará siempre en la dirección 0x0000 del mapa de registros, con la siguiente configuración binaria:



tipo	valor
Ascii	0
Byte	1
U16	2
U32	3
BCD	4
Nibble	5

A continuación, realizará la escritura de todos los bytes comenzando en la dirección 0x0001, y continuando en las celdas adyacentes. Sea cual sea el tipo de dato que se envíe, será necesario codificarlo en formatos de palabra de 16 bit.

> ***Nota:*** *cada celda del mapa de memoria posee la capacidad de almacenar 2 bytes, por lo que el número de celdas escritas en la trama vendrá determinada por la siguiente expresión:*
>
> $$N_{celdas} = ((byteSize/2\,\%\,2) == 0)\ ?\ \ byteSize/2\ :\ (byteSize/2) + 1$$
>
> *Expresa que en caso de tratarse de un número de bytes par, el total de celdas escritas será la mitad. Mientras que si el número de bytes es impar, será necesario añadir una celda más, ya que el dato será escrito en el byte menos significativo de la celda adicional.*

3.2.6.2. Lectura

El agente que tenga que realizar la acción de lectura, realizará el proceso inverso, siguiendo los siguientes pasos:

- Leer la dirección 0x0000, donde viene codificada la información de la trama.
- Realizar un bucle de lectura de posiciones consecutivas del mapa de registros comenzando en la dirección 0x0001, e iterando N_{celdas} veces.
- Interpretar la información recibida, en función del tipo de dato del que se trate.

3.2.7. Perfiles de visualización

Un comportamiento habitual en la mayoría de cámaras, es poder obtener diferentes perfiles de imagen, a través del mismo hardware. Así pues, el presente firmware dotará la posibilidad al host de poder seleccionar entre diversos perfiles de visualización. Todos ellos orientados a cubrir las necesidades de la aplicación software.

En el caso del Bridge CX3, las características más importantes que harán distinguir entre diferentes perfiles son:

- El ancho y el alto de la imagen, es decir, la cantidad de píxeles que forman cada *frame*.
- La longitud binaria con la que cuenta cada píxel, siendo 8 bits y 10 bits las más comunes.
- La codificación binaria de cada píxel, que es indispensable para el tratamiento software posterior, ya que, dicha codificación, marcará el proceso algorítmico que tiene que llevar a cabo el *host* para formar la imagen. Un claro ejemplo de dos codificaciones diferentes suele ser la codificación de 8 bits sin DPCM, en la que se envía el raw del valor de píxel. Y la codificación de 8 bits con DPCM 8/10, en la que cada píxel envía la diferencia con respecto al valor del píxel anterior.

El inconveniente principal a la hora de setear los parámetros de un nuevo perfil, es que habrá que jugar con las reglas de los actores principales que hacen posible la transacción del frame entre el CIS y el dispostivo *host*, que en el presente caso son, por un lado, el protocolo del bus MIPI y por otro lado la propia arquitectura del CX3.

Ambos impondrán restricciones de diseño que habrá que tener en cuenta, ya que afectarán al tamaño total del *frame* a enviar, y por consiguiente, repercutirán tanto al ancho y alto como al tamaño binario de cada pixel. En la siguiente Imagen se muestra de manera orientativa las condiciones que deben ser cumplidas para obtener un perfil funcional:

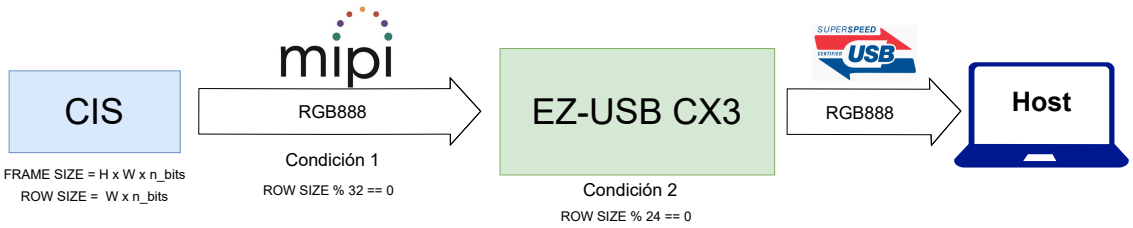


Figura 3.8: Condiciones a cumplir por los perfiles de visualización

Según el esquema, lo que acaba determinando un perfil funcional válido, es el tamaño en bits de cada fila, ya que dicho tamaño deberá cumplir las dos condiciones siguientes.

3.2.7.1. Condición 1

Para entender esta condición, será necesario acudir a la documentación oficial en lo referente al protocolo MIPI[7]. En ella, muestra de forma gráfica, como se gestiona la recepción de los datos en formato RGB888 por parte del core *CSI-2 Receiver*. Recordad que era el core que debe embeber cualquier integrado que sea capaz de recibir datos a través del bus MIPI.

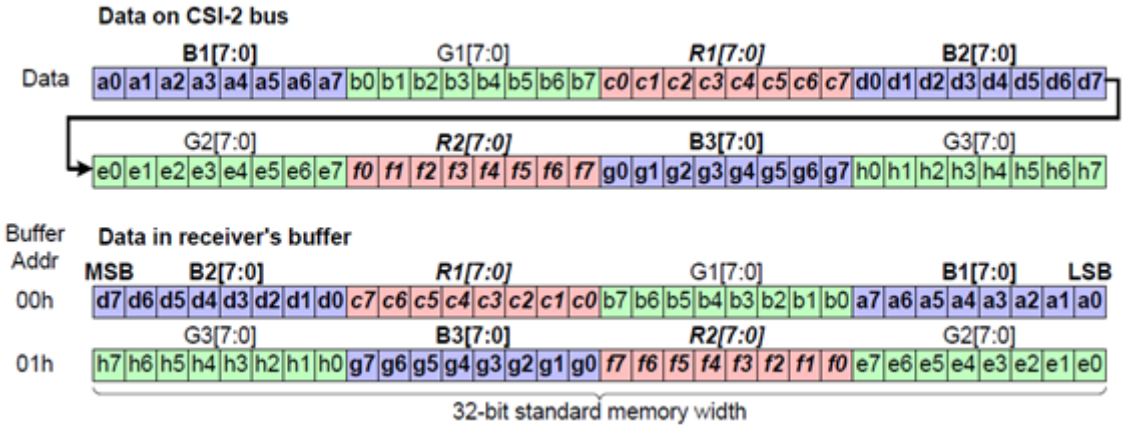


Figura 3.9: RGB888 Data Format Reception [7]

En la imagen se puede ver que el *CSI-2 receiver* utiliza un ancho de memoria de 32 bits para formar cada paquete de información de los datos que recibe del CIS. El hecho de que el tamaño en bits de cada fila no sea múltiplo de 32 hará que alguno de los paquetes no se complete, ocasionando una transacción inválida.

3.2.7.2. Condición 2

Para entender la segunda condición, se deberá consultar la documentación del CX3[3], en la que se muestran los diferentes formatos de video que es capaz de enviar hacia el host. Entre los formatos se encuentra el **RGB888**, que es el que se deberá seleccionar en el caso de querer alcanzar el máximo baudrate.

Ya que es el formato que pose el *payload* más elevado, permitiendo que el total de los bits enviados en cada transacción, se correspondan con información útil referente a un píxel concreto, y no a información vacía, como pudieran ser los bits de *zero-padding*. Esto último sucede en el envío del formato RAW10 en paquetes de 16 bits.

Configurar el CX3 para que envíe los datos en formato RGB888, conllevará que el tamaño en bits de cada fila sea múltiplo de 24, ya que, de no serlo, alguno de los paquetes no se completará, quedando corrupto, e imposibilitando la transacción al *host*.

3.3. Customización del API para el ON-Semi CIS

En este apartado se explicará de manera detallada, todos los controles que se han habilitado en el mapa de registros del CX3. Con el fin de poder manipular los parámetros más importantes del sensor que embebe apiCAM©. Para cada control se especificará tanto su dirección, como el rango de valores que soporta.

Gracias a esta customización, la aplicación host podrá controlar el CIS de una forma mucho más amena e intuitiva, evitándole la tarea de adaptar cada parámetro de control a sus correspondientes valores hexadecimales entendidos por el CIS. Simplemente, manipulará cada parámetro a través de unidades de medida mucho más cercanas al entendimiento humano.

3.3.1. Exposure Time

Según la documentación oficial del CIS, la relación entre el tiempo de exposición y el valor a escribir en su registro correspondiente, viene determinada por la siguiente expresión:

$$IntegrationTime(s) = \frac{registerVal \cdot line\_length}{2 \cdot vt\_pixel\_clock\_freq \cdot 10^6} \tag{3.1}$$

Es relevante destacar que existe una relación directa entre el tiempo de exposición de un sensor y su configuración del PLL, ya que ***vt_pixel_clock_freq*** se corresponde con uno de los clocks obtenidos del propio PLL, siendo su ecuación la siguiente. Ver el ***capítulo 5*** para mayor entendimiento.

$$vt\_pixel\_clock\_freq = \frac{19{,}2 \cdot PLL\_MULTIPLIER}{20} = \frac{19{,}2 \cdot 0xA0}{20} = 153{,}6Mhz \tag{3.2}$$

El valor del registro del CIS denominado ***PLL_MULTIPLIER***, determina su *frame rate*. Por lo que, en caso de modificarlo, habrá que recalcular la constante de proporcionalidad que relaciona el tiempo de exposición configurada por el *host* y el valor de registro a escribir en CIS.

Por consiguiente, para la presente configuración, la relación entre el valor de tiempo de exposición en segundos y el valor de registro a escribir en el CIS se reduce a:

$$registerVal = \frac{2 \cdot 153{,}6 \cdot 10^6}{4656 \cdot 10000} \times IntegrationTime(100us) \tag{3.3}$$

$$registerVal = 6{,}597938144 \times IntegrationTime(100us) \tag{3.4}$$

teniendo en cuenta que el registro posee un ancho de palabra de 16 bits, se pueden determinar los valores máximo y mínimo permitidos, siendo el valor mínimo el entero más cercano al valor de la pendiente, en este caso $6{,}59 \approx 7$:

$$Integration\_Time\_min = \frac{7}{6{,}597938144} = 1 \cdot 100us = 0{,}1ms$$

$$Integration\_Time\_min = \frac{65535}{6{,}597938144} = 9932 \cdot 100us = 993ms$$

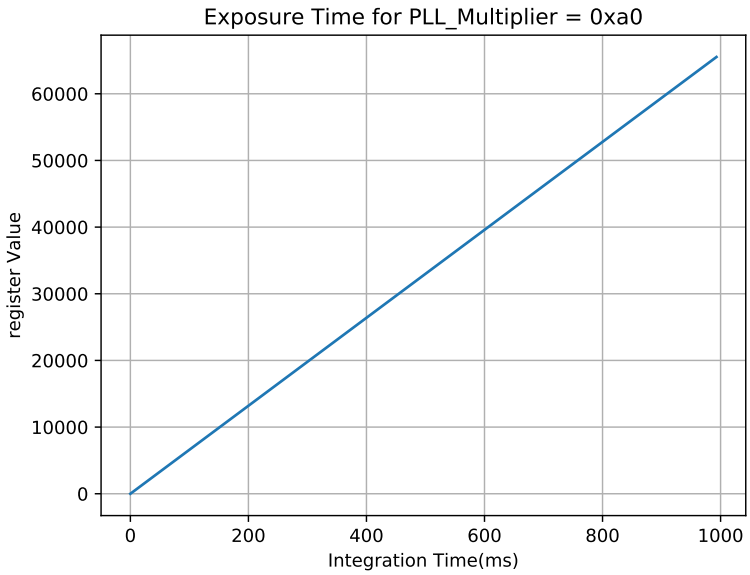


Figura 3.10: Relación Exposure Time - valor del registro

3.3.1.1. Direcciones y valores permitidos

La aplicación *host* ajustará directamente el valor de tiempo de exposición deseado, olvidándose por completo de convertir dicho valor al entero de 16 bits que configura esa exposición en el registro del CIS. Se habilitan las siguientes direcciones en el mapa de registros del CX3:

Motivo de uso	Acción	Valor	Dirección
Exposure Time	Escritura/Lectura	Centenas de us	0x1000
Get min Value	Lectura	1	0x1001
Get max Value	Lectura	9930	0x1002

Tabla 3.4: Direcciones Exposure Time

3.3.2. Analog Gain

La ganancia se obtiene ajustando el voltaje de referencia del *colum-parallel ADC*. Cuando desde la aplicación *host* se modifica el valor de ganancia, lo que realmente se está haciendo es un ajuste de la ganancia total, la cual se calculará internamente a través de la ganancia analógica (ajuste grueso) y de la ganancia digital (ajuste fino).

El ajuste de la ganancia se ceñirá a la **referencia ISO**, tabulada y proporcionada en la hoja de datos del CIS. Entrando en mayor grado de detalle, se observan dos tramos con comportamientos diferentes, lo que invita a plantear una solución basada en una función a tramos.

En el tramo comprendido entre valores **ISO-80 a ISO-775**, el ajuste de la ganancia se realiza únicamente a través de un ajuste grueso, es decir, a través de la componente analógica, encontrándose la componente digital configurada a su valor mínimo. Graficando los valores tabulados en la hoja de datos para el presente rango, se observa una relación logarítmica, que sigue la siguiente ecuación:

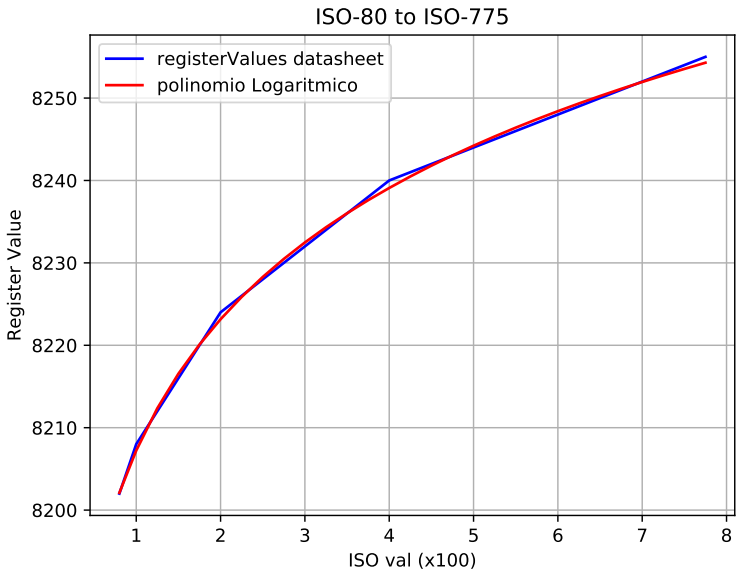


Figura 3.11: Relación Analog Gain - valor del registro Tramo Logarítmico

$$registerVal = 22{,}966 \cdot \log(ISO_{val}) + 8207{,}254 \tag{3.5}$$

Sin embargo, para el tramo correspondiente entre **ISO-775 a ISO-3200**. El ajuste de la ganancia total se realiza a través de un ajuste fino, es decir, a través de la componente digital, encontrandose la componente analógica configurada a su valor máximo. Al graficar los valores para el correspondiente rango, se observa una relación lineal:

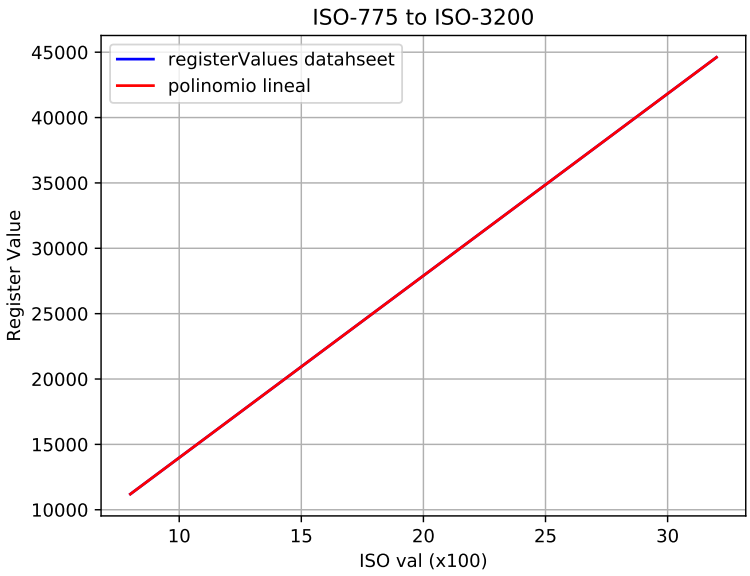


Figura 3.12: Relación Analog Gain - valor del registro Tramo Lineal

$$registerVal = 13{,}92 \cdot ISO_{val} + 62{,}99 \tag{3.6}$$

3.3.2.1. Direcciones y valores permitidos

La aplicación *host* ajustará directamente el valor ISO. Olvidándose por completo de convertir dicho valor al entero de 16 bits que setea ese valor de ganancia en el registro del CIS.

Se deberá acceder a las siguientes direcciones del mapa de registros del CX3:

Motivo de uso	Acción	Valor	Dirección
Analog Gain	Escritura/Lectura	ISO value	0x1003
Get min Value	Lectura	80	0x1004
Get max Value	Lectura	3200	0x1005

Tabla 3.5: Direcciones Analog Gain

3.3.3. Image Orientation

El CIS permitirá cambiar la orientación del *frame* a nivel hardware. Es decir, dará la posibilidad de voltear la imagen en su eje vertical, al igual que rotarla en su eje horizontal. Para ello el *host* deberá setear alguno de estos valores:

Valor	orientación
0	Orientación por defecto
1	Rotación horizontal
2	Volteado vertical
3	Rotación horizontal y Volteado vertical

Tabla 3.6: Orientaciones del frame

3.3.3.1. Direcciones y valores permitidos

Se deberá acceder a las siguientes direcciones del mapa de registros del CX3:

Motivo de uso	Acción	Valor	Dirección
Image Orientation	Escritura/Lectura	[0to3]	0x1006
Get min Value	Lectura	0	0x1007
Get max Value	Lectura	3	0x1008

Tabla 3.7: Direcciones Image Orientation

3.3.4. CIS Temperature

El CIS cuenta con un sensor de temperatura *on-chip*. Esto brinda la oportunidad de poder leer el valor de temperatura del sensor durante el uso de la aplicación. Resultando de gran utilidad para conocer su estado, desarrollar mecanismos de seguridad evitando sobrecalentamientos del integrado, etc.

Obtener el valor de temperatura no será tan trivial como leer un único registro del CIS, sino que para su cálculo, intervienen el valor de dos registros: **Temp_reg** y **Temp_calib**. El primero se corresponde con el valor en crudo que devuelve el propio sensor de temperatura. Mientras que el segundo se corresponde con un parámetro de calibración que afecta directamente a la ordenada en el origen, supliendo la variabilidad de la lectura de temperatura para diferentes CIS, ocasionada por la propia implementación en silicio del integrado y dando como resultado una completa uniformidad en la medida. Todo esto queda expresado con mayor claridad en la siguiente figura.

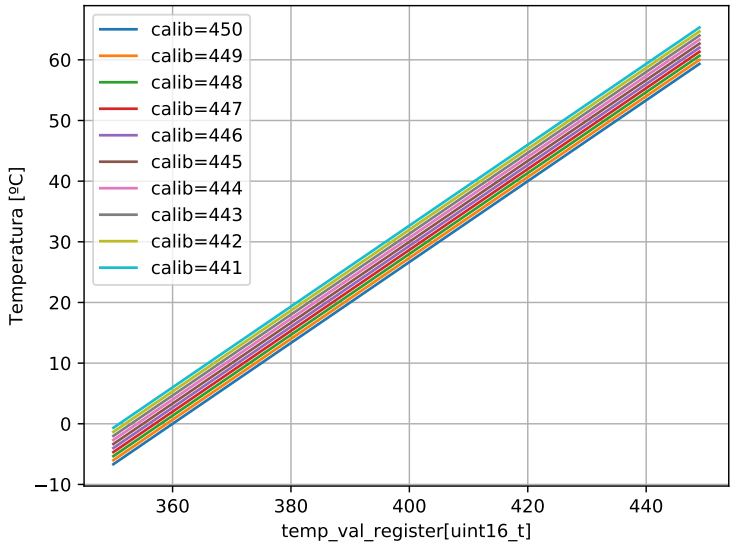
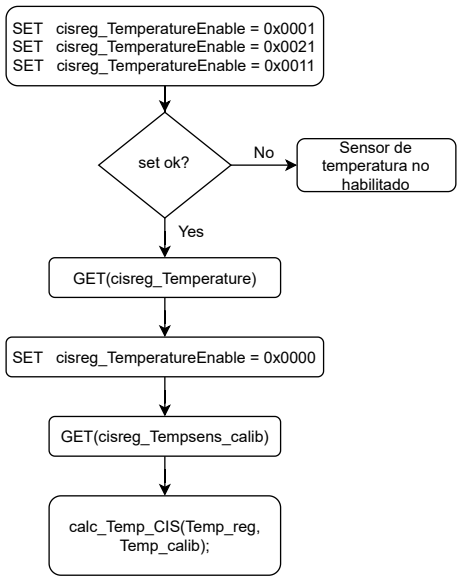


Figura 3.13: Respuesta lineal Temperatura

$$Temperature = Temp\_reg \cdot 0{,}662 + (90 - Temp\_calib)/1{,}51 \tag{3.7}$$

El flujo de acciones a realizar a nivel firmware, viene determinado por el siguiente diagrama:



3.3.4.1. Direcciones y valores permitidos

Se deberá acceder a las siguientes direcciones del mapa de registros del CX3:

Motivo de uso	Acción	Valor	Dirección
Temperature	Lectura	Grados Celsius	0x1009

Tabla 3.8: Dirección CIS Temperature

3.3.5. Bayer Pattern

Al habilitar la opción de cambiar la orientación de la imagen del CIS, será necesario obtener la información de su patrón Bayer correspondiente, ya que al tratarse de un filtro hardware, implicará que a cada píxel del sensor se le asignará siempre el mismo canal de filtrado dentro del patrón RGB. Cambiar la orientación del *frame*, no es más que modificar el orden en que se introducen los píxeles en el buffer de imagen. Por consiguiente, como cada píxel posee, sea cual sea la orientación del sensor, el mismo canal RGB, implicará que el patrón Bayer muestre una distribución distinta en cada una de las 4 configuraciones posibles. Esto mismo se representa en la siguiente figura, en la que se muestran las diferentes configuraciones del *Bayer Pattern*, en función de la orientación seleccionada en el registro de **Image Orientation**:

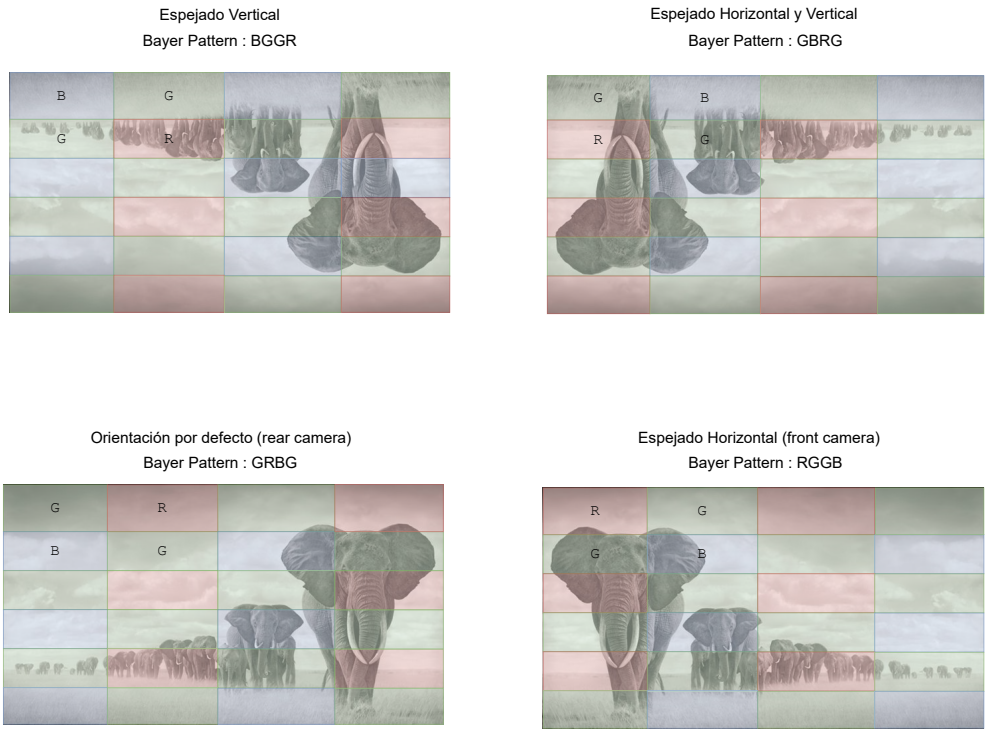


Figura 3.14: Configueraciones Bayer Pattern

Valor	orientación
0	GRBG
1	RGGB
2	BGGR
3	GBRG

Tabla 3.9: Patrones Bayer

3.3.5.1. Dirección y valores permitidos

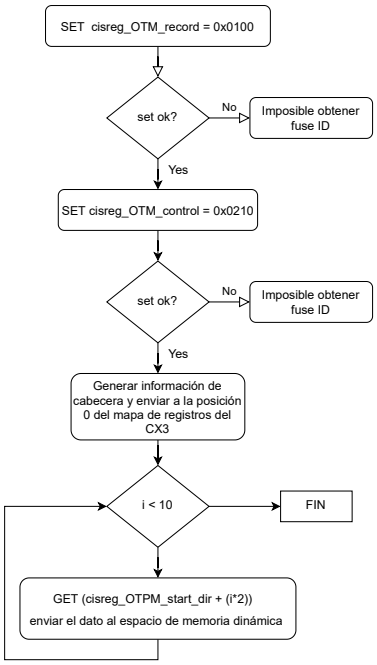
Se deberá acceder a las siguientes direcciones del mapa de registros del CX3, para obtener el patrón Bayer a aplicar en la capa software:

Motivo de uso	Acción	Valor	Dirección
Bayer Pattern	Lectura	[0to3]	0x100A

Tabla 3.10: Dirección Bayer Pattern

3.3.6. Enable FuseID

Al ajustar el registro comenzará la transacción del fuseID del CIS al mapa de registros del CX3, correspondiente al área destinada a almacenamiento dinámico. El flujo implementado en la MCU es el siguiente:



3.3.6.1. Dirección y valores permitidos

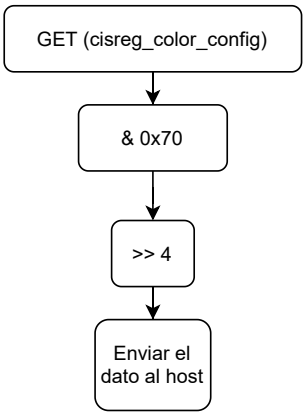
Recordar que para disparar la obtención del FuseID desde el *host*, únicamente será necesario activar el bit menos significativo del registro.

Motivo de uso	Acción	Valor	Dirección
Enable FuseID	Escritura	0 or 1	0x1101

Tabla 3.11: Dirección Enable FuseID

3.3.7. Color Config

El presente registro informará si el CIS es monocromático o color, con la siguiente lógica implementada en el CX3:



Valor	configuración
1	Sensor Color
2	Sensor Monocromático

Tabla 3.12: Configuración de color

Desde el lado de la aplicación software que se ejecuta en el *host* es importante tener en cuenta el tipo de sensor con el que se va a trabajar, ya que de tratarse de un sensor color, implicará aplicar algoritmos específicos, como por ejemplo el ***Demosaicing*** para generar el color en la imagen recibida.

3.3.7.1. Dirección y valores permitidos

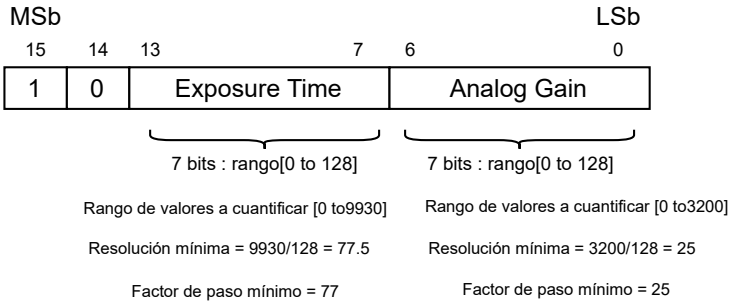
Se deberá acceder a las siguientes direcciones del mapa de registros del CX3:

Motivo de uso	Acción	Valor	Dirección
Color Config	Lectura	1 or 2	0x1102

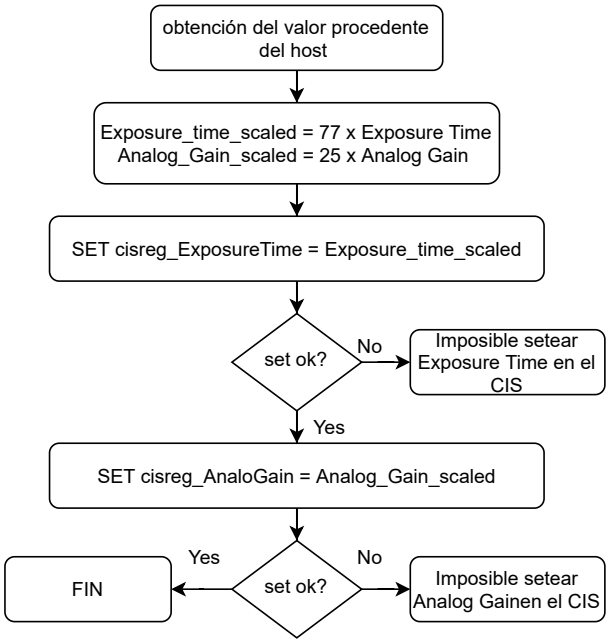
Tabla 3.13: Dirección Color Config

3.3.8. MCU Fast Setting Access 01 Register

Tal y como se explicó en apartados anteriores, este registro permitirá setear los valores de **Exposure Time y Analog Gain**, en la misma transacción UVC, utilizando la siguiente interpretación binaria:



Desde el lado *host* se podrá codificar cada propiedad con un total de 7 bits, suponiendo un rango de representación de [0 a 128]. De manera que, para poder abarcar el espectro de valores estipulados en los apartados **3.3.1 y 3.3.2**, pero contando con un menor cantidad de bits de representación, será necesario aplicar factores de escala. La lógica implementada en el CX3 es la siguiente:



3.3.8.1. Dirección y valores permitidos

Al realizar la transacción a través de una única propiedad UVC, el valor de ambos controles vendrá codificado en el propio registro de dirección:

Motivo de uso	Acción	Valor = Dirección
fast_setting_access	Escritura	0B10.bbbbbbb.bbbbbbb

Tabla 3.14: Dirección Fast Setting Access 01

3.3.9. Perfiles Disponibles

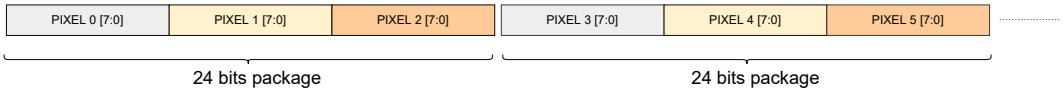
Actualmente el firmware para el ON-Semi CIS , cuenta con los siguientes perfiles de visualización:

Perfil	Filas	Columnas	Pixel bits	Frame Size(bits)
Full Frame 8	3120	4212	8	105131520
Full Frame 8 DPCM	3120	4212	8	105131520
Full Frame 10	3120	4224	10	131788800

Tabla 3.15: Perfiles Disponibles

3.3.9.1. Recepción de 8 bits por píxel a través de RGB888

El CX3 envía la información del *frame* en formato RGB888, utilizando para ello, paquetes de 24 bits. Por consiguiente, en aquellos perfiles que cuenten con 8 bits por píxel, implicará el envío de 3 píxeles en cada transacción, siguiendo la siguiente interpretación binaria:



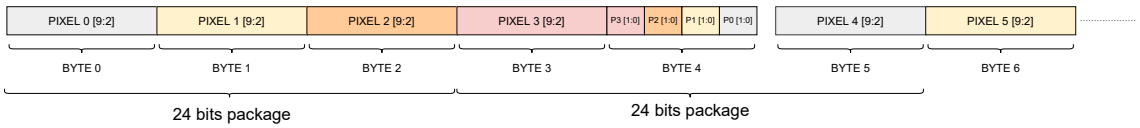
Por lo tanto, el tamaño del perfil que interpretará el *host* a través del estandar UVC es el siguiente:

$$COLUMNAS_{PERFIL} = \frac{n_{bits} \cdot columnas_{cis}}{CX3RGB888_{package}} = \frac{8 \cdot 4212}{24} = 1404$$

$$FrameSize = H \times W \times n_{bits} = 3120 \times 1404 \times 8 \tag{3.8}$$

3.3.9.2. Recepción de 10 bits por píxel a través de RGB888

La principal premisa que se deberá tener en cuenta desde el lado *host*, para poder decodificar la información proveniente del CIS en formato RGB888 con 10 bits por pixel, es que se codificará en 5 Bytes la información de 4 píxeles, siguiendo la siguiente interpretación binaria:



Los 4 primeros bytes codificarán la parte alta de los 4 píxeles consecutivos, empleando el último byte para codificar los dos bits menos significativos de cada píxel. Atendiendo al *bitwise* de este último byte, los dos bits más significativos pertenecerán al último pixel enviado, mientras que los dos bits menos significativos pertenecerán al primer pixel enviado en el paquete, tal y como se representan en la figura anterior.

De manera que el tamaño del perfil que interpretará el *host* a través del estandar UVC es el siguiente:

$$COLUMNAS_{PERFIL} = \frac{n_{bits} \cdot columnas_{cis}}{CX3RGB888_{package}} = \frac{10 \cdot 4224}{24} = 1760$$

$$FrameSize = H \times W \times n_{bits} = 3120 \times 1760 \times 10 \tag{3.9}$$

Capítulo 4

Desarrollo de Aplicación Software

Este punto del proyecto se centrará en desarrollar una aplicación software desde el lado del dispositivo *host*, que permita realizar un *debug* funcional del BridgeCX3 y su firmware correspondiente. La siguiente figura muestra con mayor grado de detalle la conexión entre los diferentes agentes que permitirán el *streaming* de video:

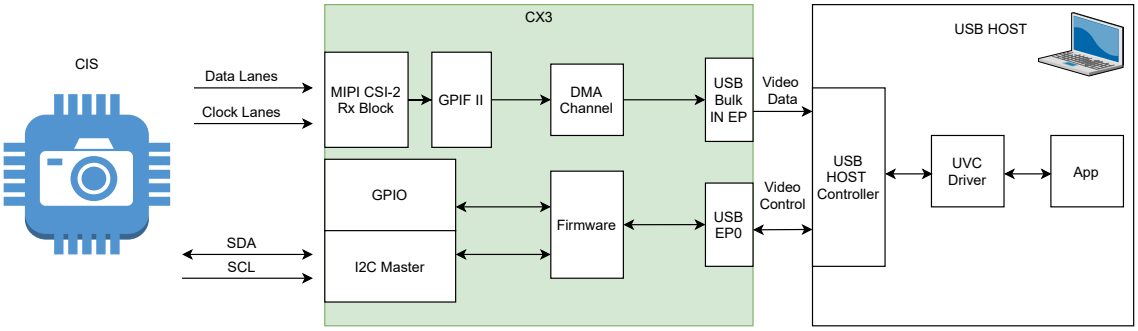


Figura 4.1: Conexión CIS-CX3-HOST

La presente figura permite explicar una serie de conceptos que dan potencial al producto. En primer lugar, el BridgeCX3 es un dispositivo basado en UVC1.1, por lo que no será necesario crear un driver *custom* para que el PC lo reconozca.

El hecho de que el producto se comunique bajo un driver estándar hace que sea posible implementar la aplicación en multitud de entornos de desarrollo distintos. De entre las diferentes opciones disponibles, se decide hacer uso de la libería **OpenCV** sobre un proyecto en Visual Studio. Con esta opción, se conseguirá aprovechar la potencia de un lenguaje compilado, frente al rendimiento que alcanzaría la misma aplicación sobre un lenguaje interpretado.

OpenCV implementa una serie de *backends*, como por ejemplo **DirectShow** o Microsoft Media Foundation **(MSMF)**, los cuales están basados en UVC. Esto soluciona mucho las cosas a la hora de realizar tanto la comunicación con el CIS, como el control de sus parametros haciendo uso de la API desarrollada en el ***capítulo 3***.

4.1. Diagrama de Actores

Explicado lo anterior, se puede hacer una representación visual de los diferentes participantes que intervienen en la comunicación CIS-HOST:

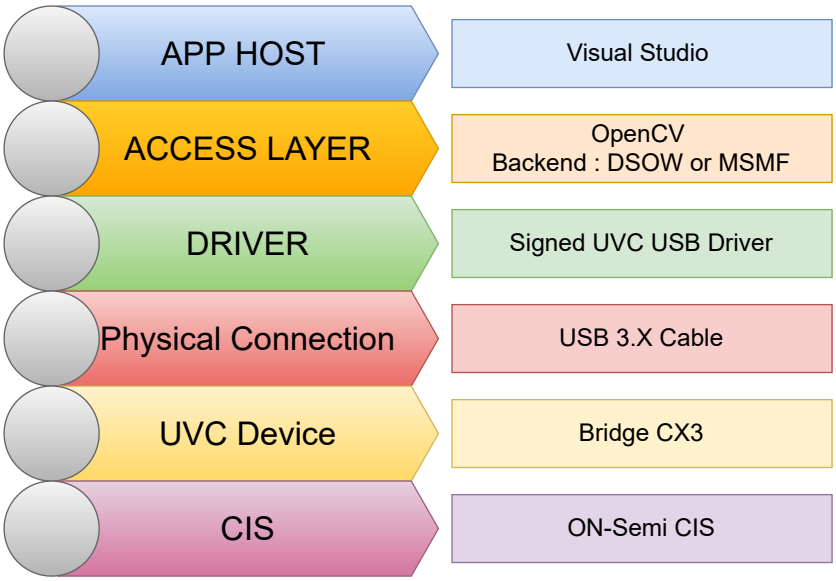


Figura 4.2: Capas de la comunicación

El verdadero **potencial del BridgeCX3** reside en que **será posible transmitir el *data frame* de cualquier CIS que siga el estándar MIPI, a cualquier dispositivo cuyo sistema operativo soporte el driver UVC**.

4.2. Estructura del algoritmo

Como ya se ha explicado en anteriores puntos del proyecto, el desarrollo de la aplicación software se centra en concebir **una herramienta de testing y debug para el BridgeCX3**. Es por esto que se deberán cubrir las siguientes funcionalidades:

- En primer lugar será fundamental obtener un *streaming* de video en cualquiera de los 3 perfiles que contempla el BridgeCX3.
- Poder controlar, en tiempo real, los parámetros fundamentales del CIS, haciendo uso del API que permite la comunicación con el mapa de registros del CX3.
- Poder calcular el **FPS**(frames por segundo), ya que se trata de un parámetro primordial para verificar la optimización de velocidad del dispositivo, explicado con mayor grado de detalle en el ***capítulo 5***.

Por consiguiente la estructura del algoritmo queda de la siguiente manera:

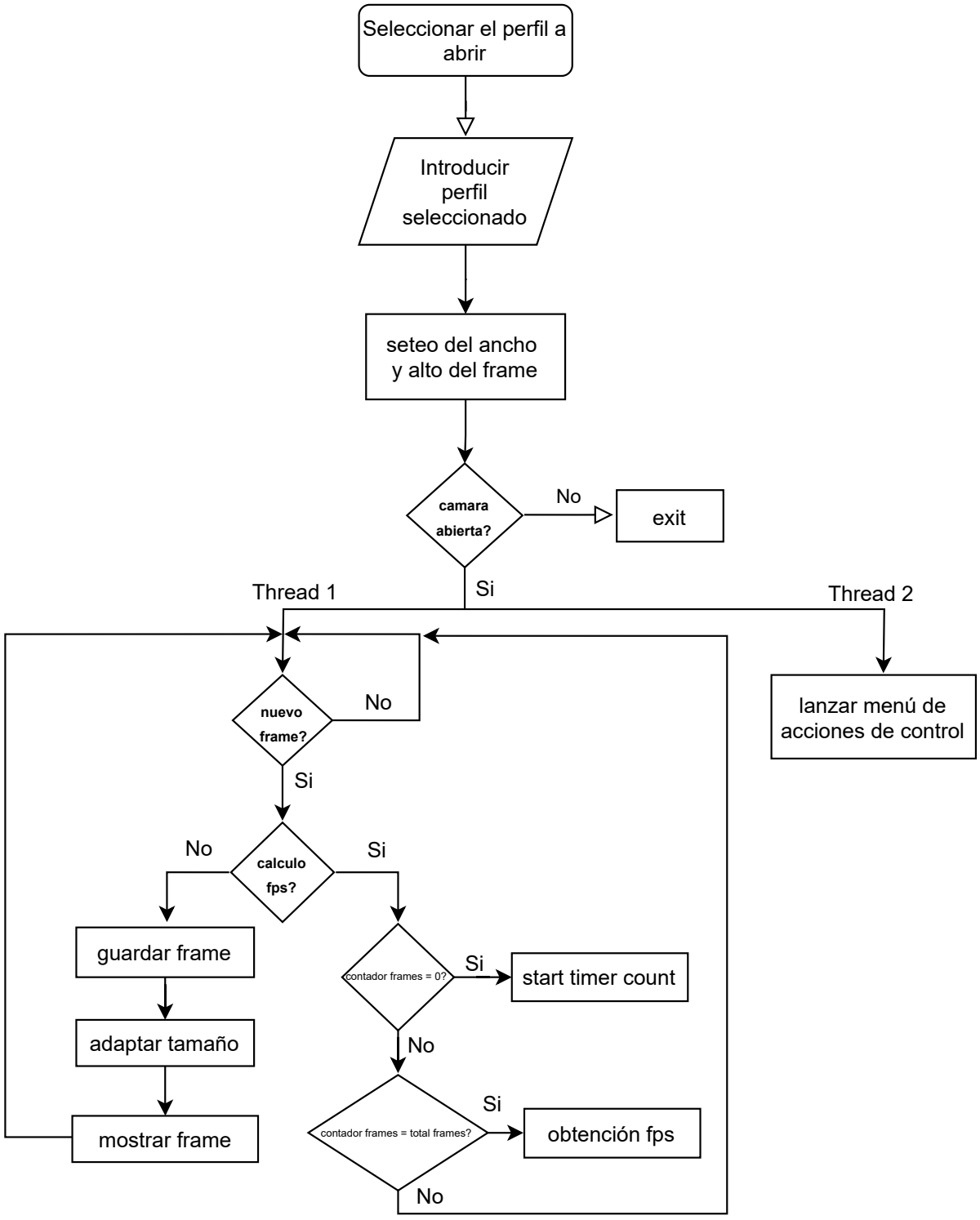


Figura 4.3: diagrama de flujo aplicación software

4.2.1. Comunicación con BridgeCX3

Para poder capturar y mostrar el *streaming* de la cámara, OpenCV proporciona una serie de librerías bastante amigables. En primer lugar, será necesario crear un objeto de la clase **VideoCapture()**, en donde su argumento se corresponde con el índice con el que el sistema operativo reconoce al BridgeCX3 como *camera device*. En caso de no seleccionar correctamente el índice, se corre el riesgo de abrir cualquier otra cámara que en ese momento esté conectada al PC bajo el driver UVC, como por ejemplo, la propia *webcam*.

Posteriormente, se hará uso de los diferentes métodos que proporciona la clase VideoCapture() y que serán necesarios para implementar la lógica de detección de cámara abierta, lectura de un *frame*, visualización del *frame* etc. En la aplicación se hace uso de los siguientes: **isOpened()**,**read()** o **imshow()**.

4.2.2. Implementación del menú

El menú de la aplicación servirá para que el usuario pueda manipular los diferentes parámetros de control del CIS, contemplados el mapa de registros interno del CX3, estructurado en el ***apartado 3.3***. Como premisa de diseño, hay que tener en cuenta que, para que las modificaciones sobre los controles de la cámara ejerzan su efecto, deberán setearse en tiempo real durante el *streaming* de video.

Esto conlleva la necesidad de realizar una aplicación basada en hilos, en la que el hilo principal, que no es más que la propia función *main*, se encargue de capturar y mostrar los *frames* provenientes del BridgeCX3. El hilo secundario implementará una **máquina de estados finitos**, que permitará mostrar por pantalla al usuario el menú y gestionar el conjunto de acciones de control al CIS.

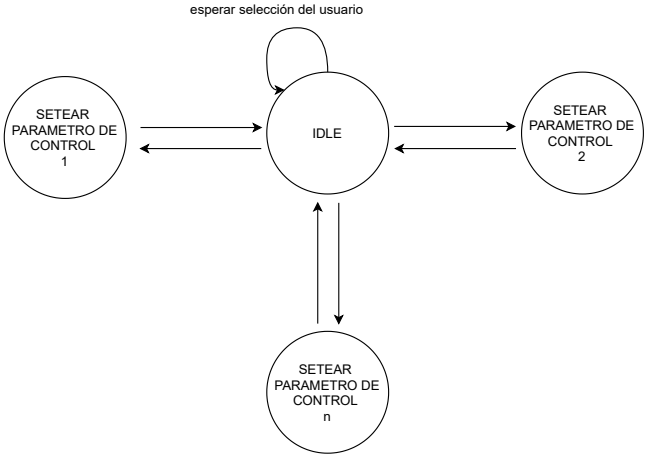


Figura 4.4: Máquina de estados finitos del menú

Con la estructura mostrada en la imagen, el usuario podrá modificar los ***N* parámetros de control**. Internamente, el algoritmo manejará las propiedades UVC **Saturation**, **Contrast** y **Sharpness** a través de los métodos **set() y get()**, siguiendo el *handshake* mostrado en **los apartados *3.2.3* y *3.2.4***.

Capítulo 5

Optimización del Data Rate

En este último apartado se abordará un aspecto fundamental relacionado con el rendimiento del dispositivo, que más allá de la mera funcionalidad, hará que el producto tenga un mayor potencial.

Hasta ahora, se ha logrado un correcto funcionamiento en lo referente tanto a diseño hardware, como a diseño firmware,consiguiendo traspasar los *frames* de apiCAM al PC, utilizando USB3.0 a través de un driver estándar.

Pero una vez logrado lo anterior, será primordial hacer que el dispositivo trabaje a pleno rendimiento, lo que se traduce en poder transferir el mayor número de *frames* por segundo posible, dotando de una mayor ventaja competitiva al producto final.

Para lograrlo, será necesario entender la estructura de registros que manipulan los PLL tanto del sensor como del CX3. A fin de cuentas, serán estos PLL los que lograrán incrementar las frecuencias de los clocks internos, encargados de capturar y enviar los datos de imagen en el CIS, y de recibirlas y enviarlas al *host* en el CX3.

Esta gestión del ***data rate*** será independiente en los dos actores principales, pero únicamente se conseguirá una aplicación funcional, si se logra alcanzar el **máximo sincronismo posible entre ellos**. Es decir, que no podrá existir una holgada diferencia entre ambos *throughputs*.

5.1. Customización data-rate del CIS

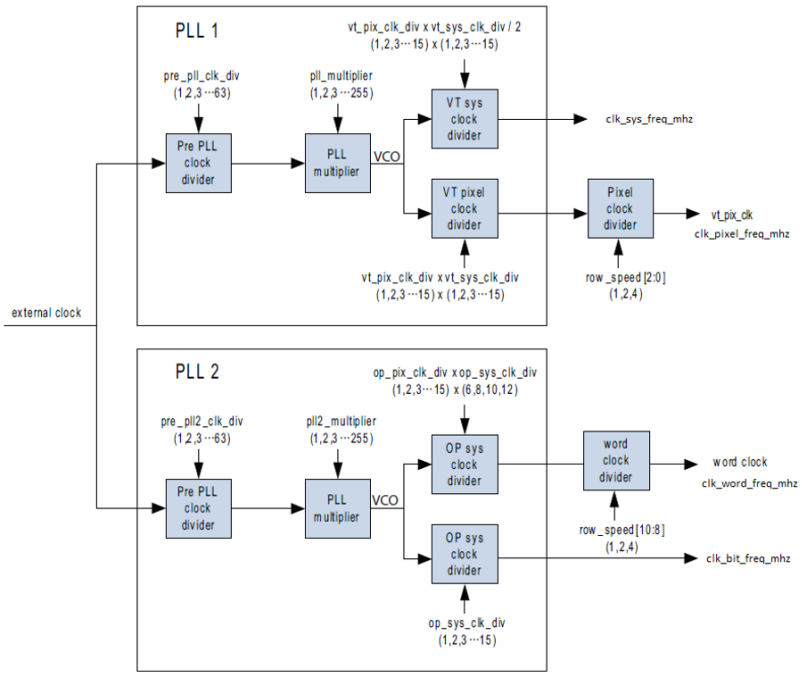


Figura 5.1: Estructura PLL del CIS

Observando el diagrama de bloques, el CIS cuenta con 2 PLL que suministran 4 señales de clock para diversos fines. La frecuencia de cada señal vendrá determinada por los coeficientes de ponderación de los diferentes bloques mostrados en azul.

Será la señal **clk_pixel_freq_mhz** la que dará pie al clock del sistema, convirtiéndose en la señal de interés, sobre la que centrar los esfuerzos para poder manipular el *data rate* del CIS de una forma sencilla, siendo su ecuación la siguiente:

$$clk\_pixel\_freq\_mhz = \frac{ext\_clk\_freq\_mhz \cdot PLL\_MULTIPLIER}{pre\_pll\_clk\_div1 \cdot vt\_sys\_clk\_div \cdot vt\_pix\_clk\_div \cdot row\_speed[2{:}0]}$$

La ecuación viene determinada por la frecuencia de clock de entrada al PLL, y de diferentes registros de configuración.

5.1.1. Datos de partida

Lo primero que se necesitará para poder encontrar una relación entre el clk_pixel_freq_mhz y el *data rate*, será obtener sus valores en una aplicación funcional. Para ello, se decide acudir al preset de inicialización de la placa de evaluación de ON Semiconductor.

Esta aplicación funcional hace trabajar al CIS con 4 líneas MIPI a razón de 880 Mbps, deduciéndose el siguiente la siguiente tasa de datos:

$$CIS\_data\_rate = 4 \cdot 880Mbps = 3{,}52Gbps$$

Para lograr este resultado, los registros se configuran con los siguientes valores, asumiendo un clock de entrada de 25 MHz, dando como resultado un **clk_pixel_freq_mhz** de $220MHz$:

PLL_MULTIPILER	0xOB(176)
pre_pll_clk_div1	0x05
Vt_sys_clk_div	0x01
Vt_pix_clk_div	0x04
row_speed[2:0]	0x01

$$clk\_pixel\_freq\_mhz = \frac{25 \cdot 176}{5 \cdot 1 \cdot 4 \cdot 1} = \frac{25 \cdot 176}{20} = 220\ MHz$$

5.1.2. Relación PLL_MULTIPLIER - Data Rate

Para encontrar una ecuación que nos permite determinar el *data rate* del CIS a partir de un único registro del PLL, será necesario mantener el valor fijo del resto de registros y manipular únicamente el registro **PLL_MULTIPLIER**.

La obtención de la relación entre ambas variables se llevará a cabo a través del clk_pixel_freq_mhz, y de los datos de partida calculados en el punto anterior.

- En la configuración de ON-Semiconductor, el *data rate* es conocido. Además, se puede suponer una relación lineal con respecto a clk_pixel_freq_mhz. Con esto se podrá obtener la constante de proporcionalidad entre ambas variables:

$$CIS\_data\_rate = K \cdot clk\_pixel\_freq\_mhz \tag{5.1}$$

$$3{,}52Gbps = K \cdot 220MHz$$

$$K = \frac{3{,}52}{220}$$

- Para terminar de obtener la relación, será necesario acudir a la ecuación que determinaba el clk_pixel_freq_mhz, teniendo en cuenta que el único registro modificable del CIS debe ser el PLL_MULTIPLIER:

$$clk\_pixel\_freq\_mhz = \frac{clk\_in \cdot PLL\_MULTIPLIER}{20} \tag{5.2}$$

Juntando las ecuaciones 5.1 y 5.2 Se obtendrá la relación final:

$$\frac{CIS\_data\_rate}{K} = \frac{clk\_in \cdot PLL\_MULTIPLIER}{20}$$

$$PLL\_MULTIPLIER = \frac{20}{clk\_in \cdot K} \cdot CIS\_data\_rate \tag{5.3}$$

Ajustando el valor del clock de entrada a 19.2 MHz, ya que se trata de la frecuencia de clock que alimenta al CIS en el BridgeCX3, se tendrá la relación completa para poder determinar el valor del *data rate* a partir de la modificación de un solo registro:

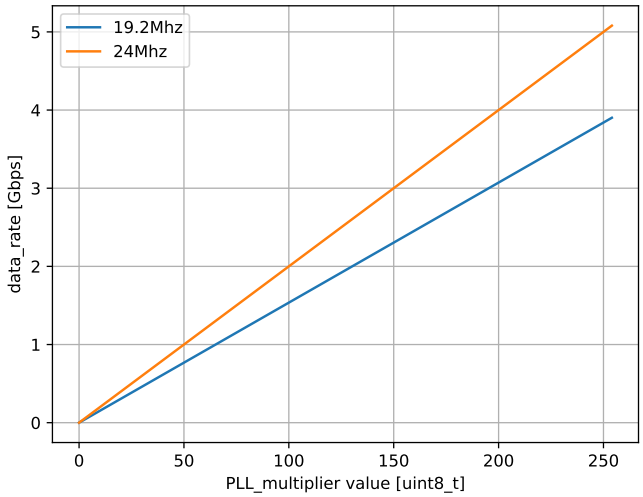


Figura 5.2: relación PLL_multiplier-CIS data rate

5.2. Customización data-rate CX3

De forma análoga a la del CIS, será necesario estudiar la estructura del PLL del CX3 para poder optimizar su *data rate*.

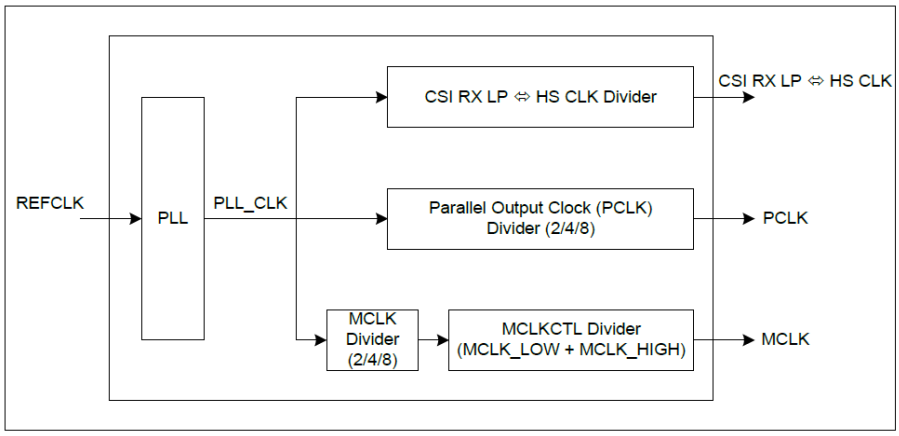


Figura 5.3: Estructura PLL CX3 [4]

Siguiendo el diagrama, se aprecia como a partir del clock de entrada se obtendrá un primer clock denominado **PLL_CLK**, del que posteriormente nacerán las 3 señales de clock internas que el CX3 utilizará para diversos fines.

De los 3 clocks de salida, se deberán centrar los esfuerzos en el **Output Parallel Clock (PCLK)**, siendo el empleado por el CX3 para determinar el flujo de funcionamiento de la interfaz GPIF-II. Recordemos que el *hard-block* GPIF-II es la máquina de estados encargada de recibir los datos del frame en formato MIPI y tratarlos para ser enviados en formato USB al *host*. Por consiguiente, será sencillo entender que el *throughput* del CX3 dependerá directamente del valor de su frecuencia.

La documentación oficial del CX3[4] marca en **100 MHz** la frecuencia máxima alcanzable por el **PCLK**. Sumado a que el CX3 está trabajando con un formato de imagen RGB888 a su salida, significa que se estarán enviando 24 bits por cada flanco de reloj, dando como resultado **un ancho de banda máximo del CX3** de:

$$CX3\_data\_rate = 100MHz \cdot 24bits = 2{,}4\ Gbps$$

Asumiendo que el CIS puede superar el máximo data rate alcanzable por el CX3, y que el ancho de banda máximo del USB3.0 es de 5Gbps, convierte al **CX3 en el cuello de botella**, en lo que a términos de velocidad se refiere. Por lo tanto, **2.4Gbps es la velocidad máxima alcanzable por el dispositivo**.

5.2.1. Ajuste del PCLK en 100MHz

Para poder configurar el PCLK a su valor máximo, primero será necesario ajustar el valor del clock que le precede, es decir, del **PLL_CLK**.

$$PLL\_CLK = REFCLK \cdot \frac{PLL\_FBD + 1}{(PLL\_PRD + 1) \cdot 2^{PLL\_FRS}} \tag{5.4}$$

Donde *PLL_FBD*, *PLL_PRD* y *PLL_FRS* son registros manipulables del CX3. Teniendo el valor del PLL_CLK, obtener el **PCLK**, será tan sencillo como dividirlo por el registro *parClkDiv*, que únicamente puede ajustarse con valores múltiplos de dos {2,4,8}.

$$PCLK = \frac{PLL\_CLK}{parClkDiv} \tag{5.5}$$

Bastará con encontrar una combinación de valores que proporcione un PCLK de 100 Mhz. Por último solo quedará ajustar el valor del **PLL_MULTIPLIER** del CIS para obtener un *data rate* lo más cercano posible a 2.4 Gbps. De esta forma, **el conjunto CIS-CX3 estará sincronizado y trabajando a la máxima tasa de datos posible**.

Combinación óptima de registros para trabajar con un **data rate de 2.4 Gbps**:

Registro	Valor
PLL_FBD	0x7C(124)
PLL_PRD	0x02
PLL_FRS	0x00
parClkDiv	0x08
PLL_MULTIPLIER	0xA0(160)

Capítulo 6

Conclusiones

Finalmente, tras la realización del proyecto se cierran los siguientes puntos expuestos al comienzo del mismo:

- Se ha logrado diseñar un PCB multicapa de reducido tamaño y alta densidad de integración de componentes, capaz de dotar de conectividad USB a un CIS, atendiendo a criterios de diseño térmico, integridad de señal e integridad de potencia. Logrando hacer trabajar al sensor en condiciones óptimas.
- Se ha desarrollado una arquitectura firmware que permite llevar a cabo un control eficiente de los parámetros más relevantes del CIS, haciendo uso de un driver estándar ampliamente utilizado, pero a través de un *handshake* propio que optimiza el control.
- Se ha implementado una aplicación software capaz de establecer conexión con el dispositivo diseñado, sirviendo como herramienta de *debug* para revisar y perfeccionar los aspectos más relevantes de la comunicación CIS-CX3, como son la recepción *frames* en diferentes perfiles de visualización, el funcionamiento de los diferentes controles implementados en el firmware, o la velocidad máxima del dispositivo.
- Se ha conseguido descender a nivel de registros, y obtener una configuración que logra optimizar al máximo las capacidades del hardware diseñado.

6.1. Producto final

La consecución de los puntos anteriores, sumado al esfuerzo del equipo de desarrollo mecánico, dan como resultado un dispositivo de altas prestaciones, capaz de proveer la tecnología de reconstrucción 3D a multitud de sectores comerciales, incluido el industrial.

El producto nace con el nombre de **apiCUBE©**, y se compone del **BridgeCX3**, **Camboard** y **apiCAM©**. Contando con un formato compacto como se muestra en la siguiente imagen:

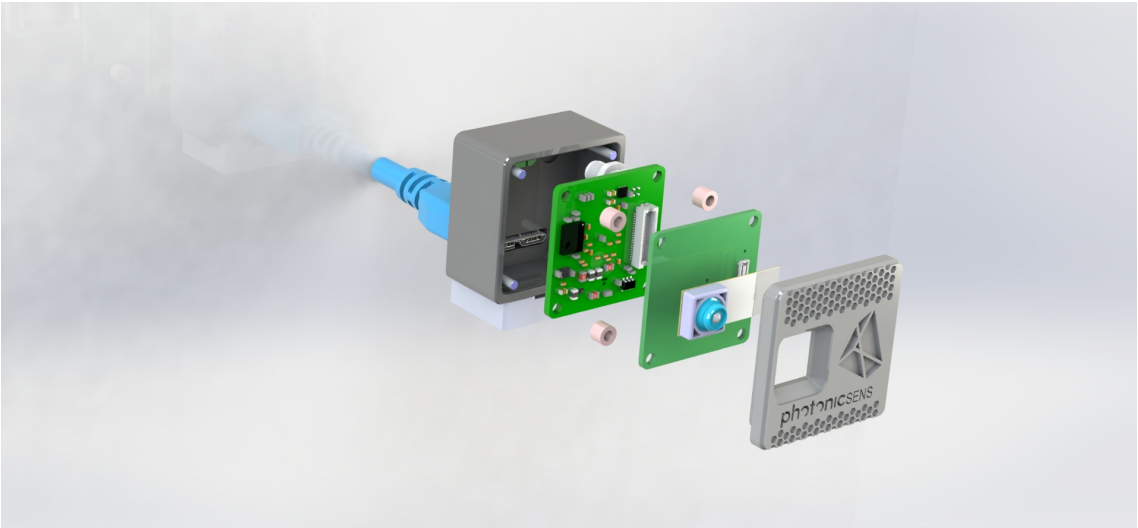


Figura 6.1: apiCUBE© componentes

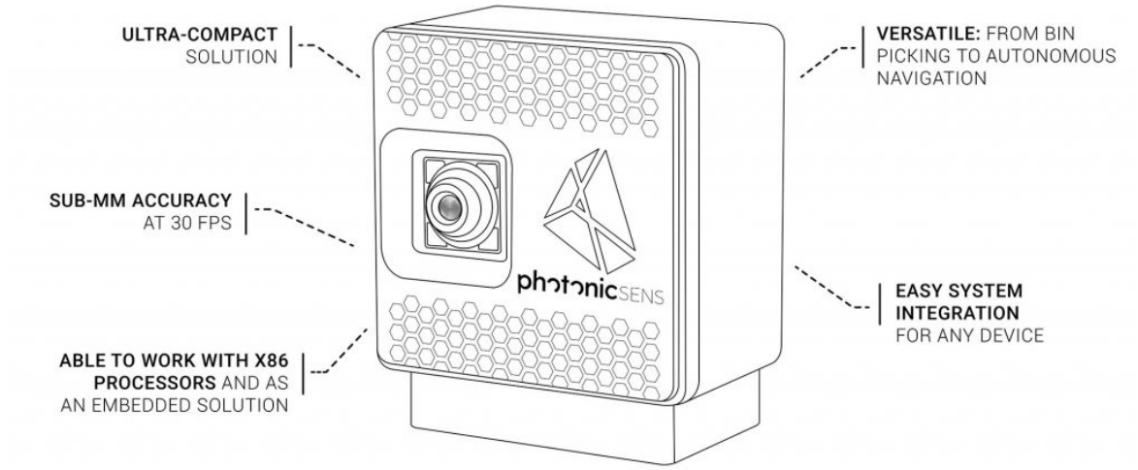


Figura 6.2: apiCUBE© [8]

6.2. Líneas futuras

El coste económico, con tendencia decreciente, de los dispositivos y los sensores. Sumado a la mayor potencia de cálculo que muestran, incluso aquellos de bajo coste. Está empujando al sector hacia un nuevo paradigma de diseño, denominado ***Edge computing***.

Este paradigma se centra en optimizar los procesos de cálculo, evitando tener que comunicar los dispositivos con la nube o con otro hardware más sofisticado, realizando la mayoría del cómputo en el propio dispositivo.

Entre las múltiples ventajas que ofrece, como por ejemplo la disminución de la latencia o la reducción del ancho de banda, destaca la disminución del coste computacional que requeriría el *host* al que se conecte apiCUBE©. Y es que, a día de hoy, la sofisticación de la capa de algoritmos que requiere el producto, necesita una potencia de cálculo elevada, que recae sobre éste.

Por consiguiente, los futuros esfuerzos se centran en la introducción de un dispositivo **FPGA**, capaz no solo de transferir los frames de un CIS, sino dotar a apiCUBE© de lógica programable orientada al procesado de imagen, concibiendo un **ISP custom** que proporcione mayor versatilidad al producto y reduzca al máximo la carga computacional requerida en la aplicación *host*. Lo que sin duda ampliará el espectro de dispositivos capaces de albergar la presente tecnología.

Bibliografía

[1] photonicSENS. “apiCAM features”. En: (2021). URL: https://photonicsens.com/.

[2] photonicSENS. “Development Board for apiCAM”. En: (2021). URL: https://photonicsens.com/.

[3] Infineon. “EZ-USB® CX3: MIPI CSI-2 to SuperSpeed USB Bridge Controller”. En: (2020). URL: https://www.infineon.com/dgdl/Infineon-CYUSB306X_EZ-USB_CX3_MIPI_CSI-2_TO_SUPERSPEED_USB_BRIDGE_CONTROLLER-DataSheet-v16_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ecbbb354559.

[4] Infineon. “EZ-USB CX3 Technical Reference Manual”. En: (2020), pág. 11. URL: https://www.infineon.com/dgdl/Infineon-EZ-USB_CX3_Technical_Reference_Manual-AdditionalTechnicalInformation-v03_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0f908b877d50&utm_source=cypress&utm_medium=referral&utm_campaign=202110_globe_en_all_integration-technical_reference_manual.

[5] Jaejin Lee Ying-Ern Ho Hao-han Hsu. “Optimal decoupling strategy to suppress radio frequency interference (RFI) from double data-rate (DDR) memory power-plane radiation”. En: *IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI)* (2017).

[6] UVC Contributors. “Universal Serial Bus Device Class Definition for Video Devices”. En: (2005). URL: http://www.cajunbot.com/wiki/images/8/85/USB_Video_Class_1.1.pdf.

[7] MIPI Alliance. “DRAFT MIPI Alliance Specification for Camera Serial Interface 2(CSI-2)”. En: (2009), pág. 98. URL: https://pdfcoffee.com/mipi-alliance-specification-for-camera-serial-interface-2-csi-2-pdf-free.html.

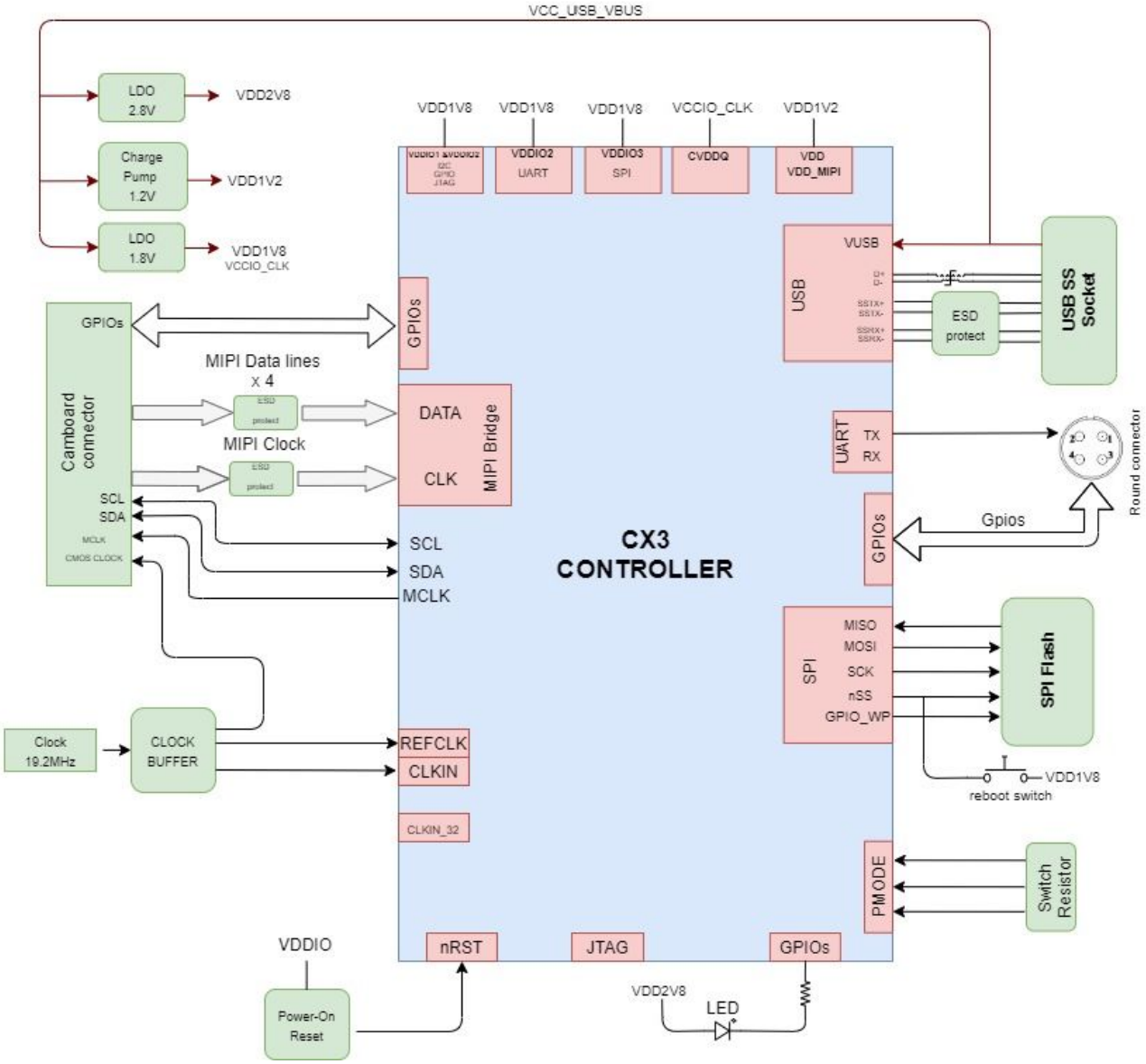
[8] photonicSENS. “apiCUBE features”. En: (2021). URL: https://photonicsens.com/.

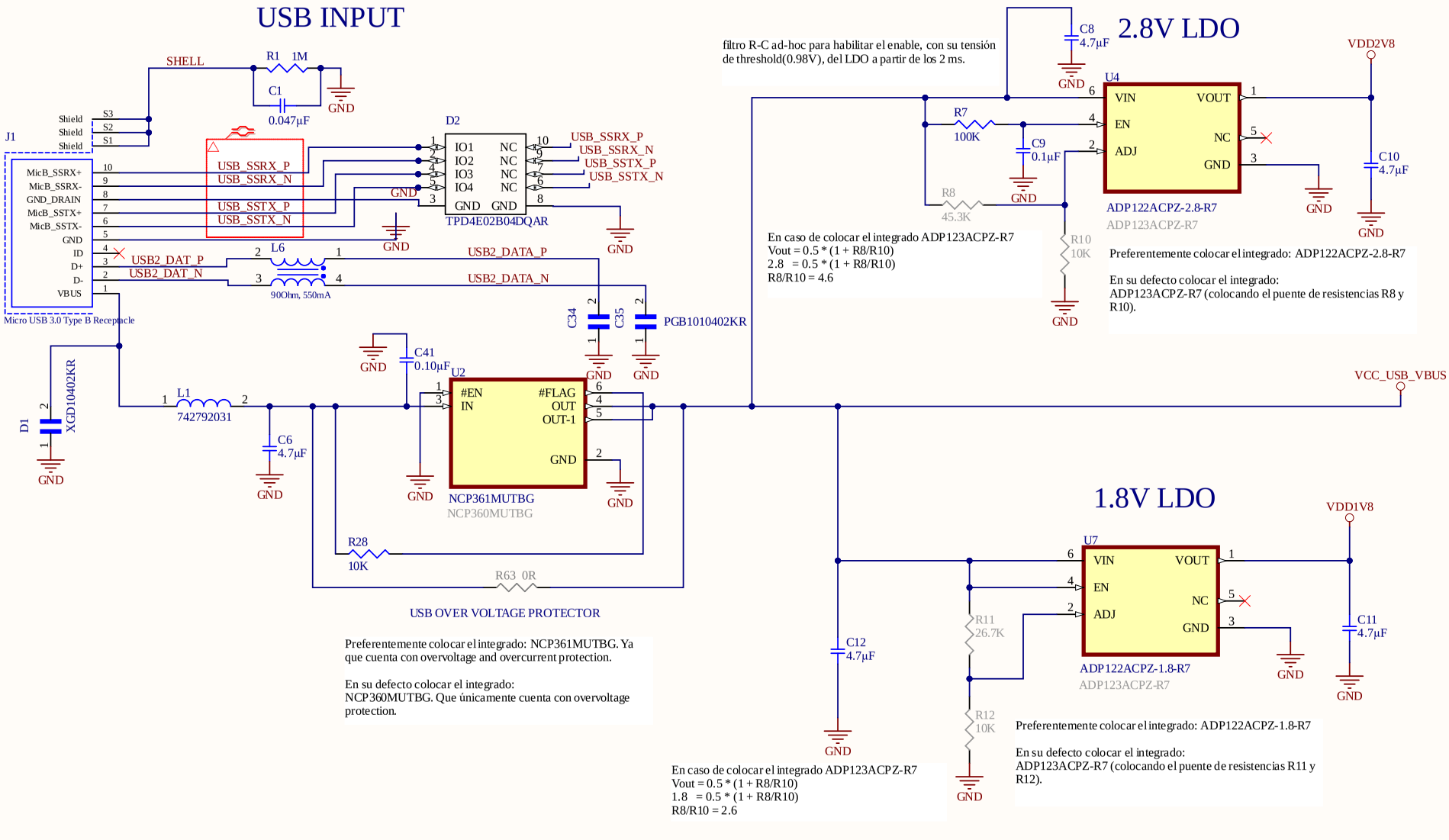
Parte II

Anexos

Apéndice A

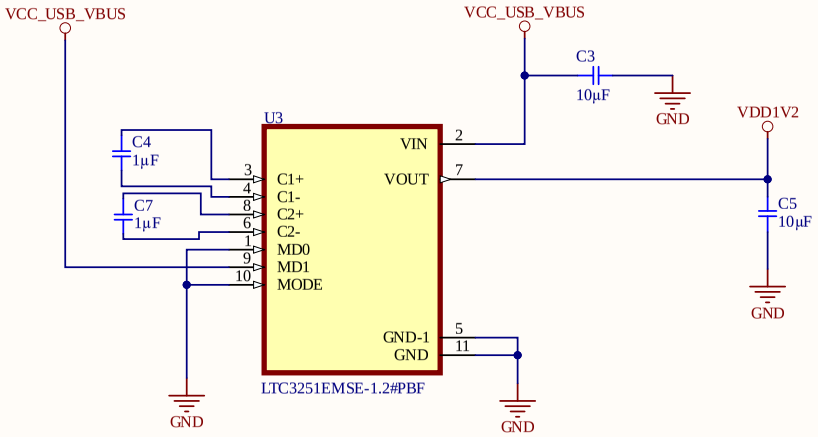
Esquemático Bridge CX3 Board


VCC_USB_VBUS
LDO 2.8V
VDD2V8
Charge Pump 1.2V
VDD1V2
LDO 1.8V
VDD1V8
VCCIO_CLK
VDD1V8
VDD1V8
VDD1V8
VCCIO_CLK
VDD1V2
VDDIO2
UART
VDDIO3
SPI
CVDDQ
VDD
VDD_MIPI
GPIOs
USB
VUSB
ESD protect
USB SS Socket
GPIOs
Camboard connector
MIPI Data lines x 4
MIPI Clock
DATA
CLK
MIPI Bridge
SCL
SDA
MCLK
CMOS CLOCK
CX3 CONTROLLER
UART
TX
RX
Round connector
GPIOs
Gpios
SPI
MISO
MOSI
SCK
nSS
GPIO_WP
SPI Flash
VDD1V8
reboot switch
Clock 19.2MHz
CLOCK BUFFER
REFCLK
CLKIN
CLKIN_32
PMODE
Switch Resistor
VDDIO
Power-On Reset
nRST
JTAG
GPIOs
VDD2V8
LED



BRIDGE CX3 V2.2 POWER		Hardware department Arturo Samper arturo.samper@photonicsens.com	photonicSENS SENSORS & ALGORITHMS
Size: A4	Revision: 1.0		
Date: 9/9/2021	Sheet 1 of 7		

1.2V DC/DC Inductorless



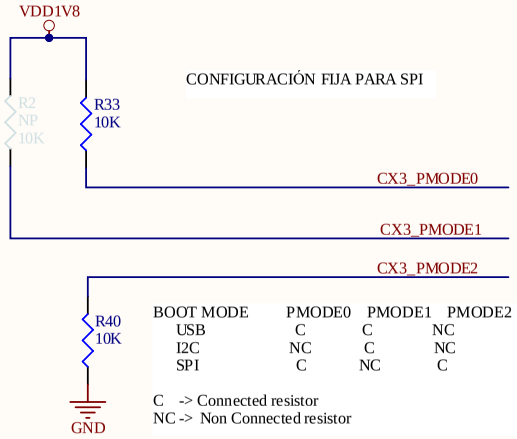
El DC/DC converter deberá operar en Continuous Spread Spectrum.

MD0 -> 0

MD1 -> 1

Se deberá trabajar en Spread Spectrum (MODE pin = 0). Que se bása en añadir Jitter al switching clock con el fin de introducir ruido esparcido por todo el espectro. Y no tener una fundamental de ruido a la frecuencia de switching.

BOOT MODE



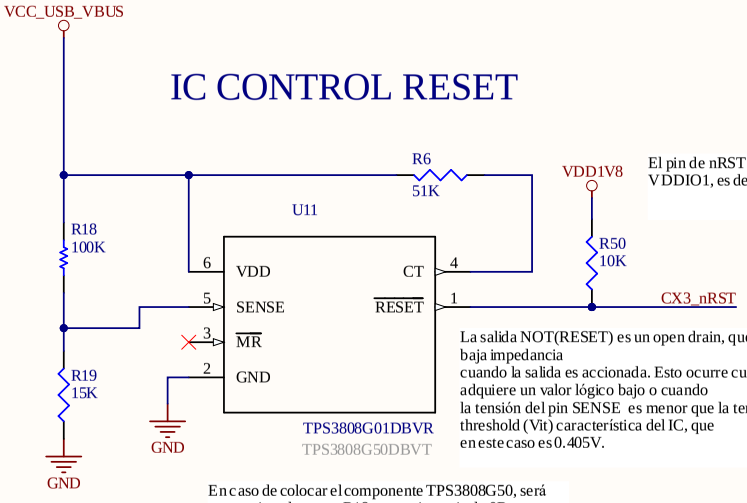
CONFIGURACIÓN FIJA PARA SPI

BOOT MODE	PMODE0	PMODE1	PMODE2
USB	C	C	NC
I2C	NC	C	NC
SPI	C	NC	C

C -> Connected resistor

NC -> Non Connected resistor

IC CONTROL RESET



El pin de nRST del CX3 trabaja en el dominio de VDDIO1, es decir, en nuestro caso a 1.8V.

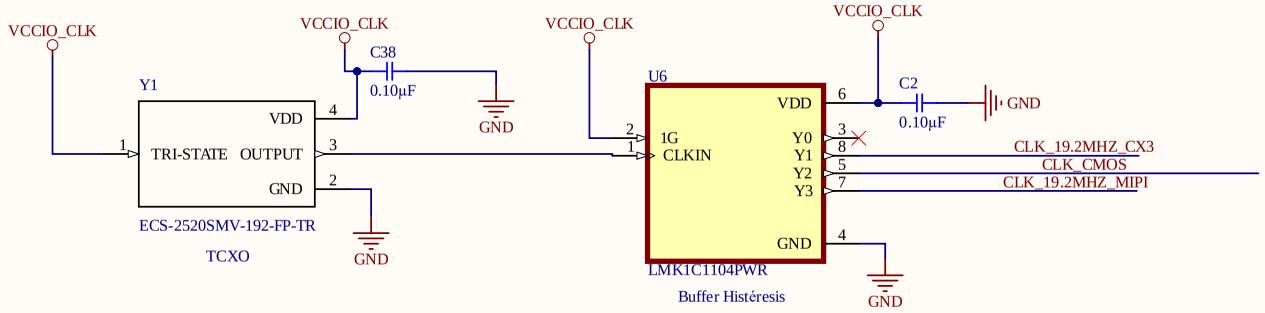
La salida NOT(RESET) es un open drain, que se pone a baja impedancia cuando la salida es accionada. Esto ocurre cuando MR adquiere un valor lógico bajo o cuando la tensión del pin SENSE es menor que la tensión de threshold (Vit) característica del IC, que en este caso es 0.405V.

En caso de colocar el componente TPS3808G50, será necesario colocar en R18 una resistencia de 0R (Jumper) y no se deberá colocar la resistencia R19.

BRIDGE CX3 V2.2 POWER2		Hardware department Arturo Samper arturo.samper@photonicsens.com	photonicSENS SENSORS & ALGORITHMS
Size: A4	Revision: 1.0		
Date: 9/9/2021	Sheet 2 of 7		

OSCILADOR DE 19.2 MHz

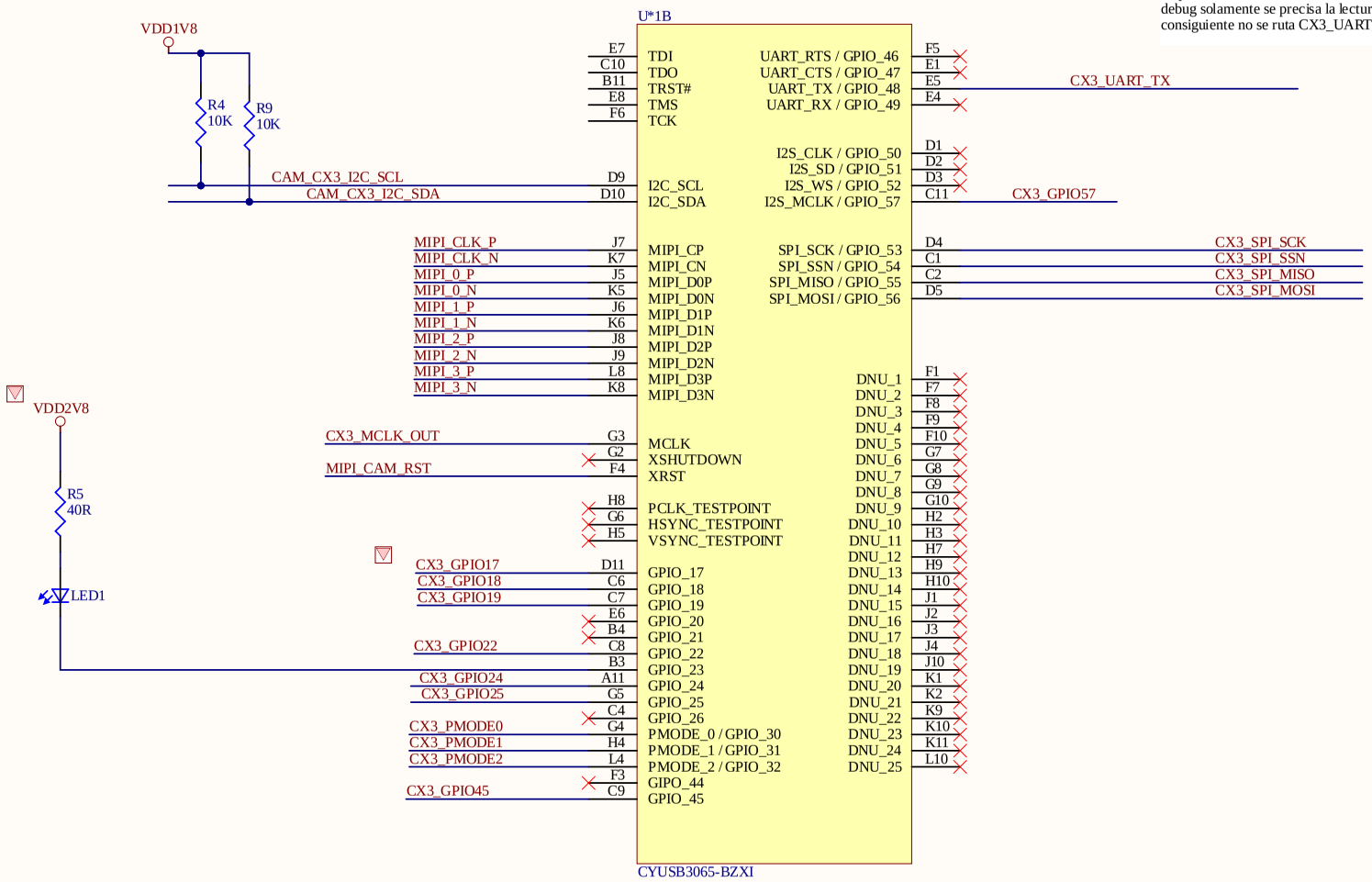
CLOCK BUFFER



BRIDGE CX3 V2.2 CLOCK GENERATION		Hardware department Arturo Samper arturo.samper@photonicsens.com	photonicSENS SENSORS & ALGORITHMS
Size: A4	Revision: 1.0		
Date: 12/19/2021	Sheet 3 of 7		

MCU

Se procede a sacar únicamente la señal CX3_UART_TX, ya que para hacer debug solamente se precisa la lectura y no el envio de datos al CX3. Por consiguiente no se ruta CX3_UART_RX.



BRIDGE CX3 V2.2		Hardware department
CX3 CONTROLLER		Arturo Samper
Size: A4	Revision: 1.0	arturo.samper@photonicsens.com
Date: 9/9/2021	Sheet 4 of 7	photonicSENS

**********POWER SUPPLY DOMAINS **********

VDDIO1_1 & VDDIO1_2 -> Tensiones de trabajo para el bus I2C, GPIO, JTAG.
También para las señales que comandan el CIS : XRST, XSHUTDOWN.
Y también para el nivel de tensión del clock que saca el CX3: MCLK
Como el CMOS habla I2C a 1.8V, estos pines estan alimentadios
a 1.8V y evitamos colocar el I2C Buffer.

VDDIO2 -> tensión de trabajo para UART, 1.8V

VDDIO3 -> tensión de trabajo para el SPI. 1.8V

VDDQ : -> Tension de trabajo de los relojes, 1.8V):
- CLKIN -> clock de 19.2 MHz del clock buffer
- CLKIN_32 - > Real time clock (se elimina).

VDD : tensión de alimentación de la lógica del core CX3. Se debe colocar a 1.2V.
A_vDD = tensión de alimentación para el PLL, y otros circuitos analógicos internos del CX3.
U3TXVDDQ/U3RXVDDQ : determina el nivel de tensión para la interfaz USB 3.0.

VUSB: Tensión de alimentación del sistema. Marcada en 5V (proveniente del PC a través del conector USB).
El CX3 tiene reguladores de voltaje internos que regulan dicha tensión marcandola en 3.3V.
****

MCU

U*1A — CYUSB3065-BZXI

VDD1V2: B10 VDD1, C3 VDD2, E9 VDD3, F11 VDD4, H1 VDD5, J11 VDD6, L7 VDD7, L5 VDD_MIPI

VUSB: E10 ← VCC_USB_VBUS; C15 0.10µF to GND

DM: A10 — USB2_DATA_N

DP: A9 — USB2_DATA_P

SSTXM: A6 — C39 0.10µF — USB_SSTX_N

SSTXP: A5 — C40 0.10µF — USB_SSTX_P

SSRXP: A4 — USB_SSRX_N

SSRXM: A3 — USB_SSRX_P

CLKIN: D7 — CLK_19.2MHZ_MIPI

REFCLK: F2 — CLK_19.2MHZ_CX3

RESET#: C5 — CX3_nRST

MIPIRESET: H6

CLKIN_32: D6 — R49 10K to GND

Intercambiamos la señal RX del USB3.0 porque facilita el ruteo y se puede (segun documentacion de FX3 y Especificaciones del USB3.0)

VDD1V2 → A7 AVDD; C33 0.10µF; B7 AVSS → GND

VDD1V2 → L3 MPZ2012S601AT000 → U3RXVDDQ: A2 U3RXVDDQ; C18 22µF; C19 0.10µF → GND

U3TXVDDQ: B5 U3TXVDDQ; A1 U3VSSQ → GND

VDD1V2 → L4 MPZ2012S601AT000 → U3TXVDDQ; C20 22µF; C21 0.10µF → GND

VDD1V8 → H11 VDDIO1_1; L9 VDDIO1_2

VDD1V8 → C45 0.10µF; C46 0.10µF; C47 10000pF → GND

Colocar cerca de los pines H11 y L9

VDD1V8 → E3 VDDIO2; C42 0.10µF → GND

VDD1V8 → B1 VDDIO3; C43 0.10µF → GND

B6 CVDDQ

VDD1V8 → L2 742792640 (DC Res: 0.35 ohm, Z@20MHz: 45 ohm) → VCCIO_CLK; C16 4.7µF; C77 0.10µF → GND

VDD1V2 → C23 0.10µF; C24 10000pF; C25 0.10µF; C26 10000pF; C27 0.10µF; C28 10000pF; C14 10000pF; C29 0.10µF; C30 10000pF; C31 0.10µF → GND

Colocar cerca de los pines F11, H1, L7

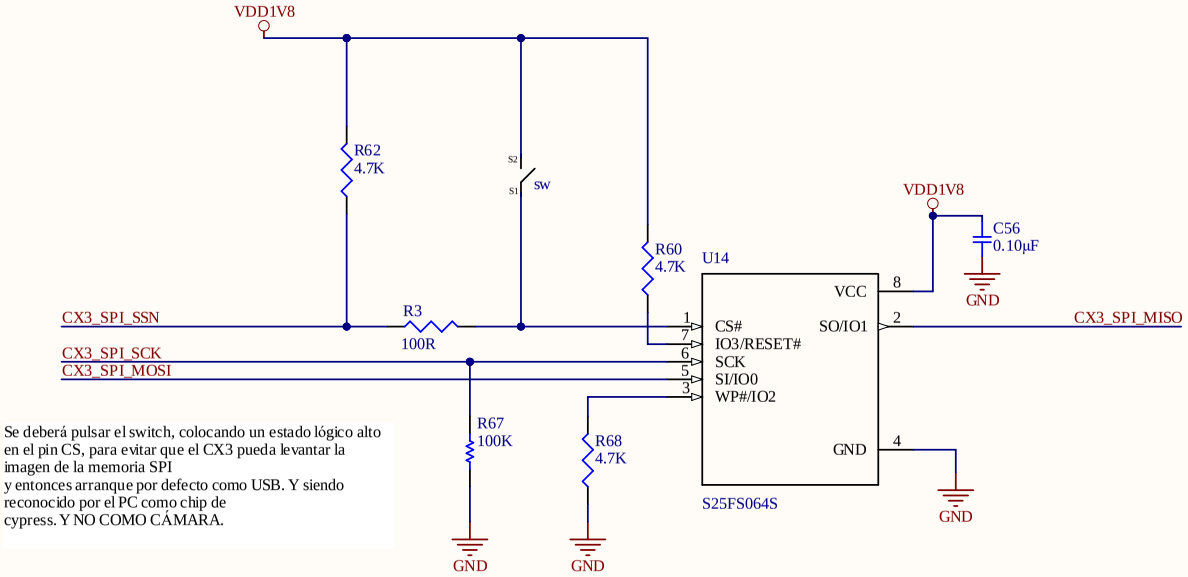
VSS1 A8, VSS2 B2, VSS3 B8, VSS4 B9, VSS5 D8, VSS6 E2, VSS7 E11, VSS8 G1, VSS9 G11, VSS10 K3, VSS11 K4, VSS12 L1, VSS13 L2, VSS14 L3, VSS15 L6, VSS16 L11 → GND

BRIDGE CX3 V2.2	
CX3 USB INTERFACE	
Size: A4	Revision: 1.0
Date: 9/9/2021	Sheet 5 of 7

Hardware department
Arturo Samper
arturo.samper@photonicsens.com

photonicSENS
SENSORS & ALGORITHMS

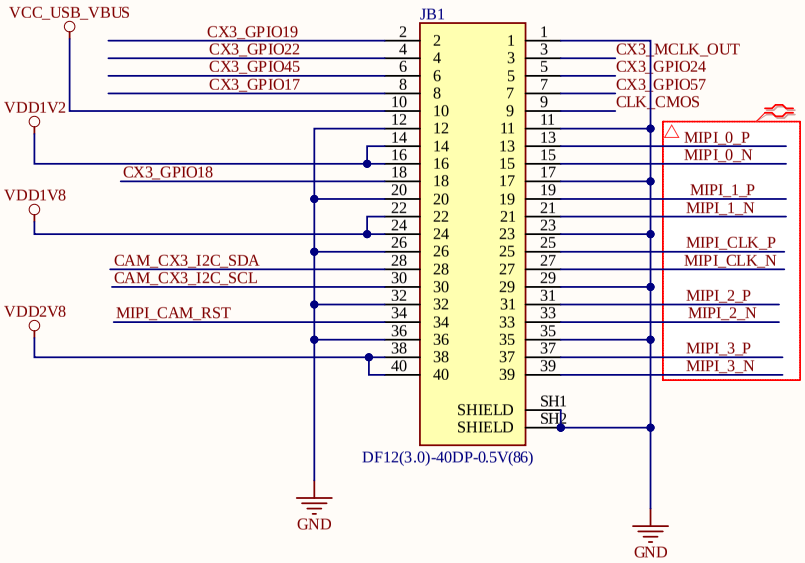
SPI FLASH MEMORY



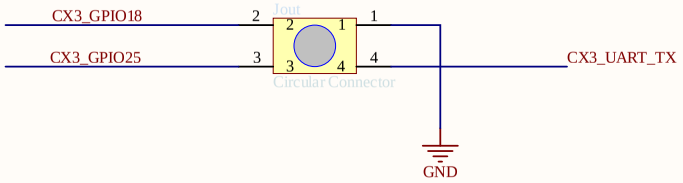
Se deberá pulsar el switch, colocando un estado lógico alto en el pin CS, para evitar que el CX3 pueda levantar la imagen de la memoria SPI
y entonces arranque por defecto como USB. Y siendo reconocido por el PC como chip de cypress. Y NO COMO CÁMARA.

BRIDGE CX3 V2.2 SPI FLASH		Hardware department Arturo Samper arturo.samper@photonicsens.com	photonicSENS SENSORS & ALGORITHMS
Size: A4	Revision: 1.0		
Date: 9/9/2021	Sheet 6 of 7		

BOARD2BOARD CONNECTOR



ROUND CONNECTOR



BRIDGE CX3 V2.2 BOARD2BOARD CONNECTOR		Hardware department Arturo Samper arturo.samper@photonicsens.com
Size: A4	Revision: 1.0	
Date: 9/9/2021	Sheet 7 of 7	

photonicSENS
SENSORS & ALGORITHMS

Apéndice B

Versionado del Firmware

Al tratarse de una arquitectura firmware extrapolable a operar con diferentes CIS, será necesario establecer unas reglas de versionado, que permitan obtener un mayor control de las diferentes imágenes (fichero binario que contiene el programa) que se ejecutarán en la MCU (CX3).

Cada imagen firmware seguirá la siguiente nomenclatura:

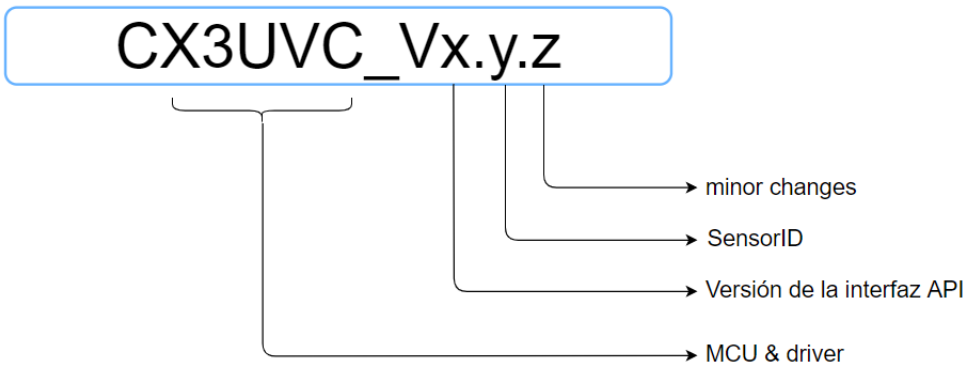
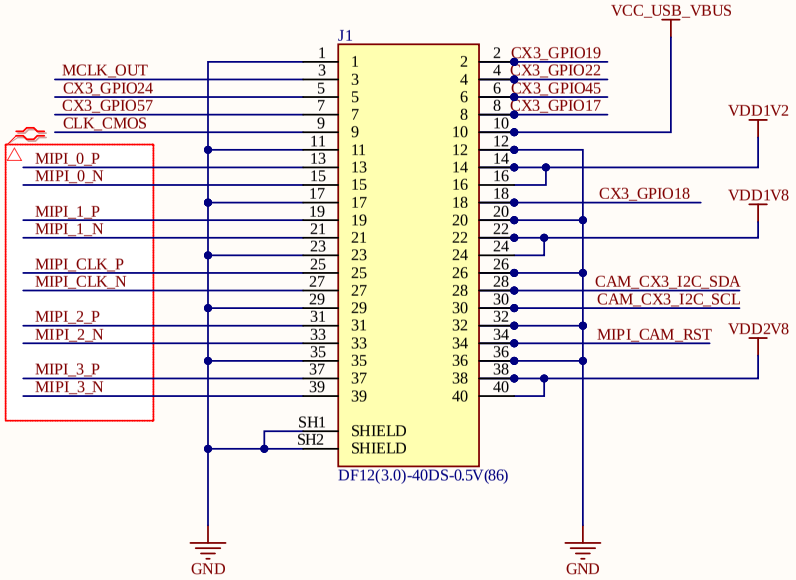


Figura B.1: Versionado de la imagen firmware

- **MCU & Driver:** Como su propio nombre indicia, hace mención en primer lugar a la MCU que conforma el dispositivo hardware, y a continuación se menciona el driver utilizado.
- **Versión de la Interfaz API:** Hace referencia a todos los puntos que implementan el handshake entre el host y la MCU, siendo la estructura base para poder controlar al CIS.
- **SensorID:** Hace referencia al sensor CMOS que embebe el dispositivo. Un mismo ID puede identificar a varios sensores de la misma familia, siempre y cuando posean una estructura de registros idéntica.
- **minor changes:** Hace referencia a cualquier cambio llevado a cabo en el firmware, sin que este afecte a la interfaz del API, por ejemplo: añadir una propiedad UVC más.

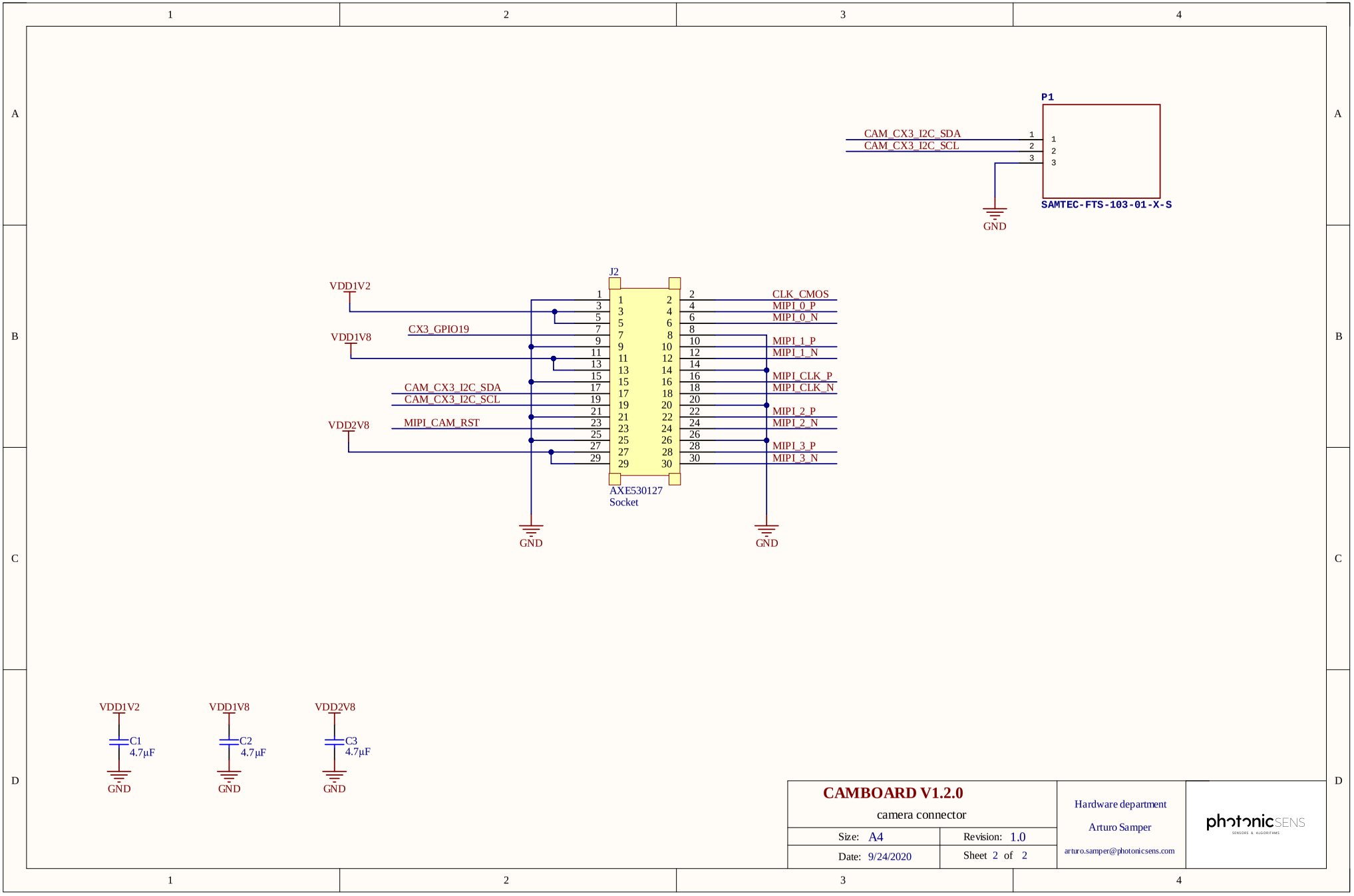
Apéndice C

Esquemático Camboard



CAMBOARD V1.2.0 board2board connector		Hardware department Arturo Samper arturo.samper@photonicsens.com
Size: A4	Revision: 1.0	
Date: 9/24/2020	Sheet 1 of 2	

photonicSENS
SENSORS & ALGORITHMS



CAMBOARD V1.2.0 camera connector		Hardware department Arturo Samper arturo.samper@photonicsens.com
Size: A4	Revision: 1.0	photonicSENS SENSORS & ALGORITHMS
Date: 9/24/2020	Sheet 2 of 2	