UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN RECEPTOR PARA COMUNICACIONES ÓPTICAS EN EL ESPECTRO VISIBLE (VLC) CON MODULACIÓN POR VARIACIÓN DEL COLOR (CSK)

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN RECEPTOR PARA COMUNICACIONES ÓPTICAS EN EL ESPECTRO VISIBLE (VLC) CON MODULACIÓN POR VARIACIÓN DEL COLOR (CSK)

HOJA DE FIRMAS

Alumno
Miguel Ángel Hernández Martínez

Tutor
Francisco Alberto Delgado Rajó

Fecha: Julio del 2017

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

DISEÑO E IMPLEMENTACIÓN DE UN RECEPTOR PARA COMUNICACIONES ÓPTICAS EN EL ESPECTRO VISIBLE (VLC) CON MODULACIÓN POR VARIACIÓN DEL COLOR (CSK)

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal

Fdo:

Secretario/a

Fdo:

Fecha: Julio del 2017

Índice de contenidos

MEMORIA

Índice de figuras

Glosario de términos

BER: *Bit Error Rate*

CCD: *Charge Coupled Device*

CMOS: *Complementary Metal Oxide Semiconductor*

CSK: Color Shift keying

DPPM: *Differential Pulse Position Modulation*

DSP: Digital Signal processor

Fmax: *Frecuencia Máxima*

FPGA: Field Programmable Gate Array

IEEE*: Institute of Electrical and Electronics Engineers*

LED: *Light emiting diode*

OOK.: On-Off Keying

OFDM: *Orthogonal Frequency Division Multiplexing*

PPM: *Pulse Position Modulation*

RF: *Radio Frecuencia*

***RGB**: Red, Green, Blue*

***Tmax:** Tiempo mínimo*

VHDL: Very Hard Descripcion Languaje

***VLC:** Visible Light Communications*

***VLCC:** Visible Light Communication Consortium*

***VPPM:** Variable Pulse Position Modulation*

WPAN: Wireless Personal Area Network

Capítulo 1

Introducción

Capítulo 1: Introducción

1.1 Introducción

Desde los albores de las primeras comunicaciones hasta hoy en día, las técnicas empleadas para el desempeño de estas funciones han ido evolucionando rápidamente, sobre todo en los últimos años.

Si bien hace algunas décadas, las comunicaciones vía radio *(RF)* han prevalecido sobre las comunicaciones ópticas, hoy en día el panorama está cambiando. Cada vez es más corriente el uso de sistemas ópticos frente a los de *RF*, llegando incluso a sustituirlos en determinadas aplicaciones. Esto se debe sobre todo a la reducción de los costes de manufacturación de estos productos y a las ventajas inherentes que presentan estos sistemas frente a los sistemas por *RF* en determinados entornos.

Con respecto a las comunicaciones inalámbricas y para aplicaciones de corto alcance, recientemente se están implementando sistemas que hacen uso de comunicaciones ópticas en el espectro visible, conocidos como *Visible Light* (*VLC*) combinando las funciones de iluminación ambiente con las de comunicaciones. En la figura 1 se aprecian diferentes escenarios de aplicación de este tipo de comunicaciones.

Figura 1-1. Escenarios típicos de VLC

En este trabajo se desarrolla un receptor para comunicaciones ópticas VLC basadas en modulación por variación del color (*Color Shift Keying*), que es uno de los tipos de modulaciones contempladas en el estándar 802.15.17 [8]

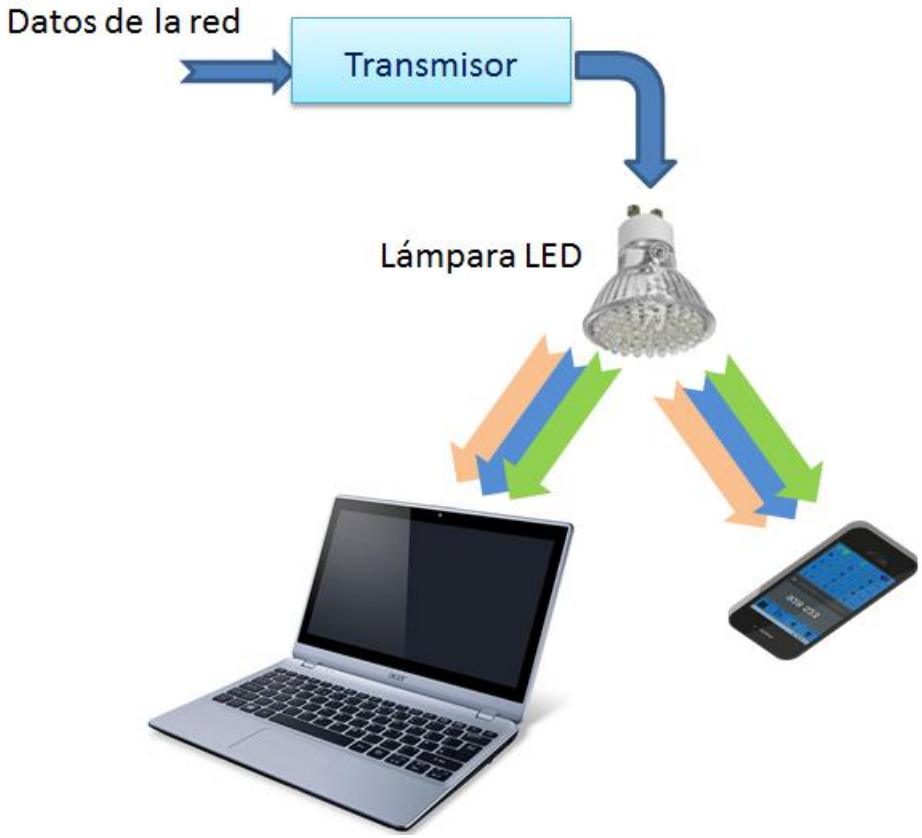


Figura 1-2. Sistema VLC

1.2 Antecedentes

La propuesta de utilizar dispositivos LED de conmutación rápida para modular la luz visible con el fin de transmitir información fue formulada por vez primera por Graham Pang [1], aunque la idea original ya la empleó Graham Bell con su Fotófono [2]. Sin embargo, no fue hasta 2003 que el grupo dirigido por el profesor Masao Nakagawa en la Universidad de Keio (Tokio, Japón), comenzó a desarrollar el concepto de sistemas combinados de iluminación y comunicación en recintos cerrados [3]. Estos sistemas son los conocidos como sistemas *Visible Light Communications (VLC).* A continuación se fundó un consorcio de diferentes entidades, tanto científicas como marcas comerciales que se denominó el Visible Light Communication Consortium **(**VLCC) en 2003.

Desde ese momento, muchas han sido las contribuciones realizadas empleando esta tecnología, como se refleja en la figura 1, con el fin de sustituir, en algunos casos, a las comunicaciones tradicionales por Radio Frecuencia (*RF*). En 2011 se elaboraba el primer estándar de IEEE para VLC, desarrollado por el grupo de trabajo 802.15.7 sobre redes de área local personales (WPAN, Wireless Personal Area Network) [4].

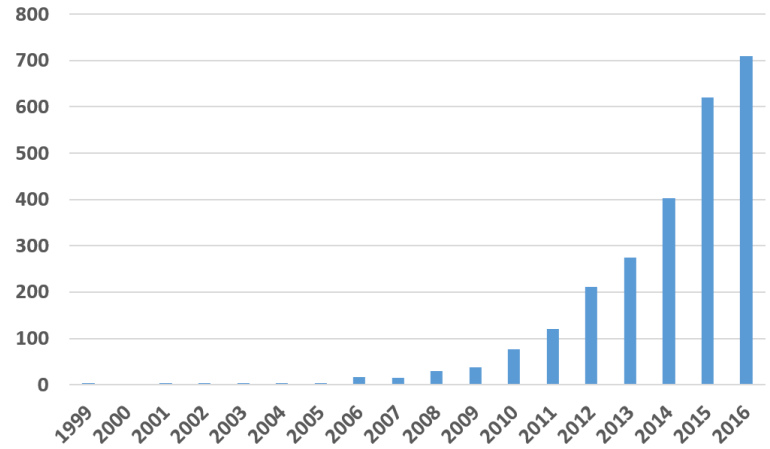


Figura 1-3. Evolución del número de publicaciones sobre VLC (Scopus)

Los sistemas de iluminación LED más empleados hacen uso de diodos LED basados en un diodo de longitud de onda correspondiente al azul rodeado por un fósforo amarillo, como se ve en la figura 2. Empleando este sistema se desarrollaron los primeros prototipos de sistemas de comunicaciones VLC [5][6][7]. La principal limitación es que la presencia del fósforo verde suponía que la velocidad de conmutación de las lámparas disminuía con lo que, a pesar de la gran potencia y de disponer de fotodiodos de gran ancho de banda, no se conseguían grandes velocidades.

Las modulaciones típicas empleadas en estos sistemas eran las propuestas en el estándar VLC. La primera empleada era la On-Off Keying (*OOK*), pero al consisitir en iluminar o no en función de que el dato entrante fuera un “1” o un “0”, produce en consiguiente parpadeo en el sistema de iluminación. Las más empleadas son las modulaciones por variación de la posición de pulso (*PPM, DPPM o VPPM*) [7][8][9] que garantizan que el nivel de iluminación se mantiene constante.

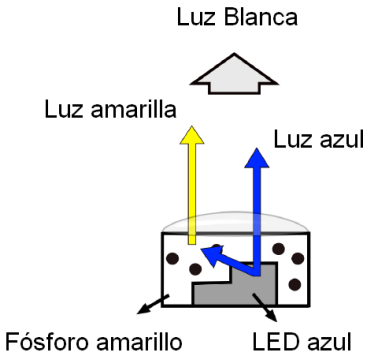


Figura 1-4. Sistema de iluminación de luz blanca

Otras aportaciones encaminadas a conseguir mayores velocidades y eficiencia son las basadas en multiplexación por división en frecuencia (Orthogonal Frequency Division Multiplexing, OFDM), que además permite la existencia de múltiples usuarios compartiendo el canal [10][11][12][13].

La aparición de nuevos tipos de sensores ópticos, así como la proliferación de los LED's RGB, que combinan las luces de los tres colores fundamentales implica dos factores:

- El aumento de la velocidad de conmutación de la lámpara
- Las posibilidad de emplear la detección de color

La figura 3 refleja cómo es un sistema de iluminación basado en el uso de LED's RGB.

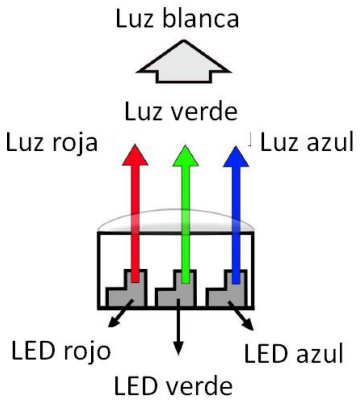


Figura 1-5. Esquema de lámpara RGB

Además de los LED's RGB, en los últimos tiempos se han empezado a emplear otro tipo de sensores que permiten detectar las variaciones de color en forma de su desglose en los tres colores fundamentales, como son las cámaras digitales

y los sensores de color. Estos sensores, en muchos casos se encuentran en los dispositivos móviles convencionales.

Todo esto ha impulsado a otro tipo de modulación más que es la modulación por variación de color (Color Shift Keying, CSK) [14][15][16], ya contemplada en el estándar VLC [8]. Además, en la actualidad se desarrollan también contribuciones basadas en el empleo de cámaras como receptores, aprovechando su presencia en los dispositivos móviles [17][18][19]. La mayor limitación de estos dispositivos es su baja velocidad, en torno a 30-60 frames por segundo en los dispositivos convencionales. Es por ese motivo que en este trabajo se procederá a la evaluación de estos sistemas CSK empleando sensores de color como receptores para intentar una mejora en la tasa de datos.

1.3 Objetivos

1.3.1 Objetivos generales

El objetivo principal del presente proyecto es el diseño de un receptor VLC. Este receptor tendrá que sacar los datos recibidos desde el transmisor hasta la salida del receptor.

Estos datos han sido codificados con modulación por variación del color (CSK), utilizando para ello un transmisor que excita a un led RGB con un patrón determinado. Un sensor RGB registra las variaciones de potencia de cada color emitido.

El receptor debe ser capaz de generar las señales adecuadas para la extracción de datos registrados por el sensor, almacenar dichos datos, decodificarlos correctamente y sacarlos a su salida.

Para la decodificación de los datos se empleará un sistema de decisión que se basa en la comparación del valor medio de los valores anteriormente recibidos y el valor actual. En base a esta decisión será la salida obtenida.

Para esto el receptor debe guardar el nivel de potencia de cada uno de los colores actuales, así como algunos valores anteriores para su posterior procesado.

1.3.2 Objetivos específicos

Los objetivos específicos de este proyecto llevan implícitas las siguientes tareas.

- Diseño de cada uno de los bloques funcionales de los que consta el receptor.
- Realización de las simulaciones pertinentes de cada uno de estos bloques para comprobar el funcionamiento correcto de cada una de las partes que componen el receptor.
- Simulación del receptor completo comprobando el perfecto funcionamiento del mismo.
- Implementación del receptor.
- Validación del correcto funcionamiento del receptor
- Obtención de los resultados prácticos y análisis de los mismos.

Capítulo 2

Sistemas VLC por modulación basada en la variación de Color

2. Sistemas VLC por modulación basada en la variación de color

2.1 Modulación por variación del color (CSK)

2.1.1 Introducción

La modulación por variación del color es un esquema de modulación específico para comunicaciones VLC y que se detalla en la sección 8.2 del estándar IEEE 802.15.7 [8].

En el estándar IEEE 802.15.7 [8] sección 8.3 se especifica un código a ciertas franjas de longitudes de onda. Como podemos ver en la figura 2-1 hay 7 códigos definidos y uno reservado. En esta figura podemos ver, por ejemplo, que el código “000” define la banda de luz visible que va desde los 380 nm a los 478 nm, el código “001” comprende la banda desde los 478 nm hasta los 540 nm y así correlativamente hasta la banda que va desde los 726 nm hasta los 780 nm, que corresponde con el código “110”. Como podemos apreciar los primeros 7 códigos aquí definidos abarcan todo el espectro visible. El código “111” permanence reservado.

La señal CSK es generada usando 3 fuentes de color de las 7 bandas definidas en la figura 2-1. Los tres vértices en una constelación CSK están definidos según la longitud de onda central de 3 bandas de color en las coordenadas xy de color de la figura 2-2.

Wavelength (nm)		Spectral width (nm)	Code
380	478	98	000
478	540	62	001
540	588	48	010
588	633	45	011
633	679	46	100
679	726	47	101
726	780	54	110
Reserved			111

Figura 2-1. Plan de las bandas para luz visible

Band (nm)	Code	Center (nm)	(x, y)
380–478	000	429	(0.169, 0.007)
478–540	001	509	(0.011, 0.733)
540–588	010	564	(0.402, 0.597)
588–633	011	611	(0.669, 0.331)
633–679	100	656	(0.729, 0.271)
679–726	101	703	(0.734, 0.265)
726–780	110	753	(0.734, 0.265)

Figura 2-2. Coordenadas xy de color

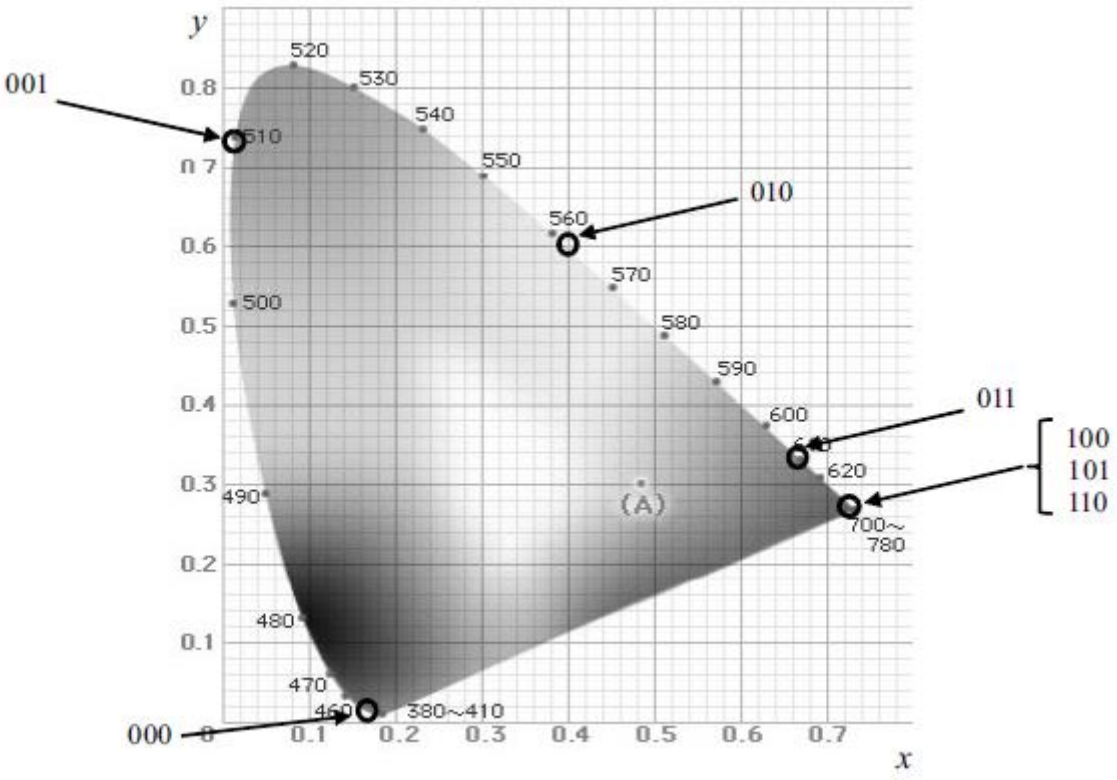


Figura 2-3. Centro de las bandas de color en coordenadas xy

Este tipo de modulación consiste en asignar a cada dato a transmitir unas determinadas bandas de color que trasmite el emisor. El número de bandas a transmitir depende de la constelación CSK elegida.

En la siguiente figura 2-4 podemos ver un esquema de un transceptor VLC en el que se hace uso de la modulación CSK.

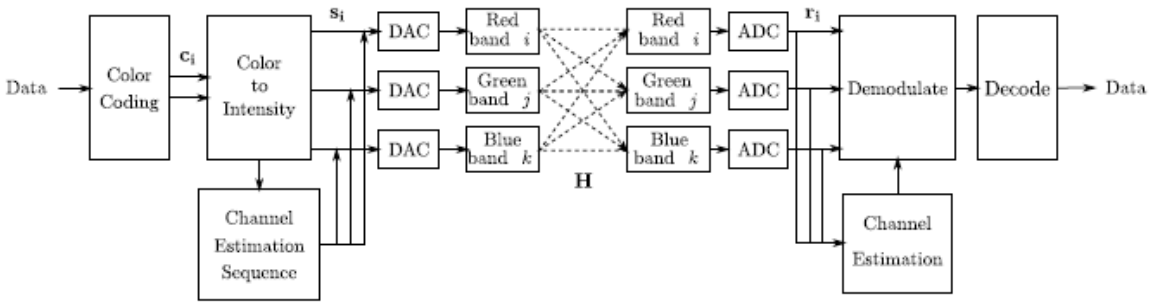


Figura 2-4. Transceptor CSK

2.2 Estándares comunes para constelaciones CSK

En la sección 12.8 del estándar IEEE 802.15.7 [8] se especifican las reglas de diseño para las constelaciones de 4,8 y 16 símbolos.

2.2.1 Constelación 4-CSK

Las reglas para el diseño de una 4-CSK tal como está descrito en el estándar IEEE 802.15.7 se muestra en la figura 2.5. Los elementos S0 al S3 representan los símbolos del 4-CSK. Los puntos I, J, K son los centros de 3 bandas de color en las coordenadas xy de color. Los elementos S1, S2 y S3 son los vértices del triángulo IJK. S0 representa el centro de dicho triángulo.

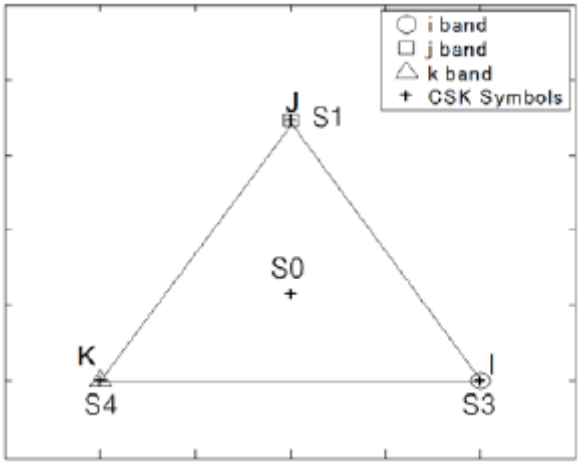


Figura 2-5. Constelación 4-CSK

Las múltiples combinaciones de las fuentes ópticas para una 4-CSK asume que el pico espectral de la fuente óptica está en el centro del plan de bandas que puede ser obtenido en Yokoi, et al. [15].

2.2.2 Constelación 8-CSK

Esta constelación está descrita como la anterior. en el estándar IEEE 802.15.7. Esta se muestra en la figura 2.6. Los elementos S0 al S7 representan los símbolos del 8-CSK. Los puntos I, J, K son los centros de 3 bandas de color en las coordenadas xy de color. Los elementos S0, S4 y S7 son los vértices del triángulo IJK.

Los elementos S1 y S2 son puntos que dividen el segmento JK y JI con una relación 1:2. Los puntos B y C son puntos intermedios de los segmentos JI y JK respectivamente. El elemento S6 es el punto medio del segmento KI.

El punto A es el centro del triángulo B-S6-I y el punto D es el centro del triángulo C-K-S6. El elemento S3 es el punto que divide el segmento AB con una relación de 1:2 y el elemento S5 que divide el segmento DC con una relación de 1:2.

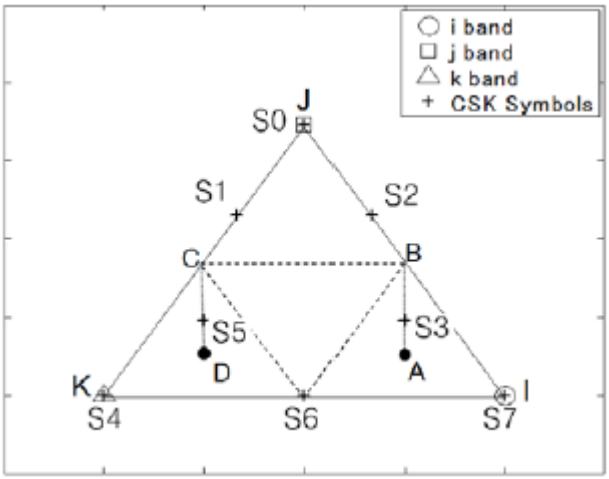


Figura 2-6. Constelación 8-CSK

Las múltiples combinaciones de las fuentes ópticas para una 4-CSK asume que el pico espectral de la fuente óptica está en el centro del plan de bandas que puede ser obtenido en Yokoi, et al. [15].

2.2.3 Constelación 16-CSK

Como las anteriores, también esta constelación está descrita en el estándar IEEE 802.15.7 y se muestra en la figura 2.7. Los elementos S0 al S15 representan los símbolos del 16-CSK Los puntos I, J, K son los centros de 3 bandas de color en las coordenadas xy de color como en los casos anteriores. Los elementos S2 y S8 están dispuestos a una distancia de un tercio del segmento JK. Los elementos S3 y S12 están dispuestos a una distancia de un tercio del segmento JI. Los elementos S11 y S14 están dispuestos a una distancia de un tercio del segmento KI. S0 representa el centro del triángulo IJK. Los elementos S1, S4, S6, S7, S9 y S13 son el centro de cada uno de los pequeños triángulos que se pueden ver en la figura 2.7.

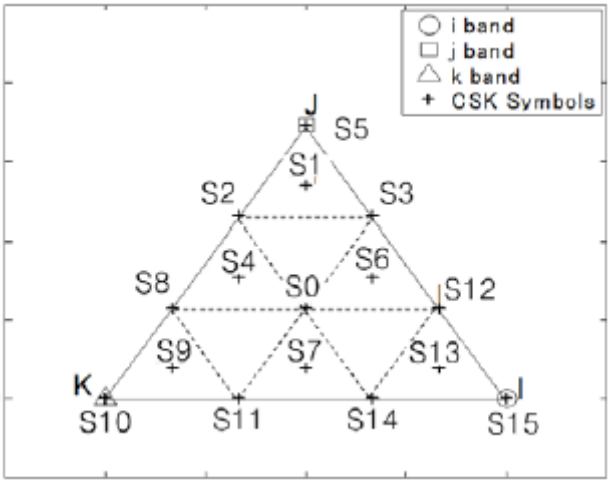


Figura 2-7. Constelación 16-CSK

2.2.4 Mapeado de datos para CSK

El mapeado de datos para una 4-CSK se muestra en la figura 2-8. Dos bits son asignados por símbolo.

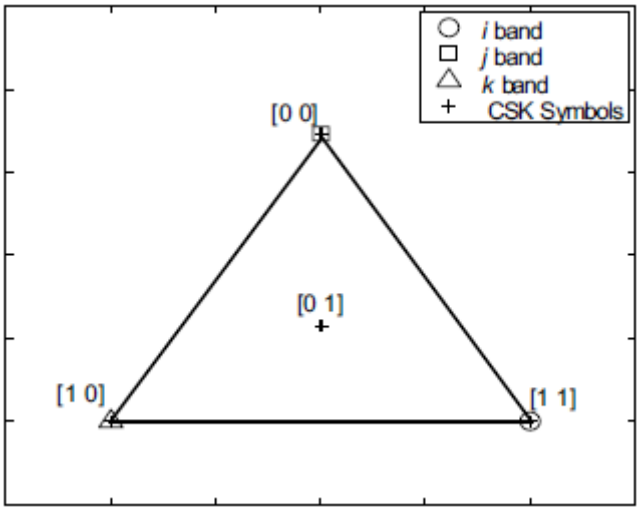


Figura 2-8. Mapeado de datos para una 4-CSK

El mapeado de datos para una 8-CSK se muestra en la figura 2-9. En este caso tres bits son asignados por símbolo.

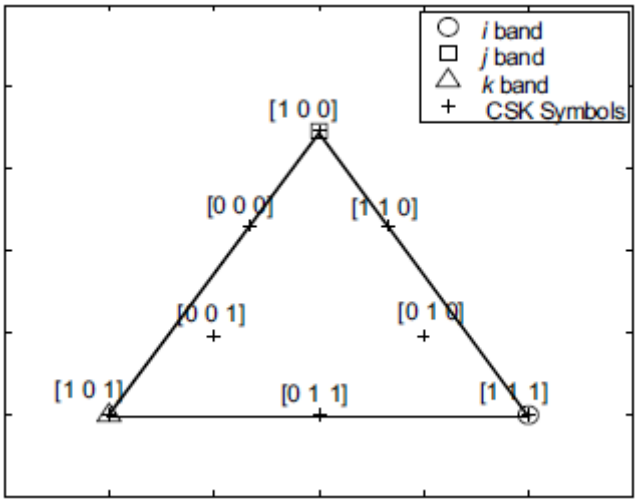


Figura 2-9. Mapeado de datos para una 8-CSK

El mapeado de datos para una 16-CSK se muestra en la figura 2-10. Esta constelación usa cuatro bits por símbolo.

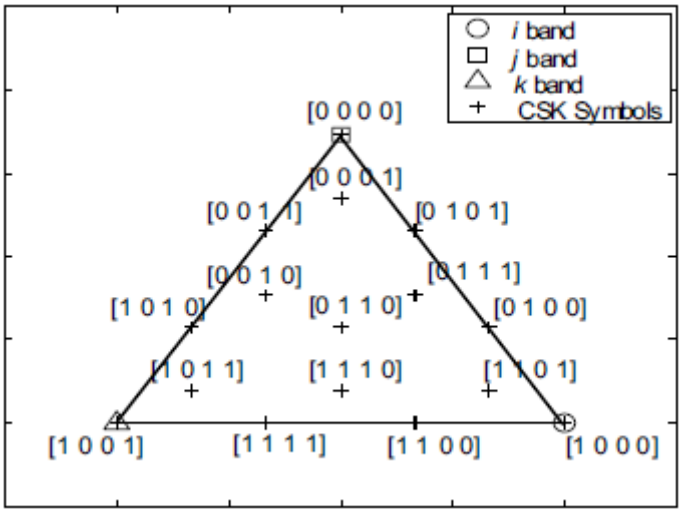


Figura 2-10. Mapeado de datos para una 16-CSK

Capítulo 3

Diseño del receptor VLC

3. Diseño del receptor VLC

3.1 Introducción

En este capítulo abordaremos el problema del diseño e implementación de un receptor VLC. Primero empezaremos con una breve descripción funcional del transmisor diseñado en esta universidad. Seguidamente haremos una descripción de los bloques principales de los que consta este receptor, donde se indicarán las funciones a realizar por cada bloque.

A continuación se propondrá un diagrama de bloques para el diseño del receptor y por último describiremos los componentes electrónicos que pueden ser candidatos para el diseño del receptor VLC

3.2 Transmisor

Por motivos de sencillez, en el transmisor se usa la constelación que está representada en la figura 3-1.

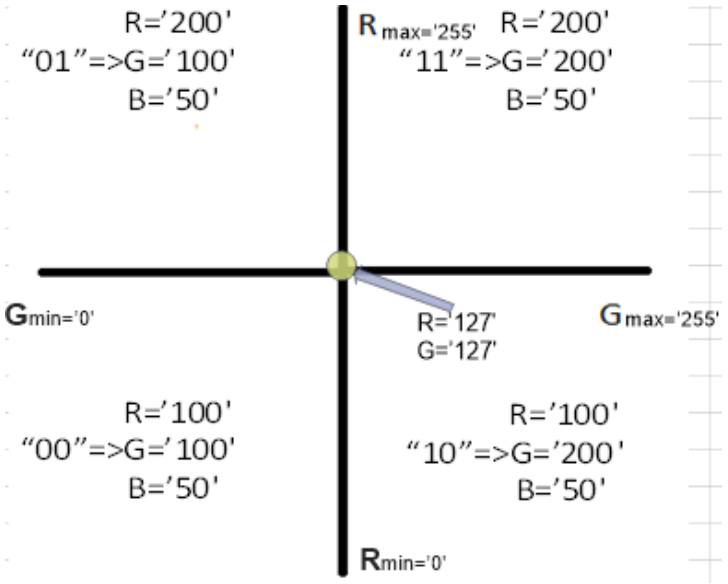


Figura 3-1. Constelación usada en el transmisor

El eje de abscisas representa la potencia relativa del color verde y el de ordenadas la del rojo. En nuestro caso la potencia del color azul se mantiene

constante. Podemos ver que el **"11"** representa niveles de potencia de rojo y verde por encima de la media (la media es el origen de coordenadas, **R='127'**, **G='127'**), el **"00"** por niveles de rojo y verde relativamente bajos, el **"01"** representa un nivel alto de rojo y bajo de verde, al contrario que el **"10"**.

El transmisor envía 20 secuencias diferentes de datos, donde cada dato produce una salida diferente. Cada una de estas salidas excita de diferente forma un diodo LED RGB, de modo que la potencia relativa de cada uno de los colores del mismo varía según los datos que se transmiten en ese momento. En este transmisor se ha mantenido constante la potencia del color azul como se ha comentado anteriormente. En la tabla de la figura 3-2 podemos ver los diferentes niveles de potencia relativa para cada uno de los datos a transmitir en una escala que va desde el **'0'** hasta el **'255',** donde el **'cero'** representa un valor nulo de potencia y el **'255'** la máxima potencia. Tras cada transmisión de datos se envía un impulso de sincronismo por una línea cableada que es el encargado de sincronizar el receptor con el transmisor.

DATO A TRANSMITIR	POTENCIA RELATIVA R	POTENCIA RELATIVA G	POTENCIA RELATIVA B
"11"	'200'	'200'	'50'
"01"	'200'	'100'	'50'
"10"	'100'	'200'	'50'
"00"	'100'	'100'	'50'

Figura 3-2. Asignaciones entre datos y colores

3.3 Receptor

En esta sección vamos a describir el funcionamiento del receptor, el diagrama de bloques del mismo y los componentes candidatos para la implementación del mismo.

3.3.1 Funcionamiento

El receptor consta de dos bloques fundamentales, el de adquisición de datos (COLOR READER) y el de procesado de los mismos (DATA PROCESS). El bloque de adquisición de datos se encarga de registrar los diferentes valores RGB captados por el sensor y el segundo que registra los 16 valores anteriores. Con estos 16 valores anteriores se calcula la media (una para cada color) e inmediatamente se compara con el valor actual.

El bloque encargado de hacer dicha comparación es el decisor. Este circuito se encarga de decidir la salida del receptor en base a la lectura del valor actual y la media de los 16 valores RGB anteriores. En teoría el valor medio correspondería con un valor que se encuentra entre el valor de máxima potencia del color rojo y verde (**R=’255’**, **G=’255’**) y el mínimo de ambos (**R=’0’**, **G=’0’**), valor cuyo lugar geométrico está el centro de la constelación de la Figura 3-1 (**R=’127’**, **G=’127’**).

Dependiendo de este resultado de comparación obtenemos una de las cuatro posibles salidas (**“00”**, **“11”**, **”10”**, **”01”**).

3.3.2 Diagrama de bloques

Para la realización del mismo proponemos el esquema de la figura 3-3. Los bloques de los que consta este receptor y sus funciones son las siguientes:

- Almacenamiento de datos
  - Se encarga de almacenar los valores actuales y algunos anteriores recibidos por el sensor RGB
- Cálculo de media
  - Este módulo calcula el valor medio de todos los datos recibidos con anterioridad que se encuentran memorizados en el módulo anterior.

- Decisor
  - Aquí se decide los datos de salida del receptor en base al valor medio calculado anteriormente y el valor actual recibido por el sensor RGB.
- Control
  - Es el responsable de la correcta adquisición de datos por parte del sensor y el control de su procesado.

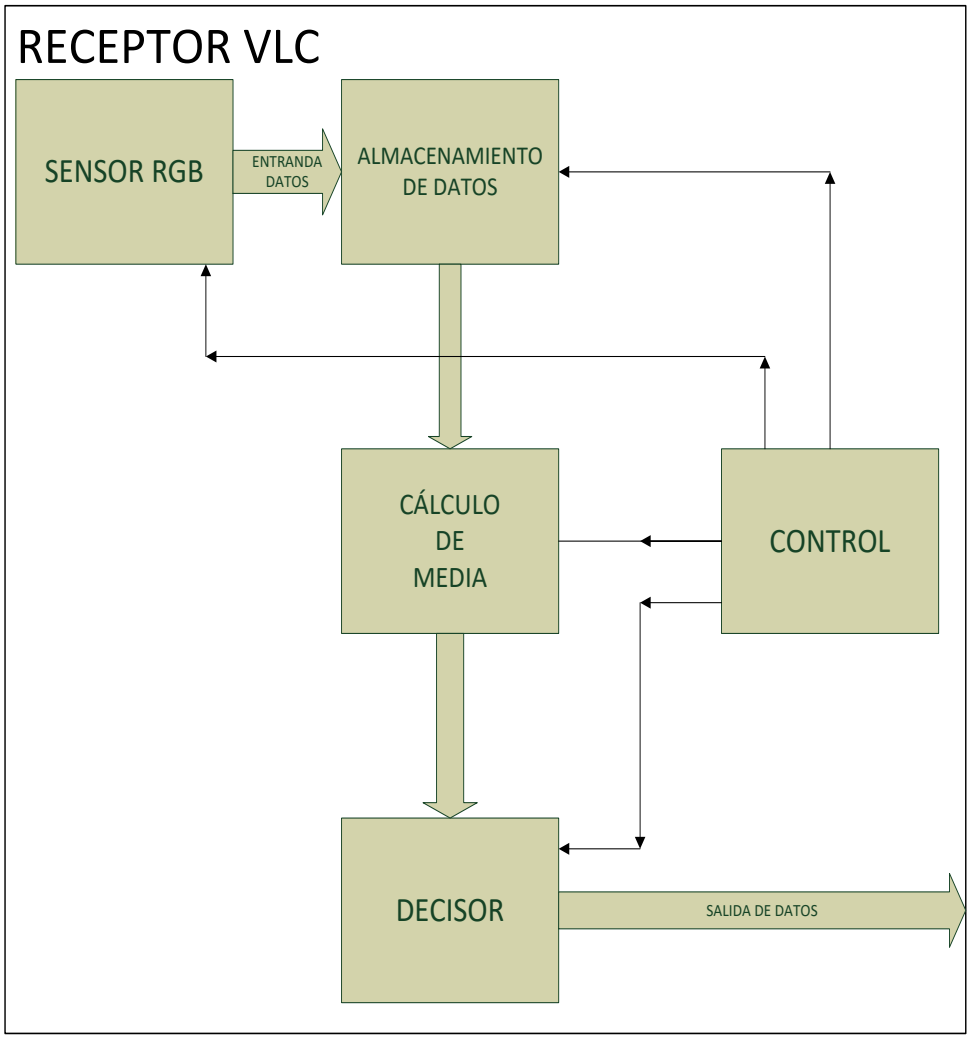


Figura 3-3. Diagrama de bloques del receptor VLC

3.3.3 Componentes electrónicos

Un sistema de recepción por luz visible (VLC) como el propuesto aquí, hace uso de varios componentes fundamentales, entre los que cabe destacar los fotosensores empleados para la captación de la luz, los componentes para almacenado de datos, los de procesado de señales y los de control de las mismas.

En la siguiente sección vamos a hacer una breve introducción a algunos componentes que pueden realizar dichas tareas.

3.3.3.1 Fotosensores

Son dispositivos sensibles a la luz en los cuales los fotones incidentes sobre una superficie abierta a través de una ventana, es transformada en una corriente eléctrica. Esta corriente eléctrica es directamente proporcional a la intensidad de la luz incidente.

Los más comunes son los de tipo semiconductor y se fabrican con diferentes composiciones en función de la longitud de onda. En la figura 3-4 podemos ver los diferentes materiales empleados para la fabricación de distintos tipos de fotosensores y la longitud de onda a la que responden.

Material	Longitud de onda (nm)
Silicio	190–1100
Germanio	800–1900
Indio galio arsénico (InGaAs)	800–2600
sulfuro de plomo	<1000-3900

Figura 3-4. Longitud de onda en función del material

3.3.3.1.1 Responsividad

La responsividad es la relación que existe entre la corriente producida en el fotodiodo y la potencia de la luz incidente sobre el mismo. Es una medida de la eficiencia de conversión de un fotodiodo.

3.3.3.1.2 Fotodiodos PIN

El fotodiodo PIN es un tipo de fotosensor semiconductor tipo PN, en la que se crea una zona de deplexión amplia por la aplicación de una tensión inversa. Con la aplicación de tensiones inversas altas se reduce la capacidad parásita del diodo, con lo que aumenta su frecuencia de corte. Son muy fiables, presentan una buena respuesta en frecuencia y generan poco ruido.

3.3.3.1.3 Fotodiodos de avalancha

Presentan una responsividad mucha más alta que los diodos PIN. Su estructura es ligeramente diferente y necesita de tensiones inversas mucho mayores que los PIN para conseguir el efecto de avalancha por el cual se obtiene un efecto de amplificación de la fotocorriente. Son componentes más caros que los que los fotodiodos PIN, necesitan de circuitos para la compensación de las variaciones de la ganancia debida a los cambios de temperatura y son más ruidosos que los fotodiodos PIN.

Figura 3-5. Imágenes de fotodiodos comerciales

3.3.3.1.4 Sensores RGB

Son fotodetectores sensibles a los colores primarios (rojo, verde y azul). Suelen estar formados por una matriz de fotodiodos sobre la cual se coloca un filtro Bayer. Con este filtro se consigue cada fotodiodo asociado a un color primario determinado sólo reciba la parte del espectro asociada con ese color.

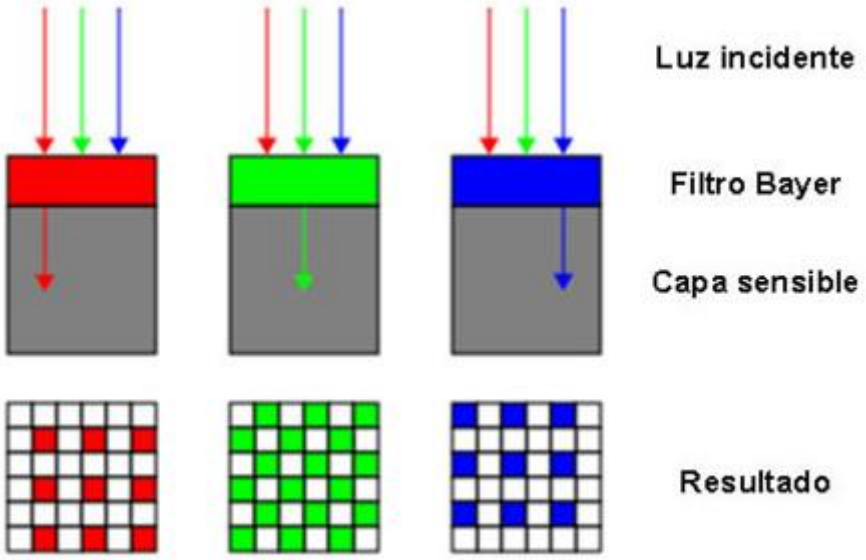


Figura 3-6. Diferentes filtros de bayer

Estos sensores se presentan en diferentes formatos dependiendo del número de fotodiodos empleados, el tipo de salida (análogica, digital serie, digital compatible con el estándar I^2C, etc) y el tipo de encapsulado.

En la figura 3-7 podemos ver uno de la casa Hamamatsu que está formado por una matriz de 9 x 9 elementos, con salida digital serie.

Figura 3-7. Sensor RGB integrado de la casa Hamamatsu

En la figura 3-8 podemos ver otro sensor RGB formado por 3 fotodiodos de la misma casa que el anterior. Este presenta una salida directa desde cada fotodiodo a cada una de las patillas del encapsulado.

Figura 3-8. Sensor RGB analógico

3.3.3.1.5 Sensor CCD

Los **CCD** (Charge Coupled Device) son sensores formados por semiconductores con tecnología MOS (Metal Oxide Semiconductor) que están **distribuidos en forma de matriz.**

Su función es la de acumular una carga eléctrica en cada una de las celdas de esta matriz. Estas celdas son los llamados **píxeles**. La carga eléctrica almacenada en cada píxel, dependerá en todo momento de la **cantidad de luz** que incida sobre el mismo. Cuanta más luz incida sobre el píxel, mayor será la carga que este adquiera.

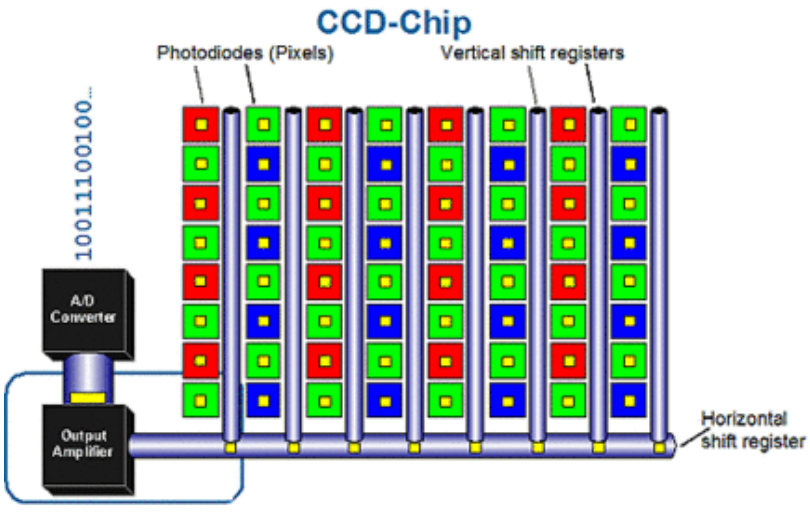


Figura 3-9. Sensor CCD

El CCD convierte las cargas de las celdas de la matriz en voltajes y entrega una señal analógica en la salida, que será posteriormente digitalizada. En los sensores CCD, se hace una lectura de cada uno de los valores correspondientes a cada una de las celdas. Entonces, es esta información la que un **convertidor analógico-digital** traduce en forma de datos.

3.3.3.1.6 Sensor CMOS

Los sensores CMOS (Complementary Metal Oxide Semiconductor) al igual que los CCD están formados por semiconductores con tecnología CMOS que también están dispuestos en forma de matriz. La diferencia es que cada celda es independiente de la otra y además la digitalización de los píxeles se realiza por unos transistores internos a la misma celda, por lo que no es necesario un conversor análogico-digital externo como es el caso de los CCD. Con esto se reduce tanto el tamaño como el coste, además presentan mayor sensibilidad que los CCD, aunque su rango dinámico es menor.

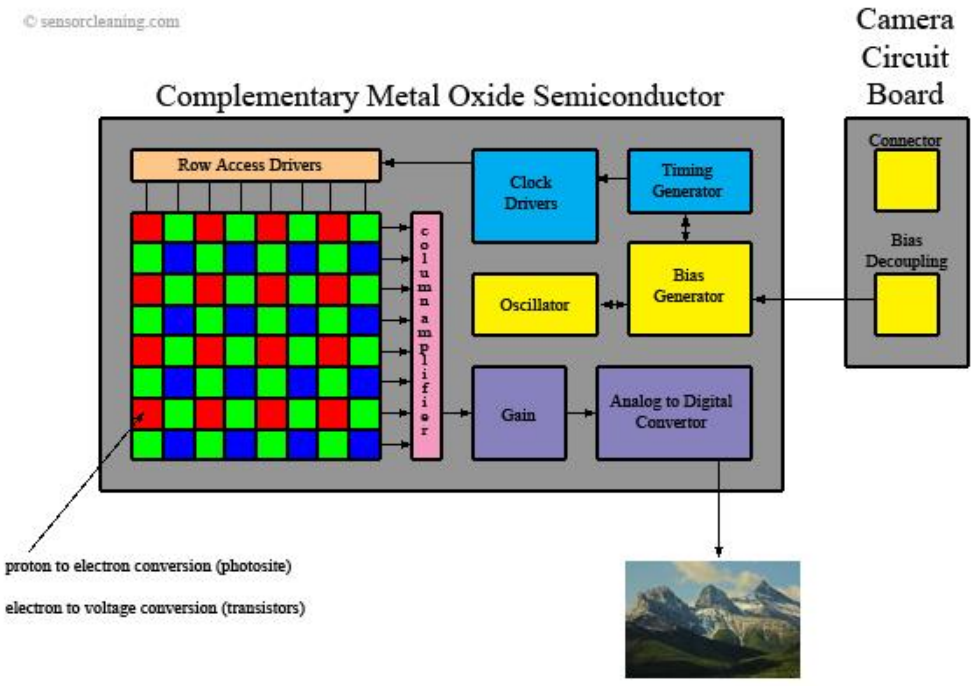


Figura 3-10. Sensor CMOS

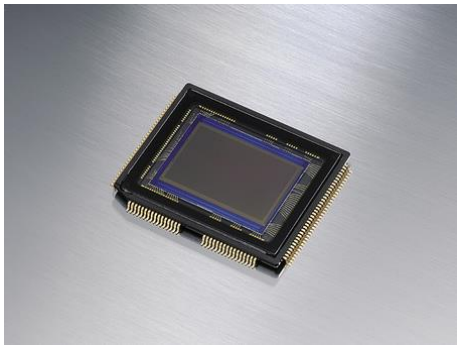


Figura 3-11. Fotografía de un sensor CMOS

3.3.3.2 Dispositivos de procesamiento y control de señales

En los siguientes apartados vamos a hacer un repaso a los dispositivos más comunes utilizado en el proceso de señales y control describiendo sus características básicas. Estos dispositivos suelen implementarse con componentes electrónicos programables.

3.3.3.2.1 Procesador digital de señales

Los procesadores digitales de señales o DSP's (Digital Signal processor), son circuitos integrados basados en un microprocesador que posee un conjunto de instrucciones, un hardware y un software optimizado para aplicaciones que requieran operaciones aritméticas a muy alta velocidad.

Las aplicaciones más habituales en las que se emplea un DSP son el procesado de señales de audio y video en tiempo real.

Figura 3-12. DPS de la casa Texas instruments

3.3.3.2.2 Microprocesadores y microcontroladores

Los microprocesadores son circuitos integrados programables de propósito general. Necesitan circuitos periféricos como memorias, módulos de entrada/salida, etc. Estos componentes se encuentran sobre todo en ordenadores, aunque también los podemos encontrar en multitud de equipos como diversos tipos de medidores, circuitos de control, etc.

Los microcontroladores son básicamente igual a los microprocesadores con la diferencia de que llevan incorporados cierta cantidad de memoria para el almacenado de programa y datos, circuitos de timer, líneas de entrada y salida para el manejo de periféricos, etc. La elección entre un componente u otro depende de la aplicación específica a realizar.

3.3.3.2.3 FPGA's

Las FPGA (Field Programmable Gate Array), son dispositivos programables que contienen bloques de lógica cuya conexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de descripción especializado como el VHDL o Verilog. La lógica programable puede realizar funciones tan simples como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en chip. Las aplicaciones son muy similares a la de los ASIC'S, aunque son más lentas, tienen mayor consumo de energía y no pueden albergar sistemas tan complejos como ellos. Sin embargo las FPGA tienen la ventaja de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), En la actualidad son muy utilizadas para la realización de sistemas digitales complejos, sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo también es menor.

Figura 3-13. FPGA de Xilinx

Capítulo 4

Implementación del receptor VLC

4. Implementación del receptor VLC

4.1 Introducción

Una vez repasadas las tecnologías existentes en los capítulos anteriores, se toman las decisiones adecuadas en cuanto a implementación se refiere. En este apartado se expondrán un diagrama de bloques definitivo en el que se basa nuestra implementación, desarrollado a raíz del apartado de diseño, el software utilizado para llevar a cabo la implementación, los recursos hardware utilizados y el motivo de su elección.

4.1.1 Diagrama de bloques

En la figura 4-1 podemos ver el esquema de bloques completo del receptor. Como ya hemos comentado anteriormente este consta de 2 bloques fundamentales, el COLOR READER y el DATA PROCESS.

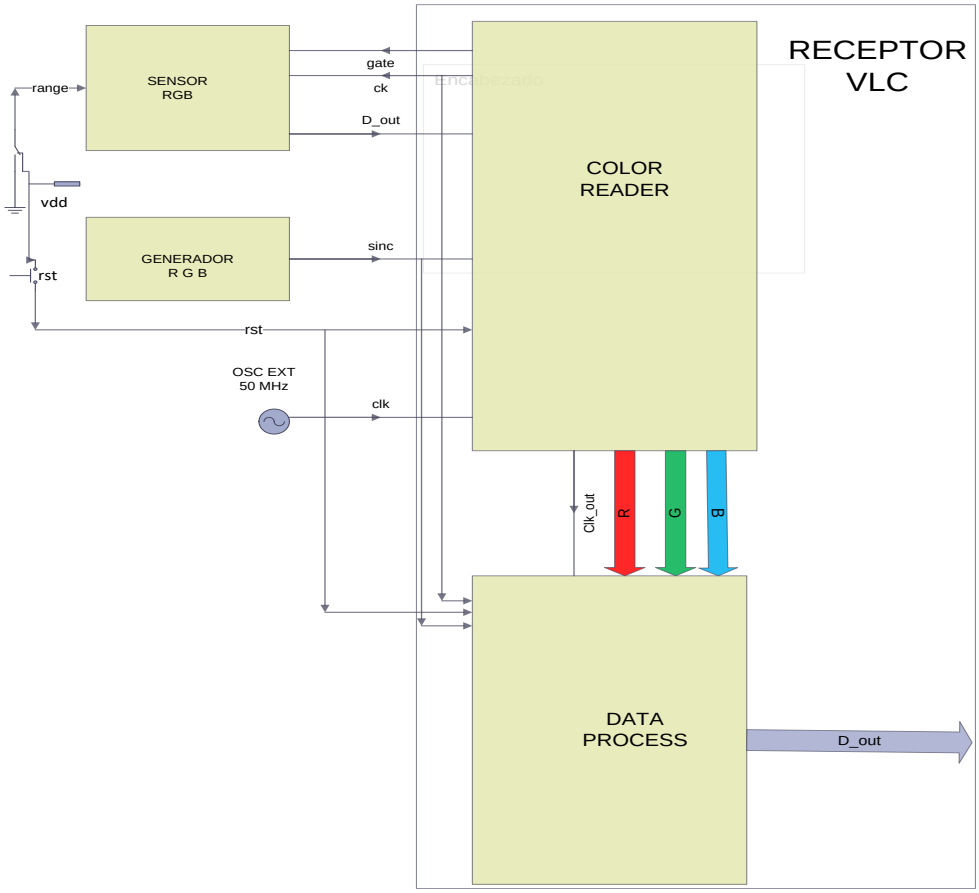


Figura 4-1. Diagrama de bloques del receptor VLC

4.1.2 Descripción funcional y estructural de cada bloque

4.1.2.1 COLOR READER

Es uno de los dos bloques fundamentales de los que consta nuestro receptor. Su función principal es la de extraer los datos captados por el sensor RGB, generando las señales adecuadas para el control de los diferentes módulos que integran este bloque.

En la figura 4-2 podemos ver los módulos de los que consta este bloque que listamos a continuación:

- Clk SKL
- Control reader
- Demux_1_3
- R_Reg, G_Reg, B_Reg

Las entradas a este módulo son "***Clk_in"*** que proviene de un reloj exterior de 50 MHz, ***"Rst"*** (también es salida) con la cual se resetea los principales módulos del sistema, la señal ***"sinc"*** que se utiliza como señal de sincronismo de lectura del sensor RGB y la de ***"Data_in"*** por donde introducimos los datos de lectura de cada uno de los colores que proporciona el sensor RGB.

Como salidas tenemos la línea ***"Gate"*** que es la que se encarga de habilitar la lectura del sensor y además sirve como señal de control en el módulo DATA PROCESS (Data process control), "***Clk"*** que una versión de "***Clk_in"*** escalada en frecuencia (CLK skl) y es la señal de reloj de todo el sistema.

4.1.2.1.1 Clk skl

Este módulo se encarga de dividir la frecuencia de entrada al circuito (50Mhz) que se obtiene de un oscilador que está integrado en la placa de prototipado por un valor entero de 2,4,6,8,32,64 ó 128 para obtener una frecuencia menor a su

salida de 25, 12.5, 6.25, 3.125, 1.5625, 0.78125 y 0.39 MHz respectivamente. La señal de salida de este módulo ("***Clk_out"***) se utiliza como reloj del sistema

completo. El motivo de este divisor es el de poder obtener retardos de tiempo lo suficientemente largos (eliminación de rebotes en pulsadores, "*wait's"* en máquinas de estado finito, etc.) sin necesidad de circuitos contadores muy grandes además de probar el funcionamiento del sistema a diferentes frecuencias.

4.1.2.1.2 Control reader

Con este módulo hacemos el control de la lectura de datos recogidos por el sensor RGB. El corazón de este módulo es una máquina de estado finitos, esta tiene once estados diferentes. En cada uno de estos estados se efectúan diferentes operaciones. Las operaciones más importantes llevadas a cabo son la generación de todas las señales que permiten sincronizar todos los circuitos implicados en la lectura de datos y su almacenamiento en registros.

La secuencia de pasos a seguir para la lectura desde el sensor como se explica en la sección 4.1.3.1 (funcionamiento del sensor) de este capítulo y la activación de las diferentes señales que controlan al demultiplexor y registros son llevadas a cabo por este módulo.

4.1.2.1.3 Demux_1_3

Este demultiplexor se encarga de recoger los datos enviados por el sensor y enviarlos a cada uno de los registros (RGB) de forma correcta. Esto se consigue con la activación de la señal ***"sel"*** del multiplexor que es recibida desde la unidad de control (color reader).

4.1.2.1.4 R_Reg, G_Reg, B_Reg

En estos registros se almacenan cada uno de los valores correspondientes a los colores de rojo, verde y azul que son captados por el sensor. El sensor utiliza 12 bits para la representación de cada color. Estos registros son de ocho bits. Hemos hecho un truncamiento a 8 bits registrando solamente los 8 valores más significativos en cada color. Esto se ha hecho porque para calcular la media en el módulo 'media' necesitamos dividir, operación que sólo se puede implementar para valores de potencia de 2 (2, 4, 8, 16,…).

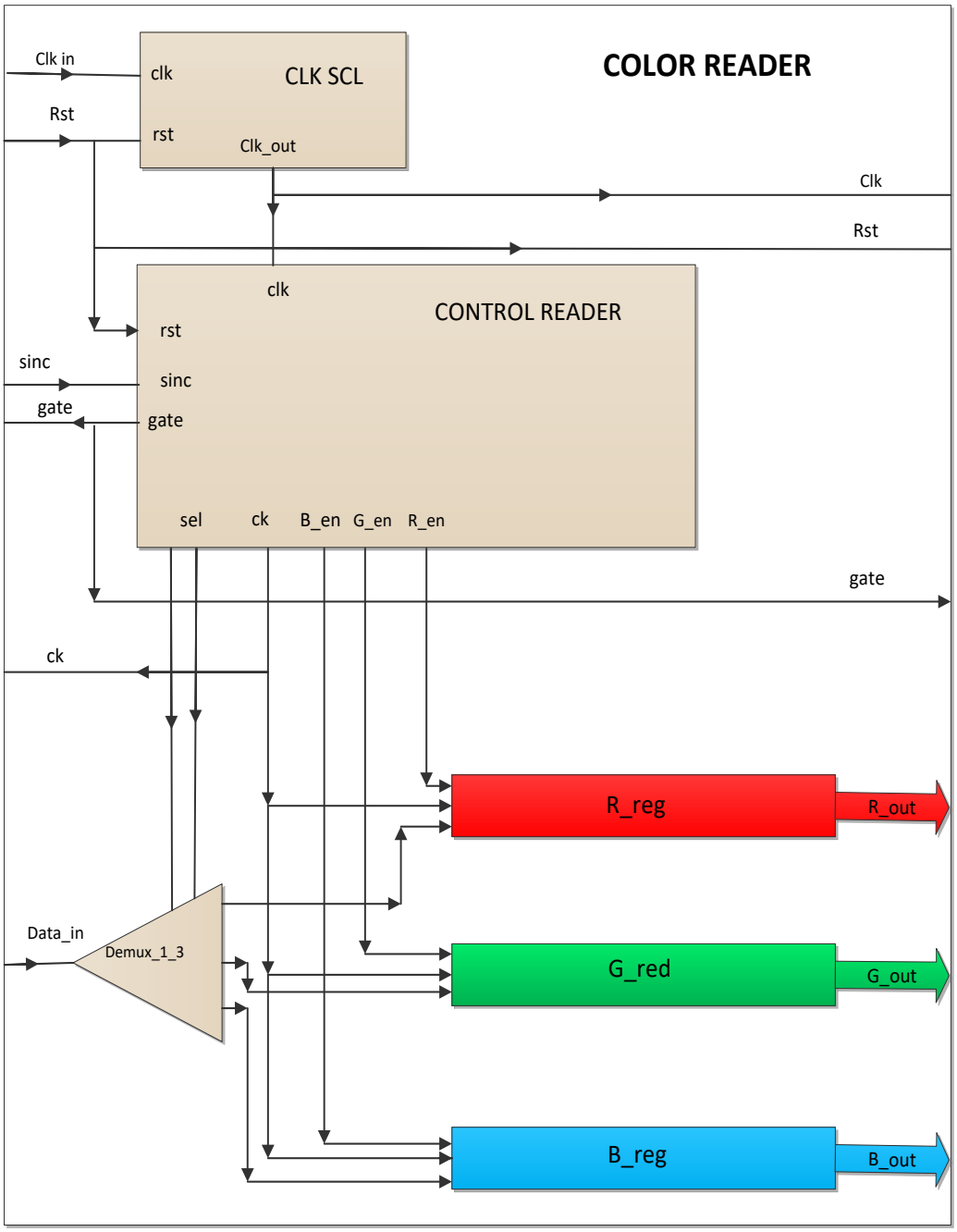


Figura 4-2. Diagrama de bloques del color reader

4.1.2.2 DATA PROCESS:

La finalidad de este bloque es procesar toda la información entregada por el bloque COLOR READER. En él se almacenan los valores actuales y anteriores

de RGB con los cuales se lleva a cabo el proceso de detección y decodificación de señales de entrada y salida.

Los módulos de los que consta este bloque lo podemos ver en la figura 4-3 que consta de los siguientes módulos:

- Data process control
- Decisor
- Media R G B
- Matrix R G B

La entrada de este módulo son principalmente los valores actuales medidos por el sensor RGB (R_in, G_in, B_in). La salida (***"D_out"***) son los valores con los que se ha codificado la señal original en el transmisor (**"00"**, **"11"**, **"10"**, **"01"**). Hay otras señales de entrada que se usan para el control del funcionamiento de este módulo (***"Clk", "Rst", "Gate"***).

4.1.2.2.1 Data process control

Este módulo tiene una función similar al que tiene el módulo color reader control en el bloque COLOR READER. También se basa en una máquina de estados finitos. Ésta sólo tiene 3 estados bien definidos. Dependiendo del estado de la señal ***"Gate"*** se activa o no los demás módulos de los que forma parte el bloque DATA PROCESS. Cuando la señal ***"Gate"*** está a nivel bajo se habilitan los módulos decisor, media R G B y matrix R G B (***"med_en", "mat_en", "dec_en"***). Por el contrario, cuando esta señal pasa a nivel alto después de un determinado tiempo para la lectura de los datos se deshabilitan todas ellas, manteniendo así los valores de una lectura entera del sensor RGB.

4.1.2.2.2 Decisor

Aquí es donde se decide la salida del receptor en función de las entradas. El circuito tiene principalmente dos entradas, la del valor actual que leemos del

sensor RGB y el valor medio de los 16 valores anteriores de RGB. Según esta comparación tendremos una de las cuatro salidas posibles (**“00”**, **“11”**, **”10”**, **”01”**).

4.1.2.2.3 Media R G B

Este módulo calcula la media de los valores de los 16 valores anteriores de RGB. Tiene una salida por cada color RGB con el valor medio de éstos y 16 x 3 entradas. En estas entradas tenemos presentes los 16 valores anteriores de RGB con los cuales se calcula media temporal de cada color RGB que se transmiten hasta las salidas RGB. Esta media servirá para comparar con los valores instantáneos de cada color obtenidos para la discriminación.

4.1.2.2.4 Matrix R G B

Esta matrix está formada por tres de registros de desplazamiento uno por cada color (Reg_desp_R, Reg_desp_G, Reg_desp_B) que a su vez están formados por 16 registros en cascada de 8 x 8 bits, es decir, 16 registros con 8 bits de entrada y 8 bits de salida ambas en paralelo. La entrada del segundo registro es la salida del primero, la entrada del tercer registro es la salida del segundo, y así sucesivamente hasta el último registro, formando un registro de desplazamiento de 16 salidas en paralelo de 8 bits cada una. En estos registros es donde se almacenan los 16 valores anteriores de RGB recibidos por el sensor RGB (R_in, G_in, B_in), los cuales son la entrada al módulo media RGB el cual se encarga de calcular el valor medio.

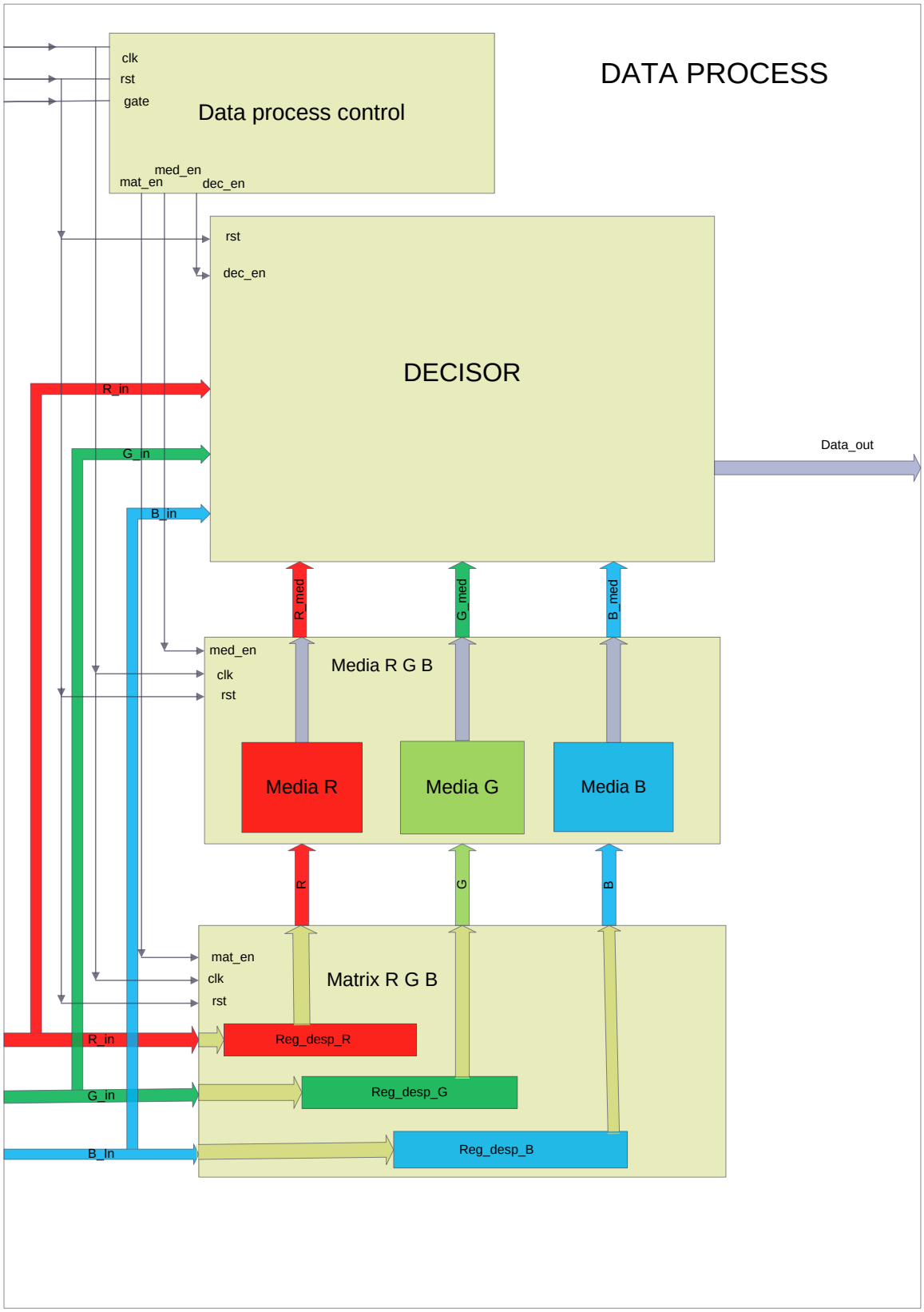


Figura 4-3. Diagrama de bloques del DATA PROCESS

4.1.3 Implementación: *Software*

Cada uno de los módulos han sido implementados con el lenguaje de descripción hardware VHDL (Very Hard Descripcion Languaje). Este lenguaje junto con Verilog son los más empleados en el diseño de sistemas digitales. Permite modelar, documentar, simular, verificar y sintetizar un sistema digital, por lo que abarca un ciclo completo de diseño salvo el trazado físico o layout.

El lenguaje también permite diseñar con facilidad bancos de prueba (test-bench), con los cuales se lleva a cabo la simulación de los sistemas modelados.

Por este motivo es por lo que hemos optado por este lenguaje, además de haberlo estudiado durante la carrera. En la parte de anexos de esta memoria podemos encontrar todo el código VHDL utilizado para implementar cada uno de los bloques de los que se compone nuestro diseño.

4.1.4 Implementación: *Hardware*

Para la implementación del receptor hemos hecho uso de una placa para implementación de circuitos digitales Diligent la cual hace uso de la FPGA Spartan 3-1000 de Xilinx y que podemos ver en las figuras 4-4 y 4-5. En esta placa se implementa en su totalidad el receptor excepto el sensor el cual está montado sobre una protoboard.

Uno de los motivos para el uso de esta placa es porque reúne todas las condiciones necesarias para la implementación del receptor, el segundo motivo no menos importante es que al disponer de una no ha sido necesaria la compra de otra similar.

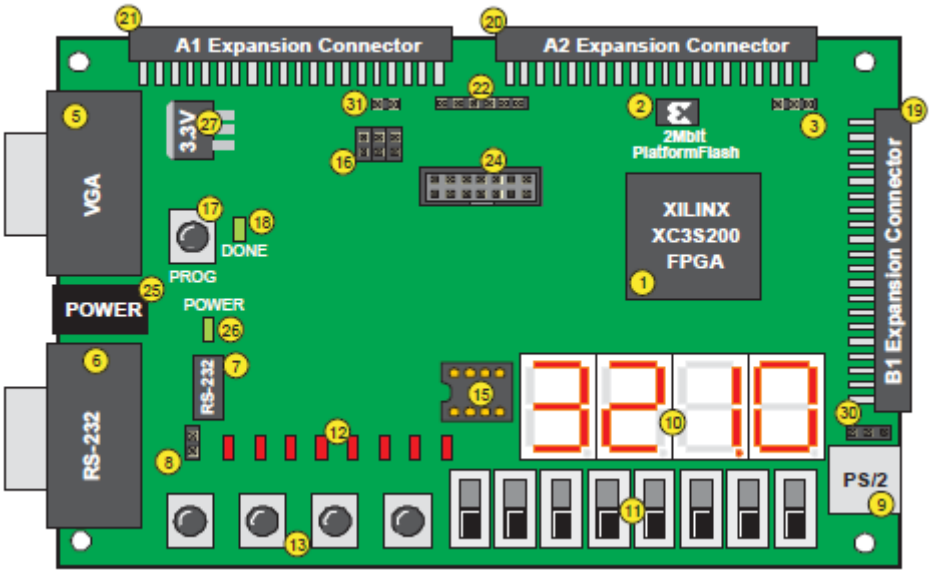


Figura 4-4. Xilinx Spartan-3 Starter Kit Board (Top Side)

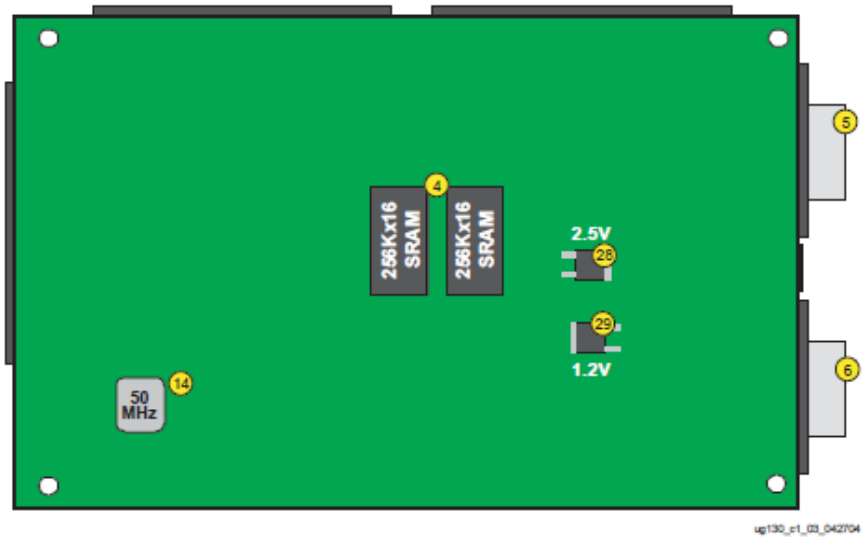


Figura 4-5. Xilinx Spartan-3 Starter Kit Board (Bottom Side)

Para la captación de los diferentes colores (rojo, verde y azul) y ponderación de los mismos utilizamos un circuito integrado de la casa Hamamatsu modelo

S9706 que podemos ver en la figura 4-6. El motivo de la lección de este tipo de sensor porque es que su salida está en formato digital lo cual es una ventaja en cuanto a la simplificación del diseño con respecto a los fotodiodos convencionales a los cuales habría que añadirles una etapa de conversión análogo-digital para el posterior procesado de las señales y en cuanto a este en concreto con respecto al CCD y CMOS por su mayor velocidad de respuesta y su menor coste.

Este está formado por una matriz de 9 x 9 fotodiodos cuando se usa en el modo de alta sensibilidad, en el modo de baja sensibilidad esta matriz se reduce a 3 x 3 fotodiodos tal y como se muestra en la figura 4-7.

Figura 4-6. Sensor RGB S9706

Sensitivity setting

Range	Mode	Effective photosensitive area*
High	High sensitivity	9 × 9 elements
Low	Low sensitivity	3 × 3 elements

* The photosensitive area of S9706 consists of 9 × 9 elements in a mosaic pattern.
The effective photosensitive area changes depending on which sensitivity mode is used, "high" or "low", as explained below.
· High sensitivity mode: 9 × 9 elements
· Low sensitivity mode: 3 × 3 elements in center

Details of photosensitive area (unit: µm)

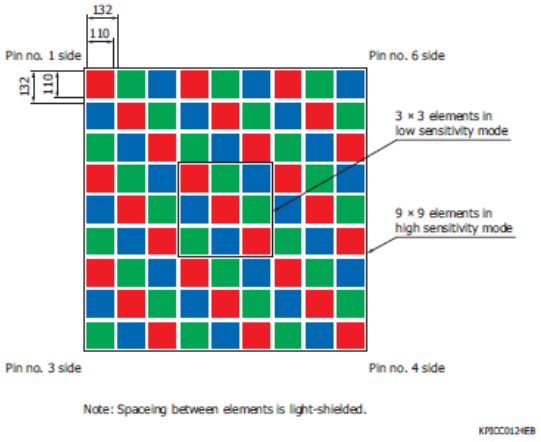


Figura 4-7. Detalles del área fotosensible

4.1.4.1 Funcionamiento del sensor

La potencia de las diferentes señales (CSK) generadas por el transmisor son guardadas en tres registros internos al sensor (los cuales podemos apreciar en el esquema de bloques de la Figura 4-8), uno por cada color. Cada uno de estos registros consta de doce bits los cuales son transmitidos en serie a través de la salida ***"Dout"*** después de la habilitación de la señal ***"Gate"*** e introduciendo 3 series de 12 pulsos por la línea de ***"CK"***.

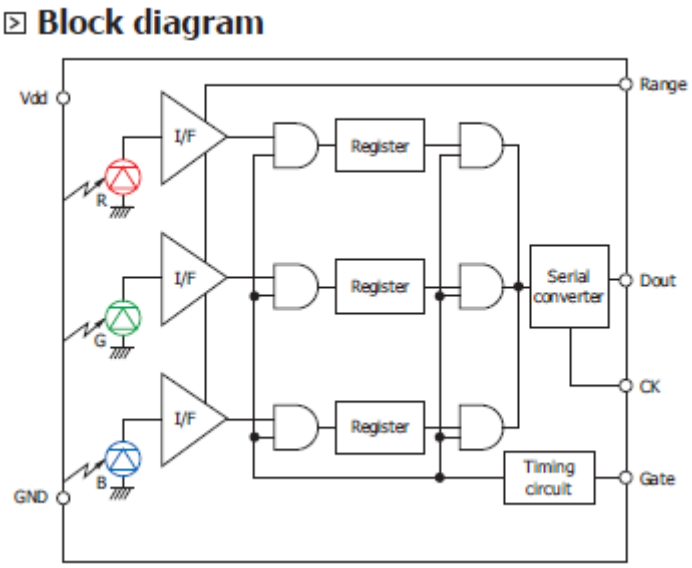


Figura 4-8. Esquema de bloques del sensor S9706

Para el correcto funcionamiento del sensor, debemos seguir los siguientes pasos:

1. Poner el terminal "***Ck"*** y ***"Gate"*** a nivel bajo.
2. Seleccionar la sensibilidad con el terminal ***"Range"***.
3. Pasar el terminal ***"Gate"*** de nivel bajo a alto para empezar con la integración del nivel de intensidad de luz.
4. Después del tiempo de integración ***'tg'***, pasamos el terminal de ***"Gate"***de nivel alto a bajo para terminar con el tiempo de integración
5. Se introducen los pulsos ***"Ck"*** para la obtención de los valores de salida.

6. La medida de la intensidad de luz de cada color la podemos obtener a través del terminal ***"Dout"***.
7. Al finalizar la lectura de datos volvemos a pasar el terminal ***"Gate"*** de un nivel bajo a alto.

Son necesarios un total de 36 pulsos ***"Ck"*** para leer el valor de los tres colores. La medida del rojo lo obtenemos con los 12 primeros pulsos, el verde por los siguientes 12 pulsos y el azul por los 12 últimos pulsos.

La salida de datos se hace desde el bit menos significativo (LSB). Los valores de salida son obtenidos con los flancos de subida dela señal ***"C"***.

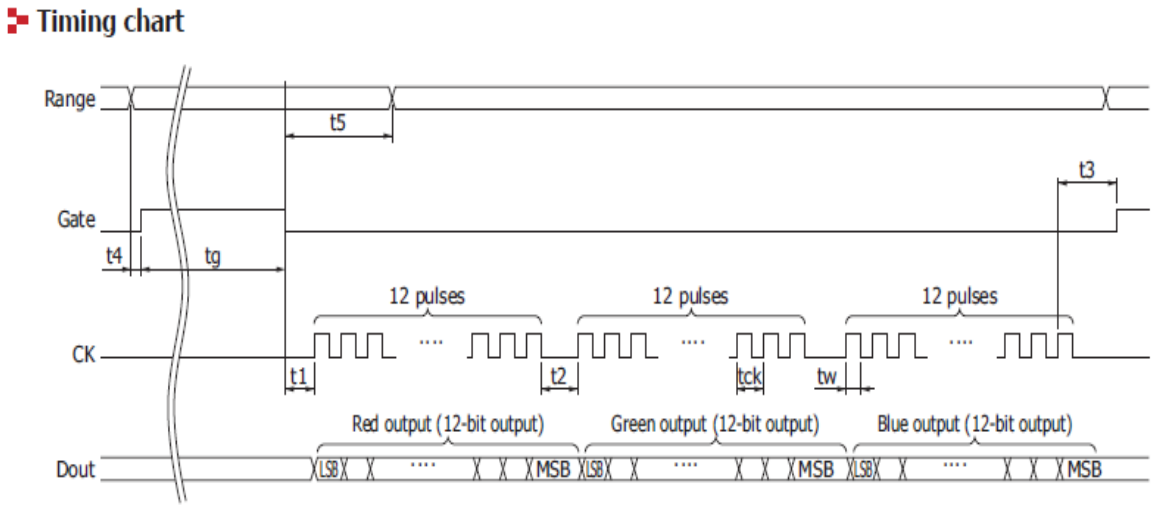


Figura 4-9. Diagrama de tiempos

Hay que respetar los tiempos mínimos ***'tg', 't1', 't2', 't3', 't4', 't5', 'tw'*** y ***'tck'*** que podemos ver en la figura 4-10.

.

Electrical and optical characteristics
(Ta=25 °C, Vdd=5 V, Tg=100 ms, A light source, unless otherwise noted)

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Photosensitive area size	-	All elements (9 × 9 elements)	-	1.2 × 1.2	-	mm
Effective photosensitive area	-	Per 1 color, High range	-	0.32	-	mm^2
Spectral response range	λ	Blue	-	400 to 540	-	nm
		Green	-	480 to 600	-	
		Red	-	590 to 720	-	
Peak sensitivity wavelength	λp	Blue	-	465	-	nm
		Green	-	540	-	
		Red	-	615	-	
Supply voltage	Vdd		3.0	-	5.5	V
Current consumption	Idd	Dark state, no load	-	5	10	mA
Photosensitivity	Sbl	Blue, Low range	0.15	0.21	0.27	LSB/lx
	Sgl	Green, Low range	0.32	0.45	0.59	
	Srl	Red, Low range	0.45	0.64	0.83	
	Sbh	Blue, High range	1.3	1.9	2.5	
	Sgh	Green, High range	2.8	4.1	5.4	
	Srh	Red, High range	4.0	5.8	7.6	
Incident light power (Conversion value in A light source)	Ibl	Blue, Low range	-	-	240	klx
	Igl	Green, Low range	-	-	110	
	Irl	Red, Low range	-	-	78	
	Ibh	Blue, High range	-	-	26	
	Igh	Green, High range	-	-	12	
	Irh	Red, High range	-	-	8.6	
Dark output	Dark	Tg=0.5 s	-	-	1	LSB
Input high level	Vih		Vdd × 0.82	-	-	V
Input low level	Vil		-	-	Vdd × 0.18	V
High level output voltage	Voh	Ioh=-0.5 mA	4.5	-	-	V
Low level output voltage	Vol	Iol=0.5 mA	-	-	0.5	V
Integration time	Tg		Refer to "Output vs. illuminance"			-
Hold time	t1		4	-	-	µs
	t2		3	-	-	µs
	t3		3	-	-	µs
	t4		2000	-	-	µs
	t5		3	-	-	µs
Readout clock period	tck		500	-	-	ns
Readout pulse width (positive)	tw		200	-	-	ns
Readout pulse width (negative)	tck-tw		200	-	-	ns

Figura 4-10. Características eléctricas y ópticas

La frecuencia teórica máxima de trabajo corresponde con los tiempos mínimos ***'t1', 't2', 't3', 't4', 't5', 'tw'*** y ***'tck'***. Como en nuestro caso el terminal de ***"Range"*** lo tenemos a nivel alto (***"Vdd"***) siempre para tener la máxima sensibilidad del sensor no vamos a tener en cuenta los tiempos de ***'t4'*** y ***'t5'***.

En este caso la frecuencia máxima de funcionamiento correspondería con la fórmula:

$$Fmax = tg + t1 + (36\ x\ tck) + (2\ x\ t2) + t3$$

Para los valores de alimentación ***"Vdd"***='*5v*' y un tiempo de integración de ***'tg'***=100 ms y máxima sensibilidad ***("Range"="Vdd")***, los tiempos mínimos indicados por el fabricante son;

- ***'t1'***=4 *µs*
- ***'t2'='t3'***=3 *µs*
- ***'tck'=****0,5 µs*

la frecuencia máxima de trabajo en estas condiciones sería:

$$Ttotal = (100\ x\ 10^3 \mu s) + 4\ \mu s + (36\ x\ 0.5\ \mu s) + (2\ x\ 3\ \mu s) + 3\ \mu s$$
$$= 100.031\ \mu s\ \approx 100\ ms$$

$$Fmax = \frac{1}{Ttotal}\ \approx 10\ Hz$$

Como podemos observar la frecuencia máxima depende casi exclusivamente del tiempo de integración **'*tg*'**. Para tiempos de integración menores se pueden obtener frecuencias de trabajo mayores a costa de una menor sensibilidad según vemos en la figura 4-11.

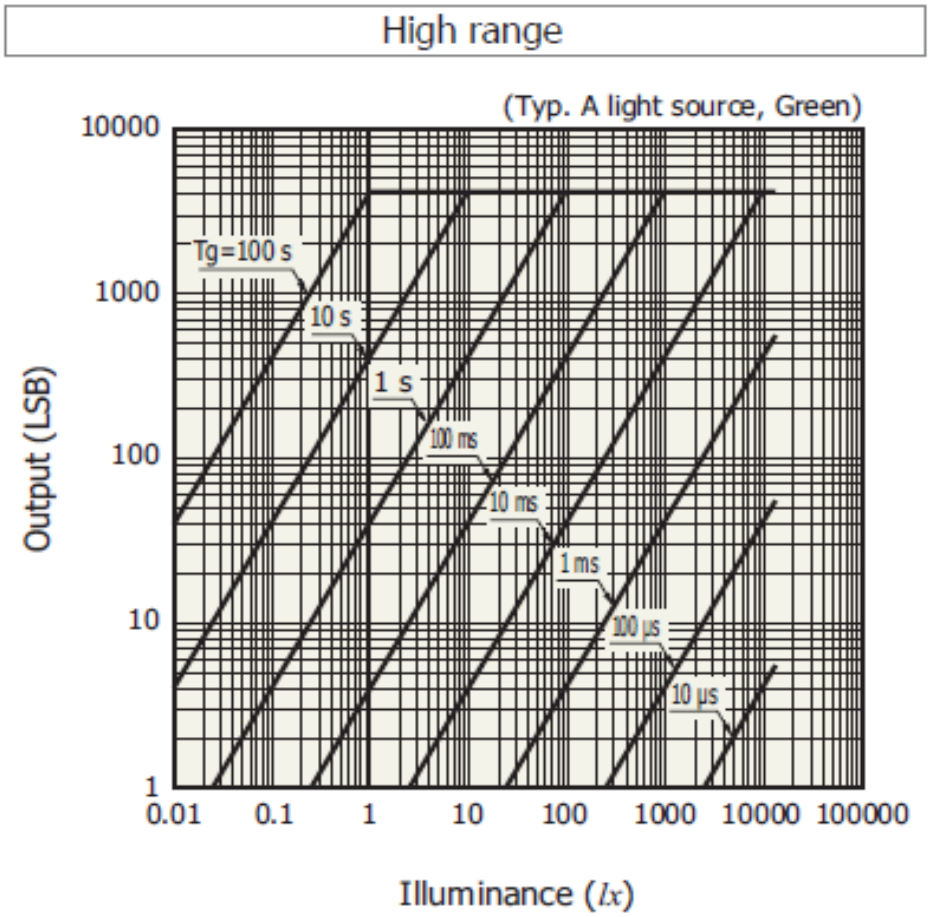


Figura 4-11. Iluminancia mínima para distintos tiempos de integración

4.1.5 Implementación: Herramientas

Como herramienta de implementación para el 'volcado' sobre la placa de nuestro diseño hemos utilizado "ISE project navigator" versión 14.6 cuya interfaz podemos ver en la figura 4-12. Esta es una herramienta de Xilinx con la cual podemos programar cualquier FPGA fabricada por Xilinx.

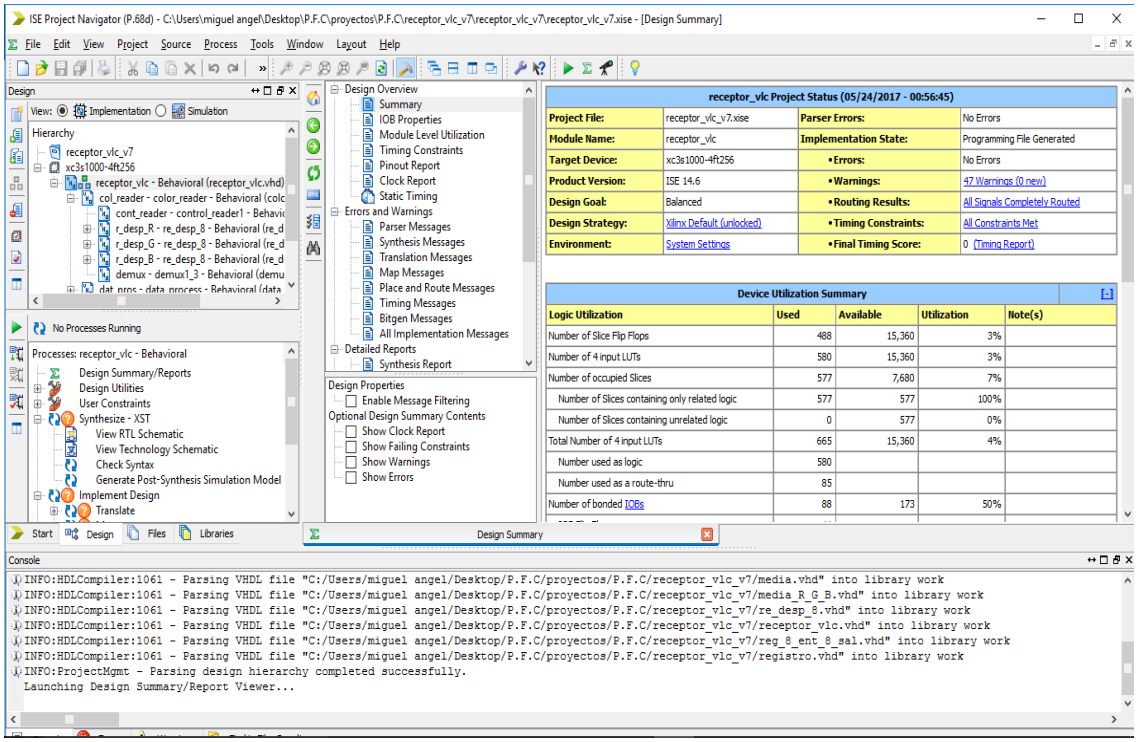


Figura 4-12. Interfaz de la herramienta ISE

Una descripción exhaustiva de la herramienta se sale de nuestro propósito, por lo tanto sólo haremos una pequeña introducción explicando únicamente lo más relevante.

En el interfaz de la herramienta podemos observar varias ventanas. En la primera de la izquierda podemos ver el tipo de dispositivo empleado, que en este caso es una FPGA modelo xc3s1000-4ft256 y la jerarquía del diseño según se muestra en la figura 4-13.

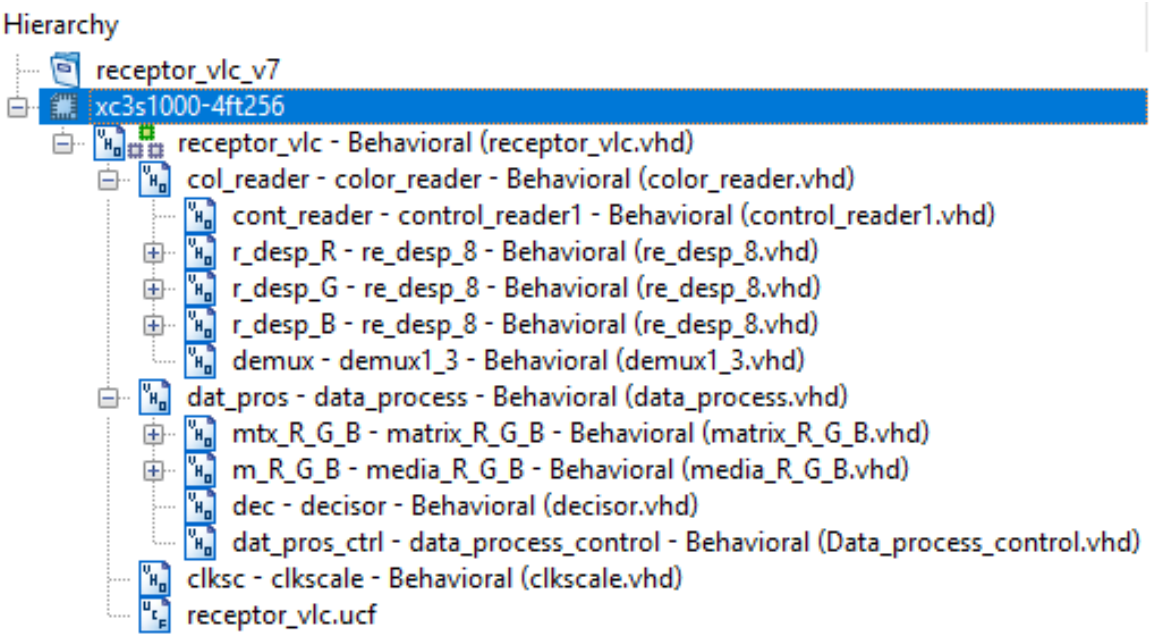


Figura 4-13. Jerarquía del diseño

Podemos observar que la jerarquía más alta en nuestro diseño es 'receptor_vlc'. Vemos que está formada por 2 módulos principales el de color reader y el de data process, A su vez cada uno de estos está formado por otros en un nivel de jerarquía inferior y que corresponden con los descritos en el apartado 4.1.2.1 y 4.1.2.2 del presente capítulo.

En la ventana inmediatamente inferior a esta que podemos ver en la figura 4-14, se muestran una serie de utilidades. Entre ellas cabe destacar:

- "*Desing Summary/Repor"*
- *"Synthesize"*
- *"Implemet Desing"*
- *"Generate Programming File"*

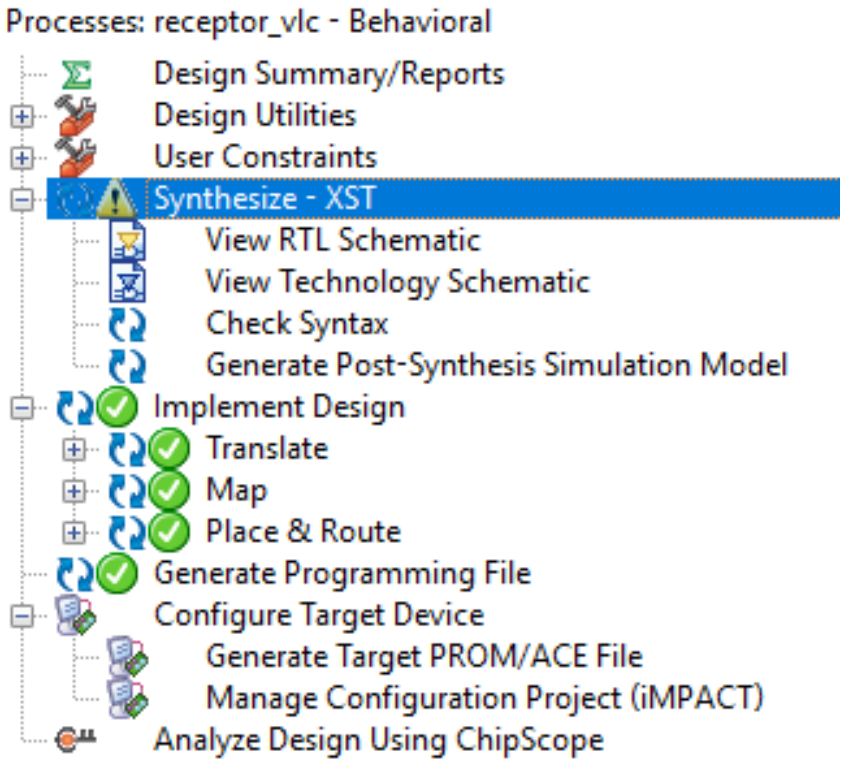


Figura 4-14. Utilidades del diseño

4.1.5.1 "Desing summary reports"

Pinchando con el puntero del ratón en esta línea, se muestra en la parte superior derecha de la herramienta un resumen con todos los recursos utilizados por la FPGA para llevar a cabo la implementación, así como la disponibilidad de los mismos como apreciamos en la figura 4-15.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	488	15,360	3%	
Number of 4 input LUTs	580	15,360	3%	
Number of occupied Slices	577	7,680	7%	
Number of Slices containing only related logic	577	577	100%	
Number of Slices containing unrelated logic	0	577	0%	
Total Number of 4 input LUTs	665	15,360	4%	
Number used as logic	580			
Number used as a route-thru	85			
Number of bonded IOBs	88	173	50%	
IOB Flip Flops	11			
Number of BUFGMUXs	4	8	50%	
Average Fanout of Non-Clock Nets	2.21			

Figura 4-15. Recursos utilizados de la FPGA

4.1.5.2 "Synthesize"

Al hacer doble 'click' con el ratón sobre esta línea comienza la síntesis del diseño. Si todo es correcto nos mostrará un 'tick' de color verde en esta línea y en la consola de abajo se indicará que todo el proceso se ha desarrollado satisfactoriamente.

En caso de haber algún warning, cambiará el 'tick' de color verde por un triángulo con un símbolo de exclamación dentro de un fondo amarillo. Ello nos indica que tenemos que tener cuidado porque hay algo inesperado, como pueden ser conexiones abiertas, salidas que no están conectadas a alguna entrada, ect,….Habría que revisar el informe de la síntesis y ver dónde están los warnings y si en verdad son motivo de preocupación.

En caso de que la síntesis sea imposible ya sea porque no se han tenido en cuenta las reglas de diseño, o intentar sintetizar por ejemplo una sentencia de 'wait' la cual no es directamente implementable, nos mostrará un mensaje de error en la consola y aparecerá un símbolo similar a un "prohibido el paso" de una señal de tráfico.

4.1.5.3 *"Implemet Desing"*

Es en esta parte donde se llevan a cabo las tareas de *"Translate", "Map", "Place & Route"*. En esta etapa se realiza fundamentalmente la asignación de cada uno de los recursos de la FPGA necesarios para la implementación del diseño, su ubicación dentro de la FPGA y el conexionado entre ellos.

Es un proceso que se realiza totalmente automático, si bien podemos poner algunas restricciones a la herramienta y también podemos conectar cualquier parte del circuito con los conectores de expansión (A1, A2, B1), necesitando para ello editar el fichero 'receptor_vlc.ucf'. Este archivo se muestra en la figura 4-16.

Podemos ver una descripción detallada de los mismos en la hoja de anexos (Descripción básica de la placa Digilent).

```
1  CONFIG PROHIBIT=M14;   ## BTN1
2  CONFIG PROHIBIT=M13;  ## BTN0
3
4  CONFIG PROHIBIT=K13;  ## SW7
5  CONFIG PROHIBIT=K14;  ## SW6
6  CONFIG PROHIBIT=J13;  ## SW5
7  CONFIG PROHIBIT=J14;  ## SW4
8  CONFIG PROHIBIT=H13;  ## SW3
9  CONFIG PROHIBIT=H14;  ## SW2
10 CONFIG PROHIBIT=G12;  ## SW1
11 CONFIG PROHIBIT=F12;  ## SW0
12
13 NET rst LOC=L14;      ## BTN3
14 NET clk LOC=T9;
15
16 NET gate LOC=A13;
17 NET ck LOC=A12;
18 NET sinc_in LOC=L13;  ##E6 salida 4 del B2-- BTN2 L13
19 NET data_in LOC=B10;
20
21 NET data_out_R_CR<7> LOC=B5;
22 NET data_out_R_CR<6> LOC=B4;
23 NET data_out_R_CR<5> LOC=D10;
24 NET data_out_R_CR<4> LOC=D8;
25 NET data_out_R_CR<3> LOC=D7;
26 NET data_out_R_CR<2> LOC=E7;
27 NET data_out_R_CR<1> LOC=D6;
28 NET data_out_R_CR<0> LOC=D5;
29
30 NET data_out_G_CR<7> LOC=A5;
31 NET data_out_G_CR<6> LOC=A4;
32 NET data_out_G_CR<5> LOC=A3;
```

Figura 4-16. Fichero UCF

4.1.5.4 "Generate programming file"

Este es el último paso dentro de la síntesis del circuito. Aquí es donde se genera el fichero para la programación de la FPGA. Para poder cargar dicho fichero y programar la FPGA necesitamos lanzar la herramienta iMPACT la cual también nos permite conectamos a la placa de prototipado mediante un cable diseñado para tal propósito conectando un puerto llamado JTAG de la placa y el puerto USB del ordenador.

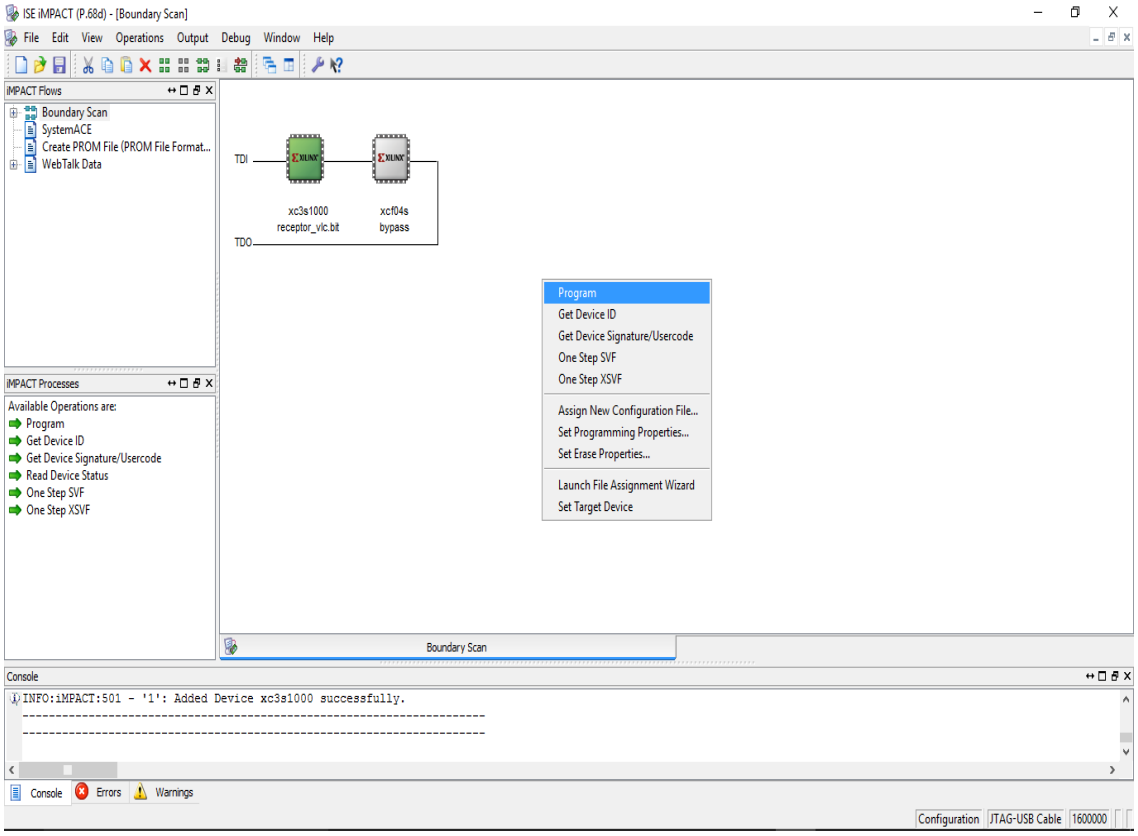


Figura 4-17. Herramienta iMPACT

Haciendo doble 'click' sobre "*Boundary Scan*" la herramienta se 'conecta' con la placa a través del cable JTAG-USB, encontrando los dispositivos montados sobre la placa. En este caso podemos ver cómo ha detectado la presencia de la FPGA y su modelo exacto. Para iniciar la programación primero tenemos que cargar el fichero de programación que se encuentra en el directorio donde se han guardado los ficheros del diseño el cual tiene una extensión "fichero.bit".

Después de cargar este fichero ya podemos programar la FPGA. Para ello sólo tenemos que pulsar el botón derecho del ratón y hacer ‘click’ en “*program*”.

Capítulo 5

Simulaciones

5. Simulaciones

5.1 Introducción

En este capítulo vamos a repasar todas las simulaciones del conjunto receptor, como las de cada uno de los módulos que lo forman. Primero empezaremos por la simulación del bloque 'COLOR READER' y después lo haremos con el bloque 'DATA PROCESS'. Seguidamente comprobaremos el funcionamiento del receptor VLC completo.

Para la simulación empleamos el software de simulación de circuitos digitales ModelSim PE Student Edition 10.4ª que es propiedad de Mentor Graphics. Hemos elegido este software porque trabaja sobre VHDL, además de ser gratuito para la versión estudiante. Aunque la licencia de estudiante es útil por un período de 3 meses, pasados estos podemos renovarla sin coste alguno las veces que necesitemos.

5.2 Descripción del simulador

Este simulador permite la creación de proyectos a los que les vamos añadiendo nuevos archivos u otros ya creados escritos en lenguaje VHDL (entre otros). Estos archivos constan de una entidad y una arquitectura. La entidad representa un módulo con unas entradas y salidas determinadas, mientras que la arquitectura es una descripción de lo que hace dicha entidad (descripción funcional).

Cada una de estos archivos representa un módulo del circuito que vamos a simular. En la figura 5-1 podemos ver en la ventana izquierda los módulos (archivos) de los que consta el receptor, la fecha de creación y hora, mientras que en la ventana derecha podemos ver el código del archivo que tenemos abierto en ese momento.

En la barra superior tenemos varios botones con diferentes funciones, las más importantes son "*Compile*" y "S*imulate*". Con este último botón podemos simular nuestro proyecto obteniendo una nueva ventana llamada "*Wave*", donde podemos observar las diferentes señales en la parte de la izquierda y sus formas

de onda en la parte derecha según se puede apreciar en la figura 5-2. Los valores que adquieren dichas señales se pueden mostrar en varios formatos (binario, hexadecimal, decimal,…).

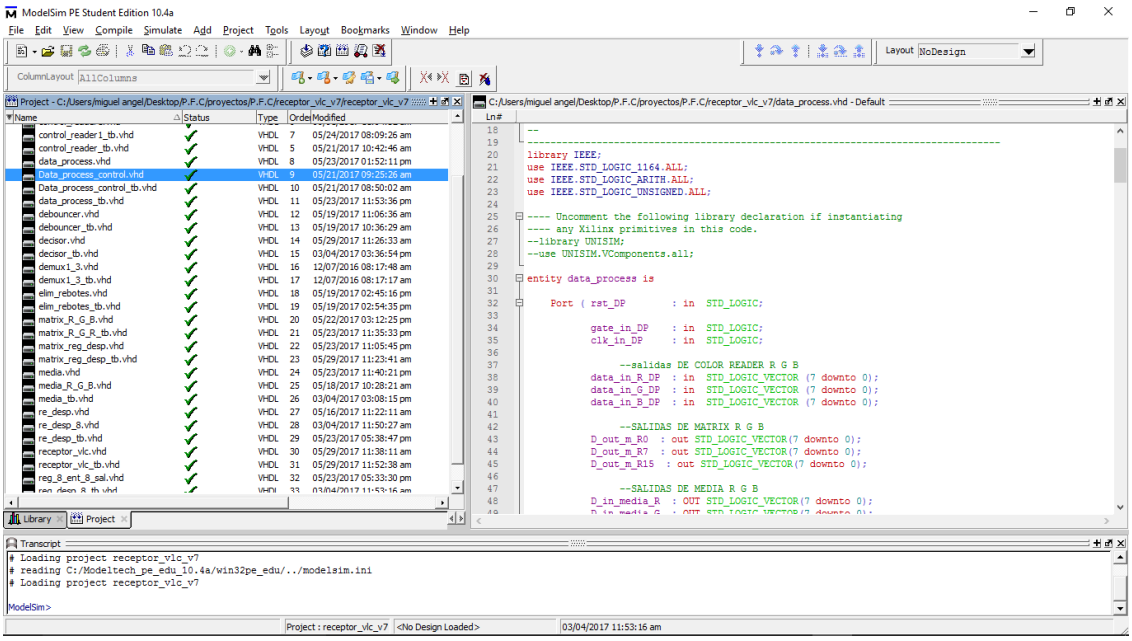


Figura 5-1. Simulador ModelSim

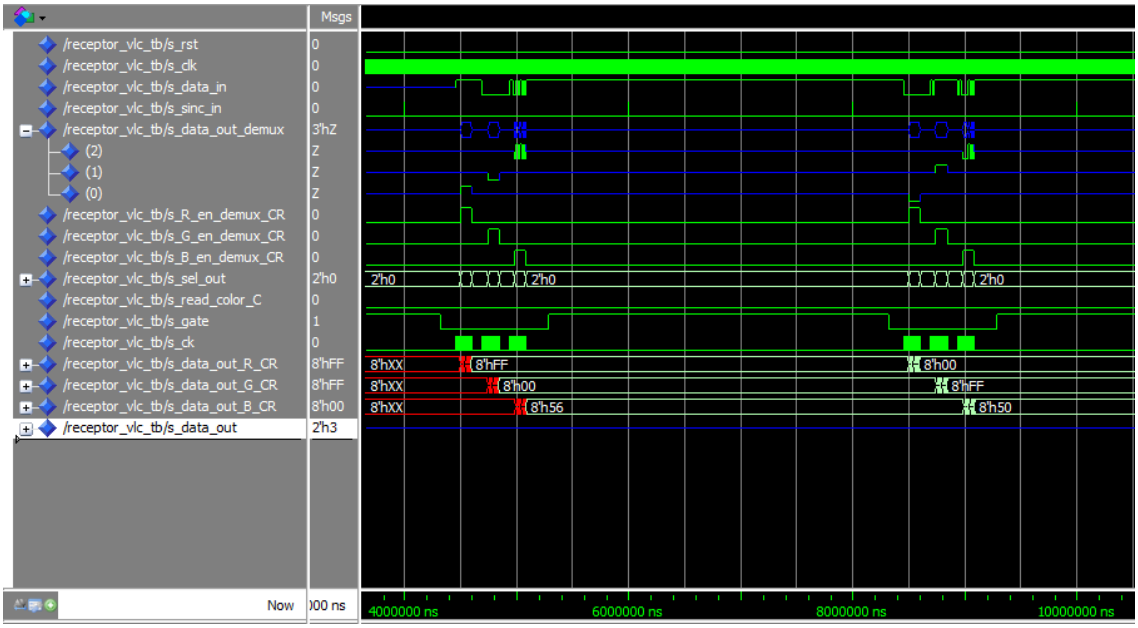


Figura 5-2. Diferentes señales y sus formas de onda

5.3 Receptor VLC

Una vez explicada brevemente la herramienta vamos a mostrar los resultados de la simulación de nuestro receptor.

5.3.1 Extracción de datos desde el sensor (COLOR READER)

Para poder extraer los valores de rojo, verde y azul, tenemos que enviar distintas señales al sensor RGB como ya se indicó en el capítulo 4, sección 4.1.3.1 (funcionamiento del sensor).

En la figura 5-3, podemos ver cómo bajamos primero la señal de "***Gate***" *('s_gate'*), tras lo cual enviamos por la línea "***CK"*** (*'s_ck'*) 36 pulsos, 12 por cada color como ya habíamos explicado en el capítulo anterior. Tras estos 36 pulsos la señal de ***"Gate"*** pasa a nivel alto y se termina con ello un ciclo de lectura.

Esta operación de lectura se lleva a cabo cada vez que se envía una señal de sincronismo desde el transmisor, la cual nos indica que hay nuevos datos que leer.

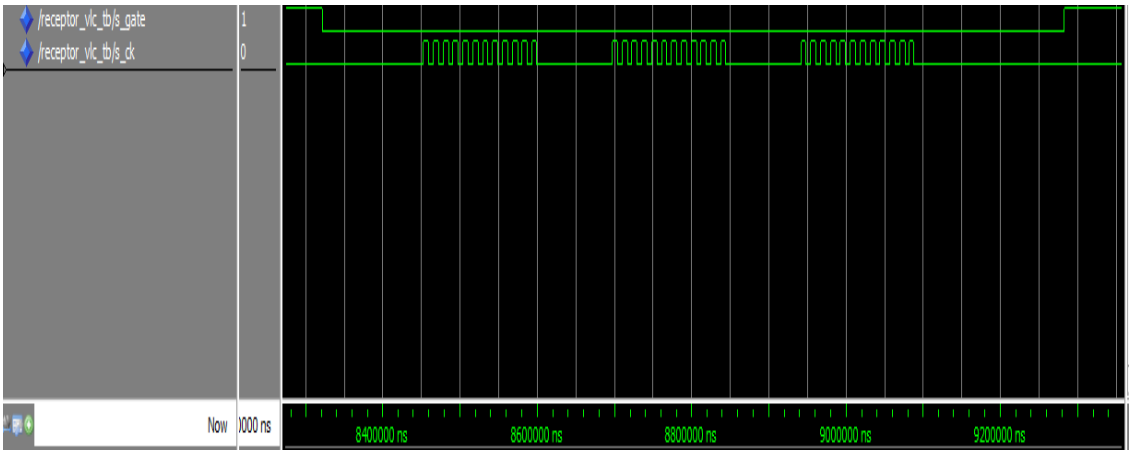


Figura 5-3. Señales "***Gate"*** y ***"CK"***

5.3.2 Obtención de datos en los registros R, G, B

Los datos obtenidos desde el señor RGB pasan al receptor a través de la línea ***"Data_in"*** a la entrada de un demultiplexor 3_1 como se puede apreciar en la figura 5-4. Es este multiplexor el encargado de dirigir los diferentes valores de rojo, verde y azul a sus respectivos registros haciendo uso para ello de la señal ***"sel"***(*'s_sel_out'* en la simulación de la figura 5-5), que es generada por el control reader.

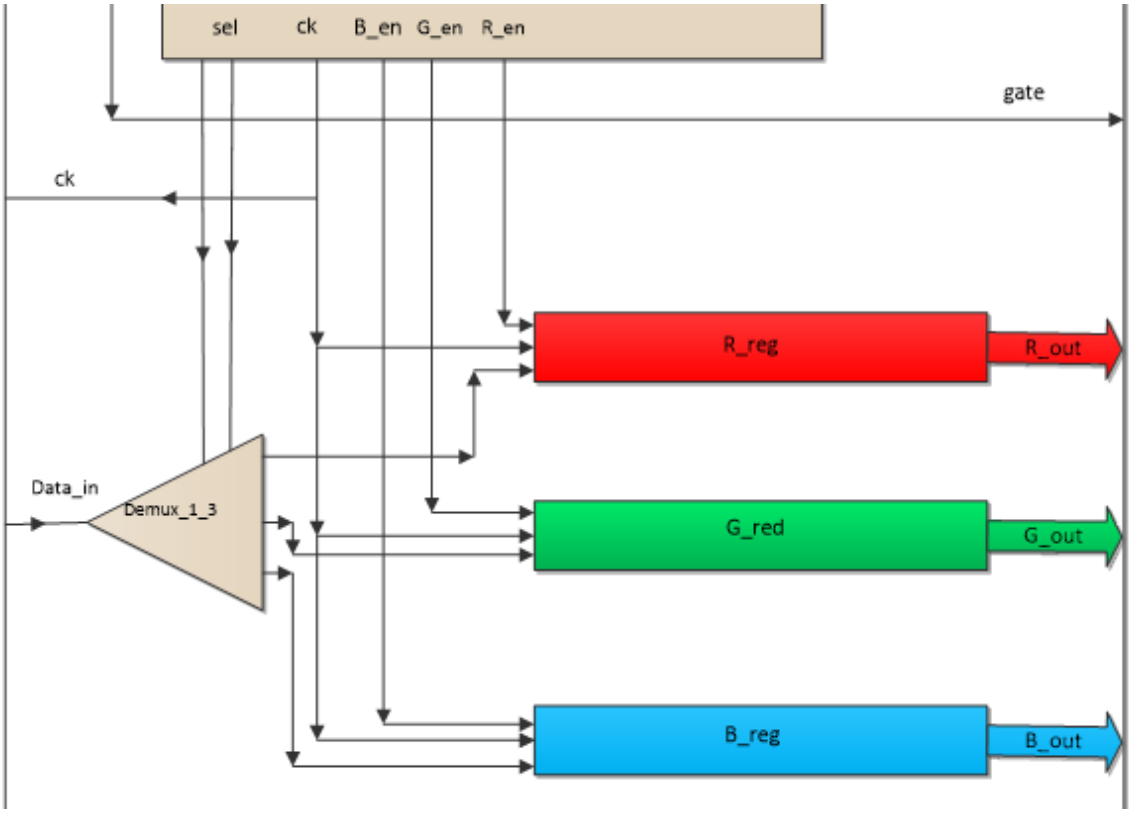


Figura 5-4. Detalle del demultiplexor y registros RGB

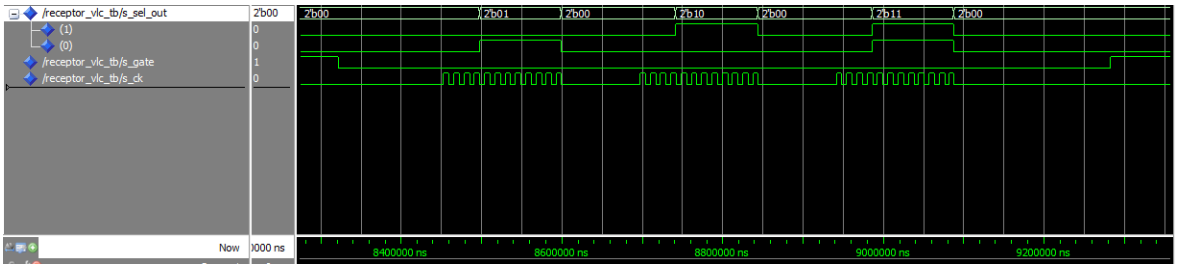


Figura 5-5. Salida *'s_sel_out'*

En la figura 5-6 podemos ver el resultado a las salidas de cada uno de los registros en función de los datos de entrada *('s_data_in'*) así como los diferentes valores de las demás señales implicadas en el funcionamiento general del módulo de lectura COLOR READER.

La salida del registro R_reg (rojo) está señalada como *'s_data_out_R_CR'*, la del registro G_reg (verde) como '*s_data_out_G_CR*' y la del B_reg (azul) como *'s_data_out_B_CR'* donde CR hace referencia al módulo al que pertenece (COLOR READER).

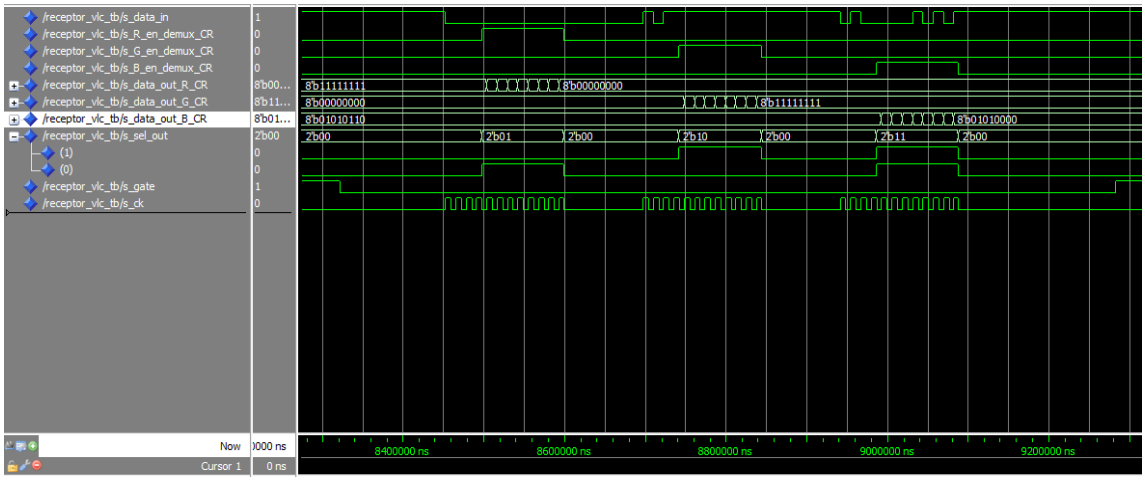


Figura 5-6. Salidas de los registros en función de las entradas

La salida de estos registros correspondesn con la entrada al módulo DATA PROCESS. En cada uno de ellos se almacena el valor del rojo, verde y azul representados con 8 bits.

El valor inicial de los 12 bits que obtenemos del sensor (por cada color) se han truncado a 8 bits por la razón de que en el siguiente módulo (DATA PROCESS) tiene que hacer la media lo que incluye la operación de división, si bien esta operación es soportada por el simulador, a la hora de la síntesis del circuito no es posible la división por valores diferentes de potencias de 2 (2,4,8,…) por lo que se ha optado a trucar los 12 bits iniciales recogidos por el sensor a los 8 bit más significativos (MSB) dados por este.

5.3.3 Procesado de los datos R, G, B (DATA PROCESS)

Los valores obtenidos de rojo, verde y azul en el módulo COLOR READER (registros R_reg, G_reg y B_reg), van pasando a unos registros de desplazamiento (uno por cada color como se aprecia en la figura 5-7), en los cuales se almacenan los 16 valores anteriores de RGB.

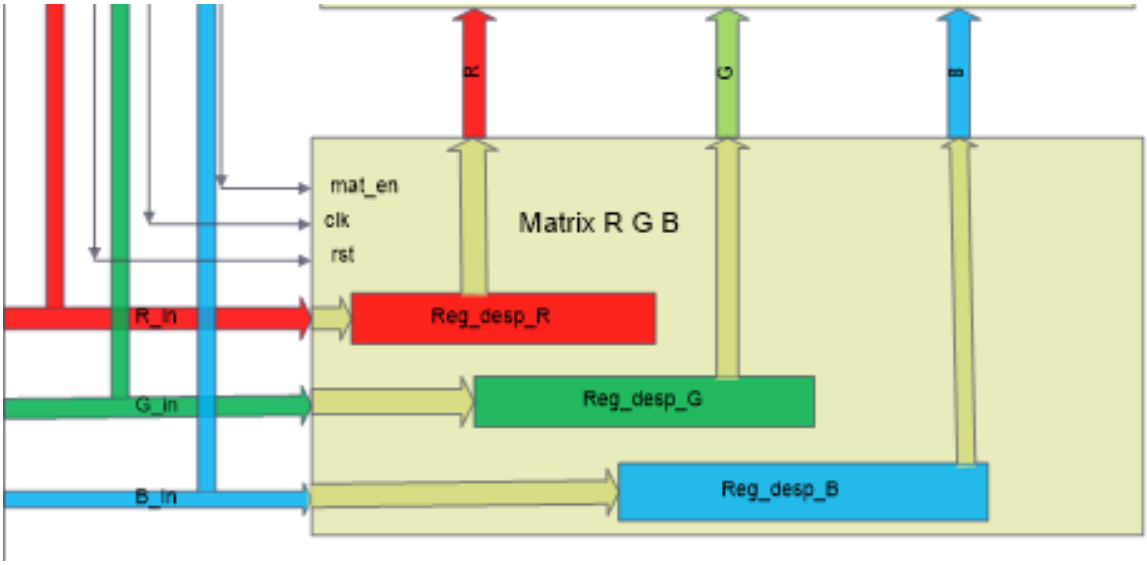


Figura 5-7. Detalle del módulo Matrix RGB

Estos registros de desplazamiento (Reg_desp_R, Reg_desp_G, Reg_desp_B), están formados por 16 registros de 8 x 8 bits que a su vez están conectados en cascada. El conjunto de todos los registros, forman el módulo Matrix R G B. Cada salida de este módulo está formado por un bus 16 x 8 líneas.

Estas líneas son las entradas al módulo Media R G B de la figura 5-8. Éste se encarga de calcular la media de todos los valores almacenados por el módulo Matrix R G B. A la salida obtenemos los valores medios de rojo, verde y azul en un bus de 8 bits por cada color (R_med, G_med, B_med).A este le sigue el módulo Decisor en el cual se decide la salida en función del valor actual

de cada color y la media de los 16 anteriores. La salida se efectúa por un bus de 2 líneas ***("Data_out"***) de las cuales extraemos uno de los 4 valores posibles transmitidos (**"00"**, **"11"**, **"10"**, **"01"**).

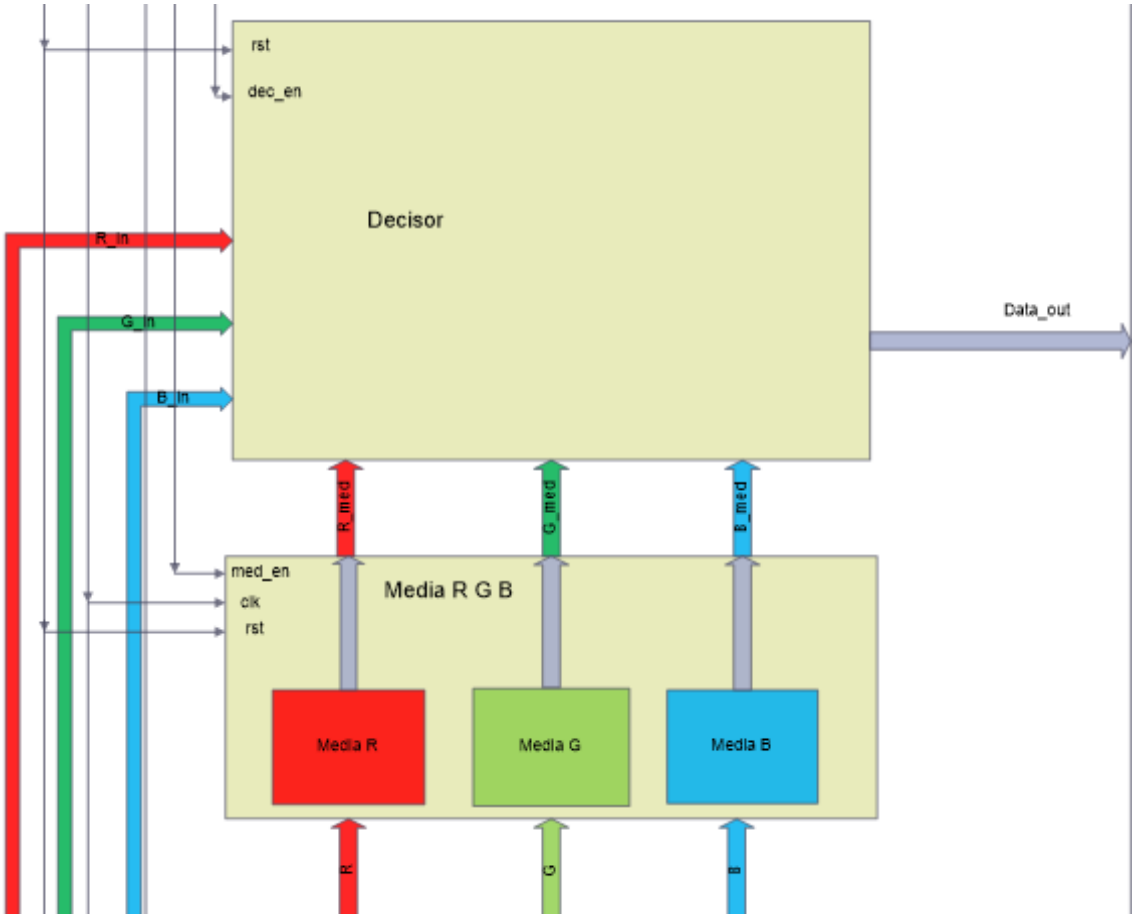


Figura 5-8. Detalle del módulo media

En la siguientes figuras podemos observar los valores a las salidas de cada uno de estos módulos así como los valores a la salida del receptor.

En la figura 5-9 podemos ver los primeros valores en los registros R_reg, G_reg, B_reg en valor hexadecimal (*'s_data_out_R_CR', 's_data_out_G_CR', 's_data_out_B_CR'*) y la salida del módulo media (D_out_m_R, D_out_m_G, D_out_m_B).

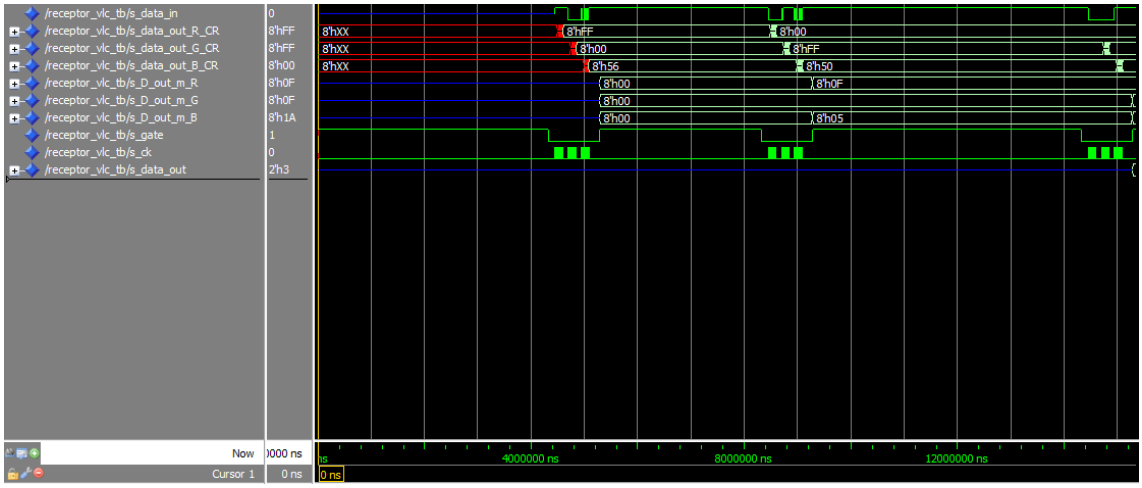


Figura 5-9. Inicio de la simulación

Evidentemente al principio de la simulación como podemos obserbar en la figura 5-9, los valores medios de R, G y B son ‘cero’ ya que los registros se encuentran ‘vacíos’. En la figura 5-10 podemos apreciar como va subiendo el valor de la media de cada uno de los colores después de los 3 primeros valores de la salida.

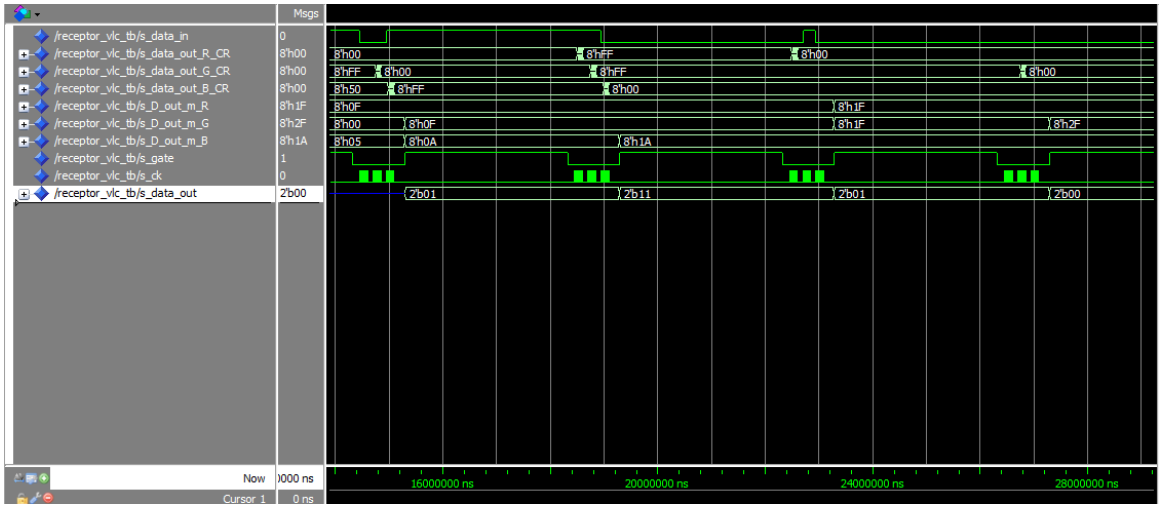


Figura 5-10. Primeros valores a la salida del receptor *(‘s_data_out’)*

Como podemos ver en la figura 5-11, después de 45 ms de simulación los valores medios se van estabilizando y observamos cómo la salida es coherente con los datos de entrada según se muestra en la lista de abajo.

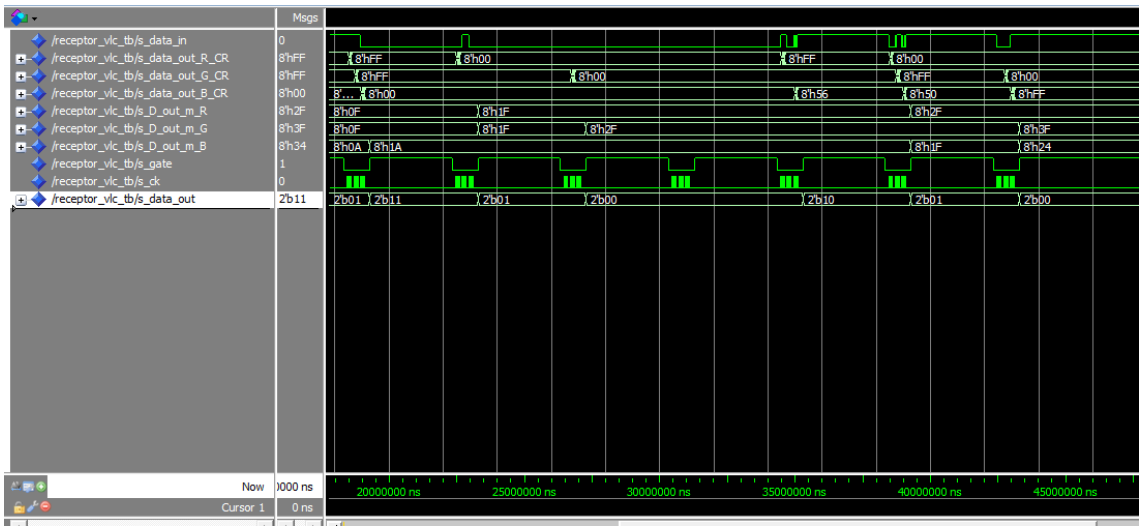


Figura 5-11. Simulación después de 45 ms

(R=”FF”, G=”FF” => ‘*S_DATA_OUT’* =”11”)

(R=”FF”, G=”00” => ‘*S_DATA_OUT*’ =”10”)

(R=”00”, G=”FF” => ‘S_DATA_OUT’ =”01”)

(R=”00”, G=”00” => ‘*S_DATA_OUT*’ =”00”)

Notas:

1. *En la simulación se han empleado valores diferentes para la codificación de los datos respecto de los empleados en la práctica.*
2. *Las señales de simulación están en cursiva y llevan como prefijo una ‘s’ seguida por un guión bajo (s_....)*
3. *Las señales del circuito están cursivas en negrita con doble comilla.*

Capítulo 6

Resultados prácticos

6. Resultados prácticos

6.1 Introducción

En este capítulo recogemos los resultados producto de cada una de las tareas prácticas llevadas a cabo en el laboratorio para la prueba del correcto funcionamiento del receptor.

6.2 Señales del sensor RGB

Para el correcto funcionamiento del sensor RGB empleado en este proyecto, se necesitan unas señales que hacen posible la lectura del valor de cada uno de los componentes R, G, B, tal y como se explicó en el capítulo 4 sección 4.1.3.1. En las siguientes gráficas captadas con un osciloscopio digital podemos ver las distintas señales del sensor RGB.

En la figura 6-1 podemos ver en el canal 1 del osciloscopio (trazo amarillo) la señal de ***"Gate"*** con la cual se activa la lectura del sensor. En el canal 2 (trazo azul) podemos ver la señal ***"CK"*** que consta de 36 pulsos, necesarios para la lectura de cada uno de los valores del color rojo, verde y azul.

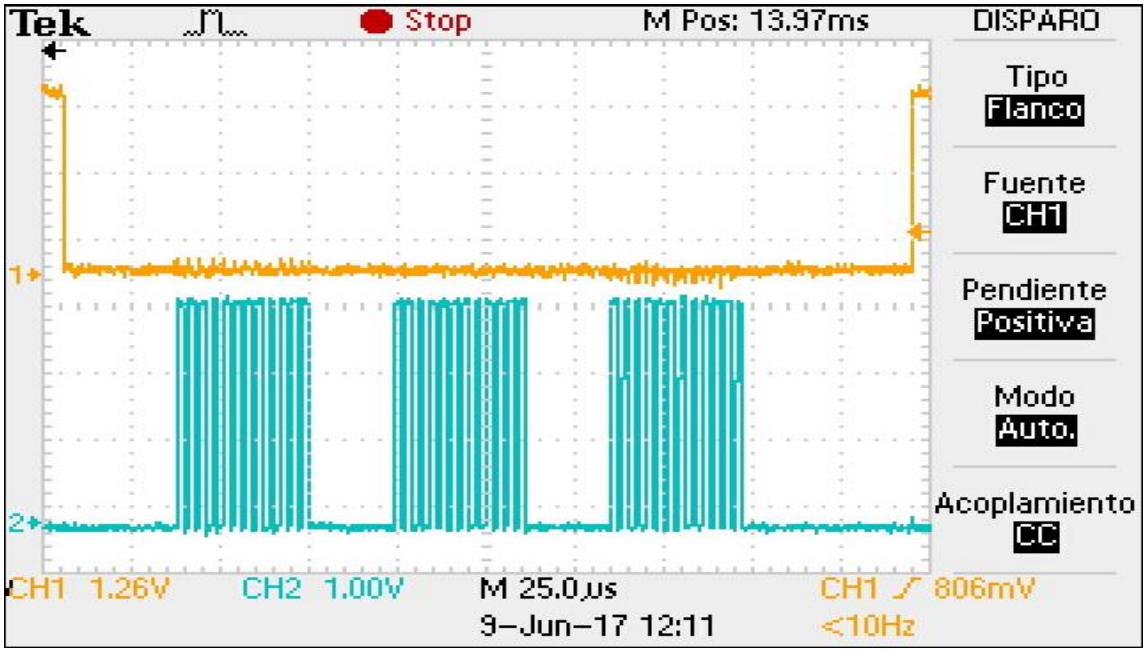


Figura 6-1. Señales de control del sensor

Los primeros 12 pulsos de la señal ***"CK"*** leen el valor del color rojo, los 12 siguientes el color verde y los 12 últimos el color azul.

En la figura 6.2 podemos ver un 'zoom' de los pulsos ***"CK"***.

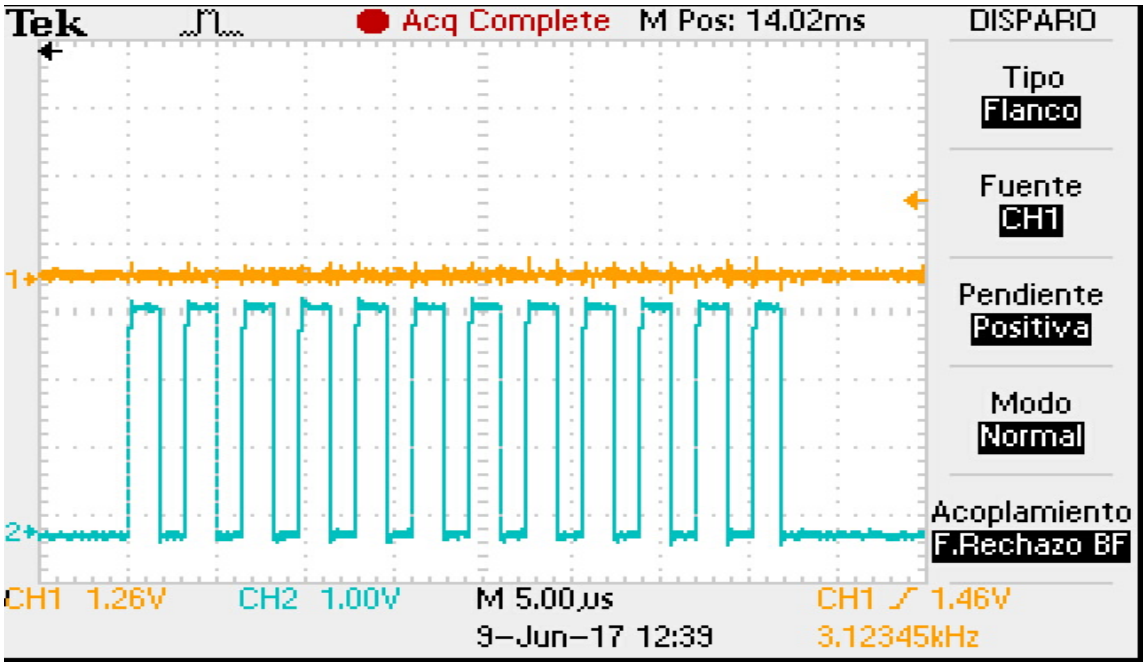


Figura 6-2. Pulsos ***'CK'***

Como podemos ver están formados por 12 pulsos, necesarios para la lectura de cada color.

En la figura 6-3 podemos ver varias secuencias de lectura, en las cuales se hacen varias medidas tanto del color rojo como del verde y el azul.

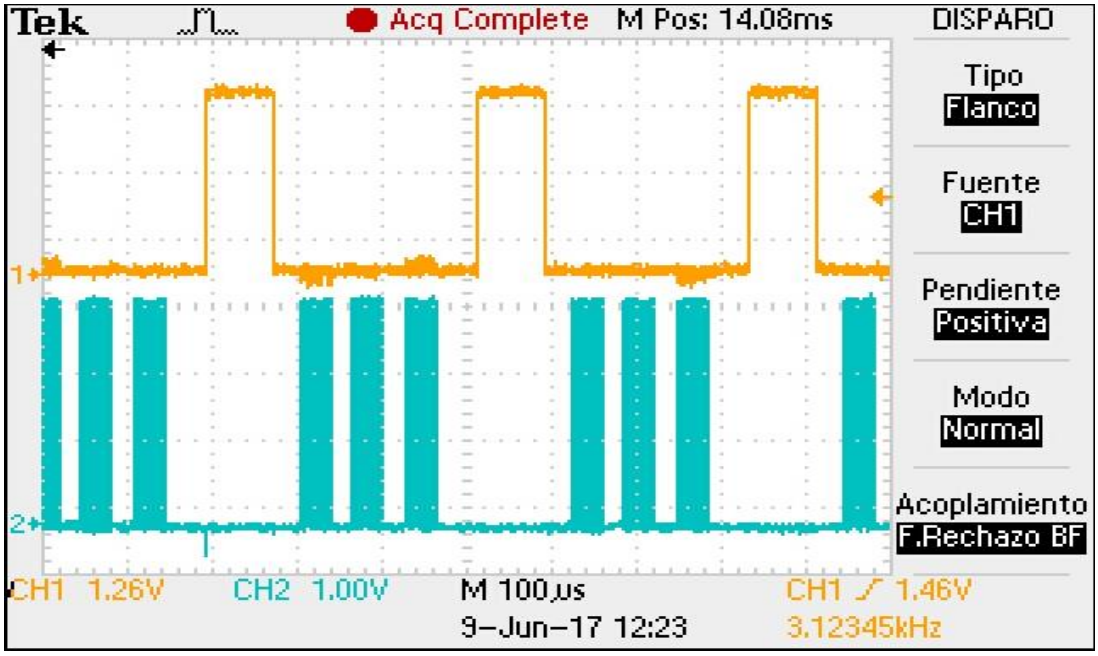


Figura 6-3. Varias señales de control del sensor

En las siguientes figuras vamos a ver las distintas salidas del sensor RGB (canal 1, trazo amarillo) para distintos colores a los que es expuesto el sensor por medio de un led RGB.

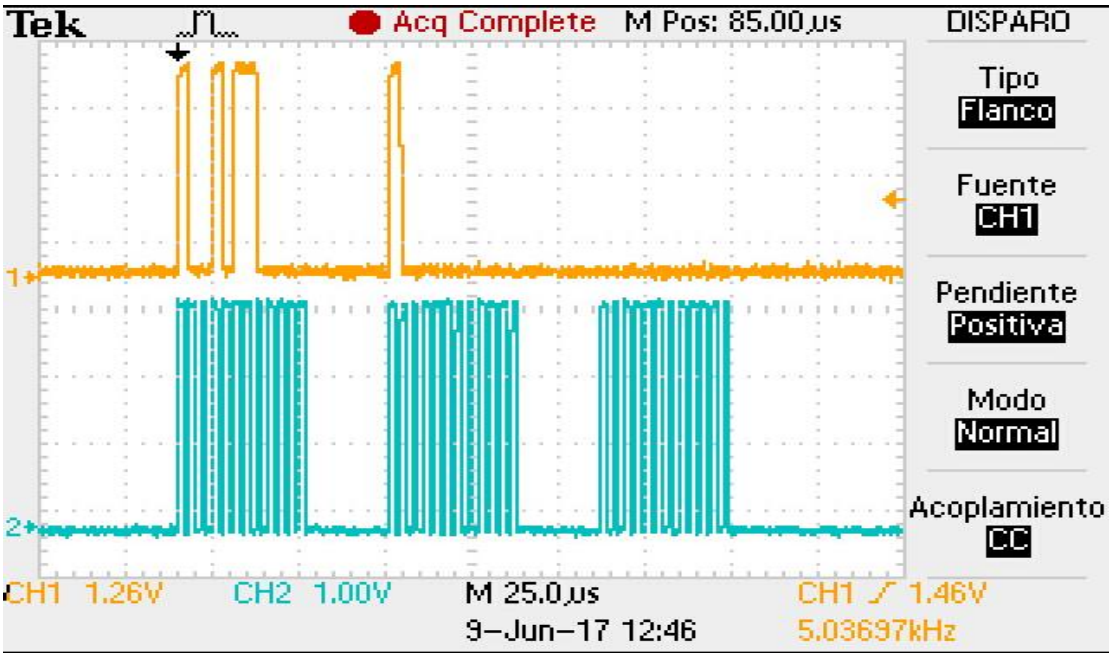


Figura 6-4. Salida del sensor para una exposición al color rojo.

Podemos apreciar en la figura 6-4 cómo también hay un pequeño valor en el espectro del verde. Esto es debido al ancho espectral que presenta el diodo LED.

En la figura 6-5 hemos hecho un zoom de la anterior figura. Cada pulso de la señal ***'CK'*** recoge el nivel de potencia del color representado por 12 bits, 1 bit por cada pulso, empezando por la izquierda con el bit menos significativo (LSB).

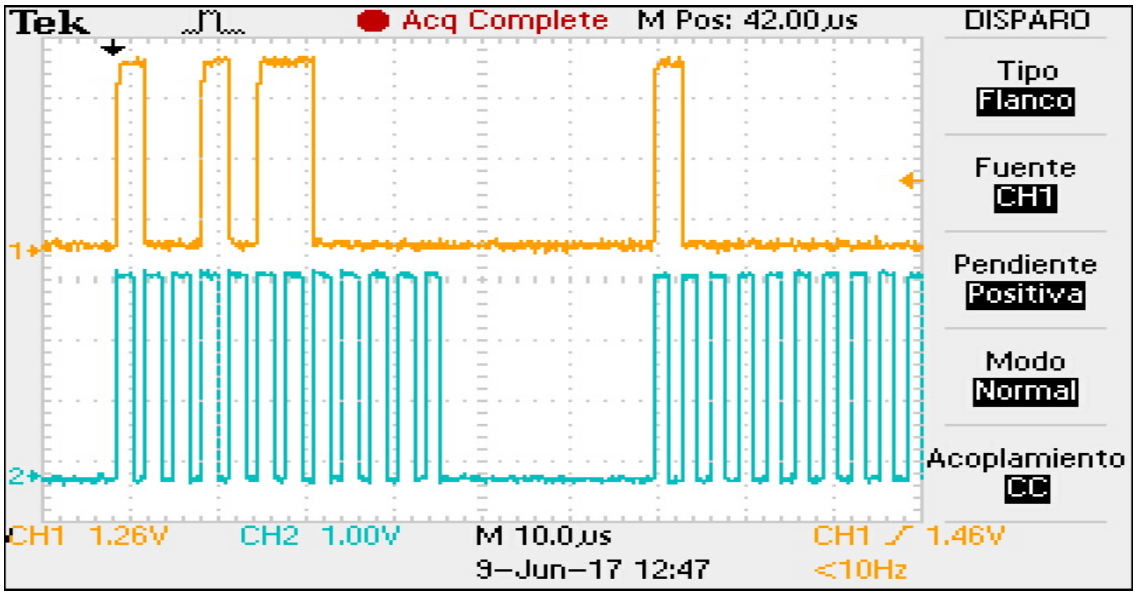


Figura 6-5. Señales de salida rojo y verde

Así a simple vista podemos ver que la salida para el color rojo está determinada por los siguientes bits. "1 0 0 1 0 1 1 0 0 0 0 0", que representa al número 75 en decimal. Para el verde la secuencia sería la siguiente, "1 0 0 0 0 0 0 0 0 0 0 0" que representa al 1 en decimal. Esto nos indica que el nivel de potencia relativa del rojo es 75 veces superior a la del verde.

En la siguiente figura se puede apreciar aún mejor la salida del color rojo.

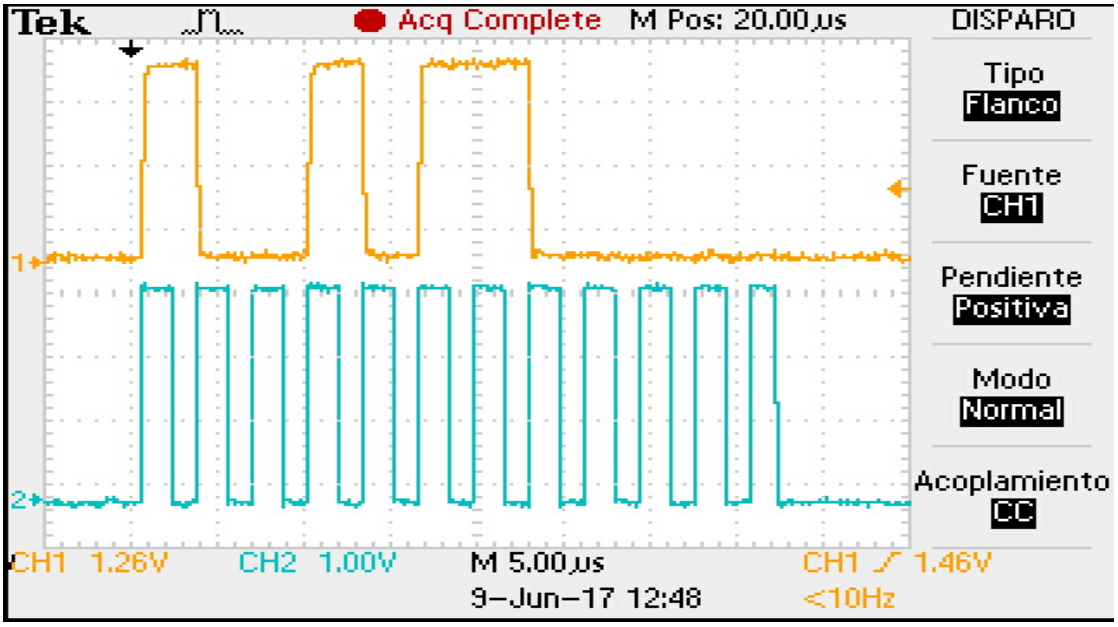


Figura 6-6. Salida del sensor para el color rojo

En las siguientes figuras 6-7 y 6-8 podemos ver las distintas salidas del sensor como respuesta a los colores verde y azul.

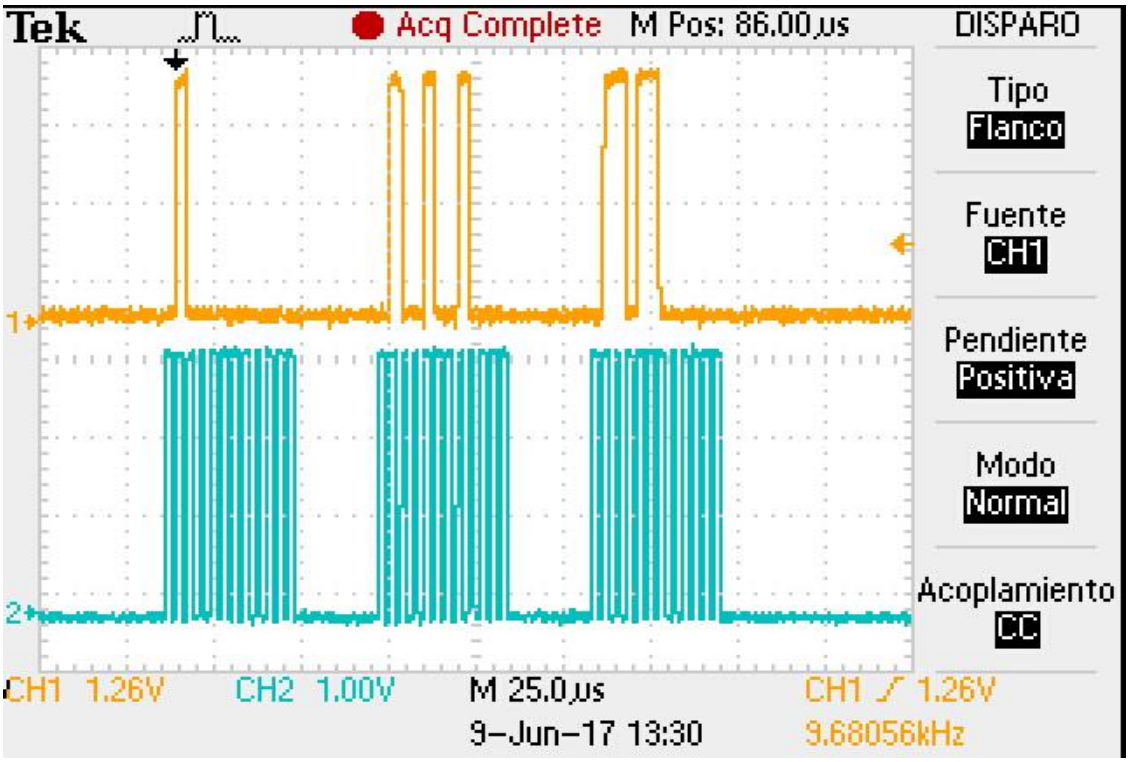


Figura 6-7. Salida del sensor para el verde.

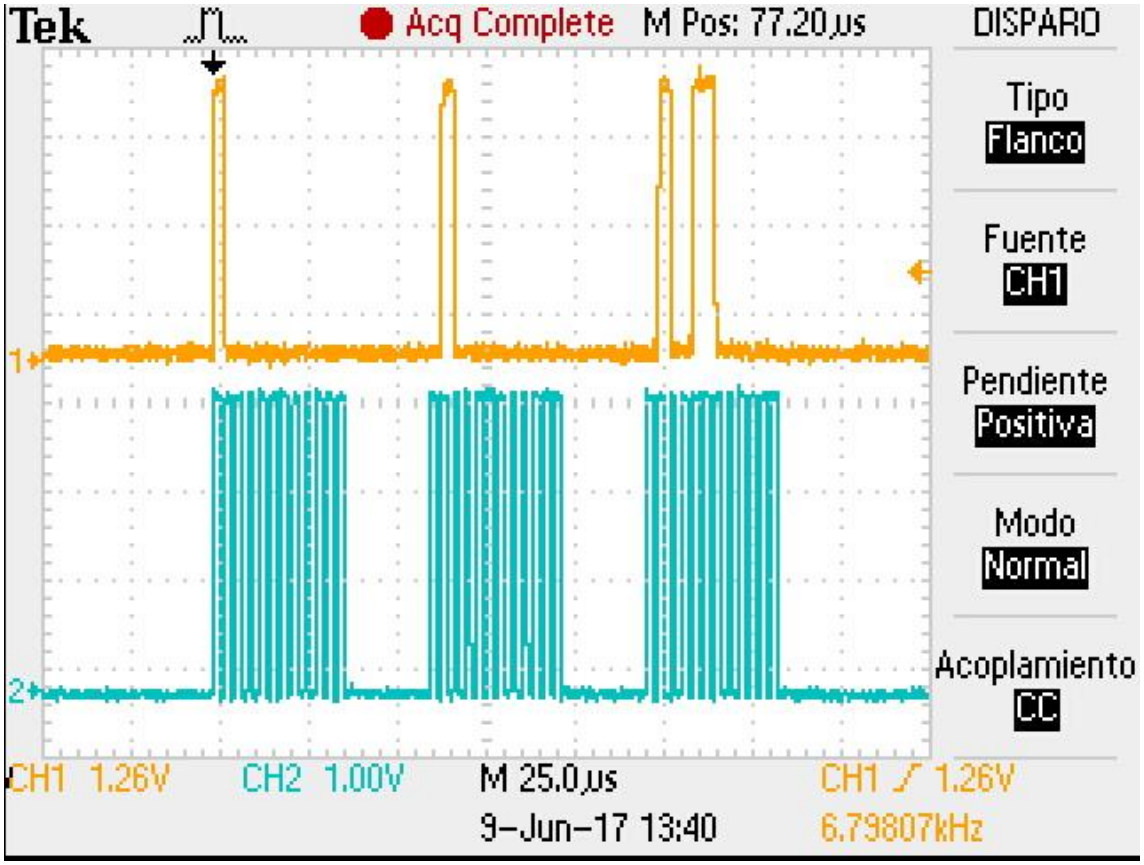


Figura 6-8 Salida del sensor para el azul

6.3 Señales de entrada-salida

En las próximas figuras veremos cómo responde el receptor a diferentes entradas generadas por el transmisor. El canal 1 y 3 del osciloscopio (trazo amarillo y verde respectivamente) corresponden con las entradas al receptor y en el canal 2 y 4 respectivamente las salidas a dichos canales.

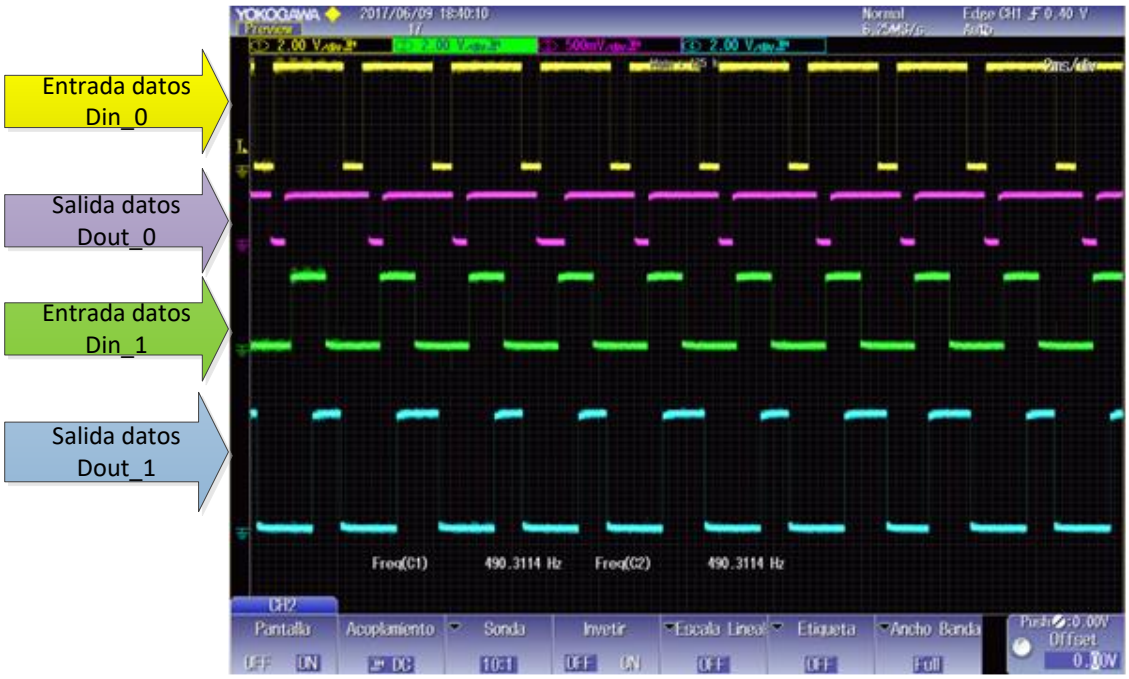


Figura 6-9. Entradas y salidas del receptor (1)

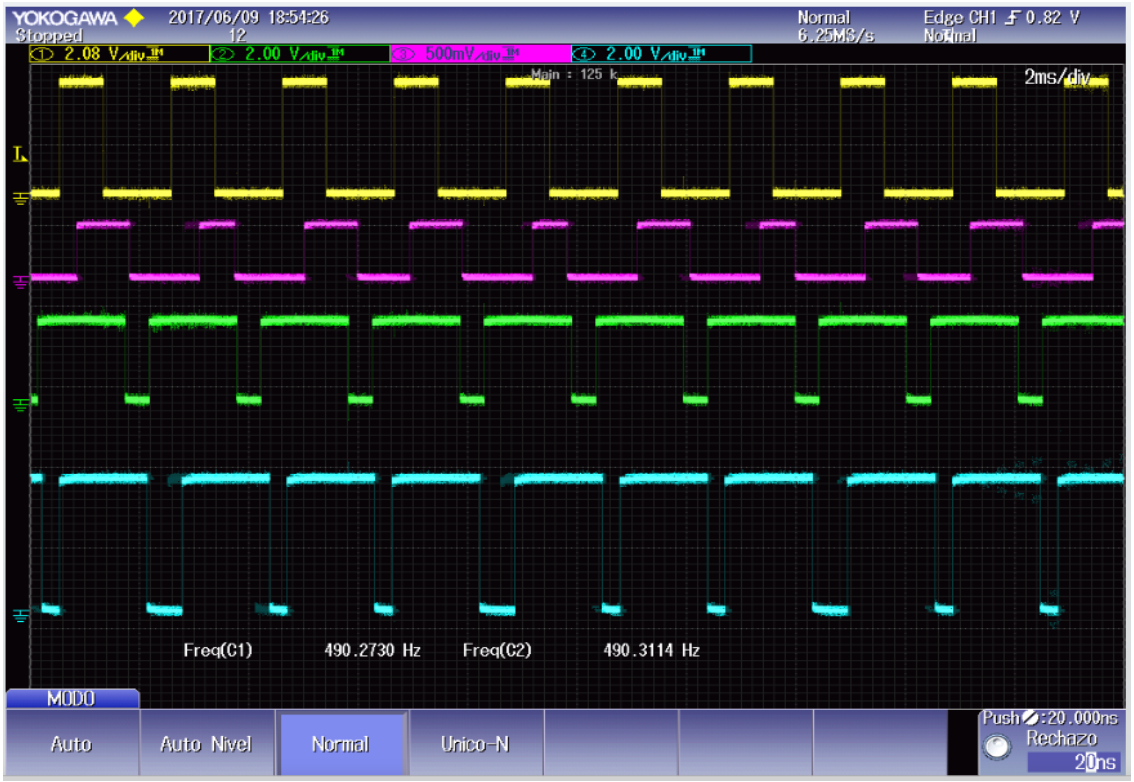


Figura 6-10. Entradas y salidas del receptor (2)

Figura 6-11. Entradas y salidas del receptor (3)

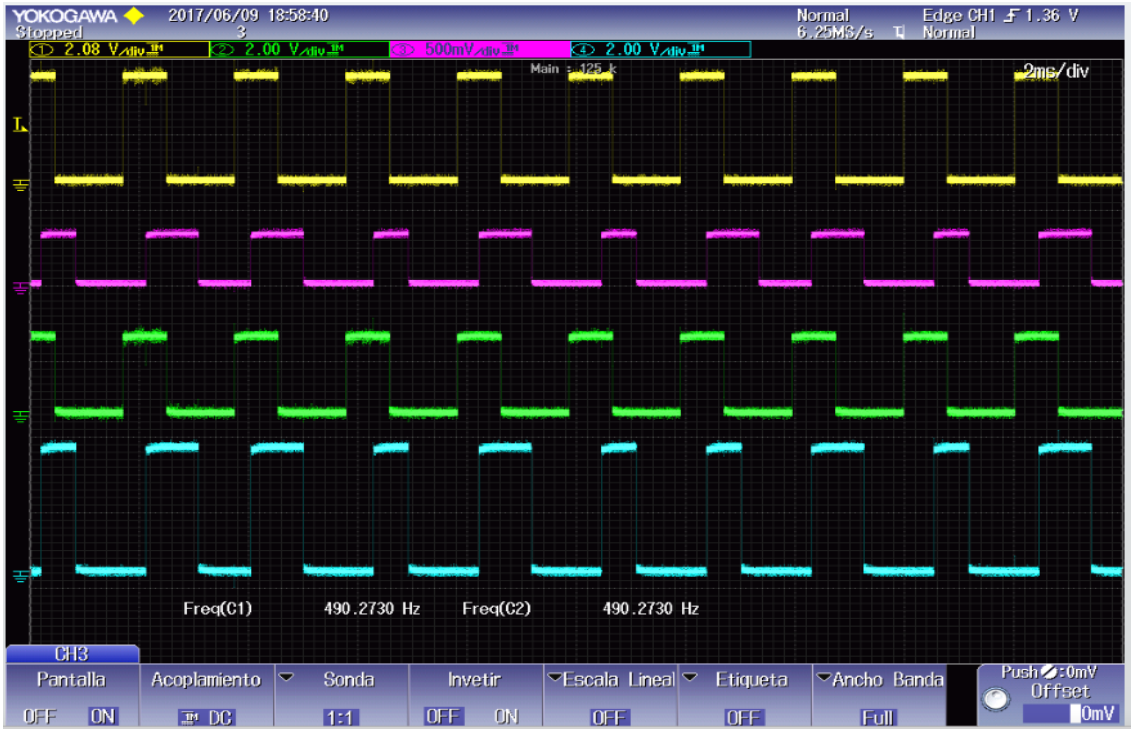


Figura 6-12. Entradas y salidas del receptor (4)

Como podemos apreciar hay cuatro tipos diferentes de figuras que se diferencian en lo siguiente:

1. Con pulsos anchos en el canal 1 y 2 del osciloscopio y pulsos de anchura media en el canal 3 y 4 del osciloscopio. Esto lo podemos apreciar en la figura 6-9.
2. Con pulso de anchura media en el canal 1 y 2 del osciloscopio y pulsos anchos en los canales 3 y 4. Figura 6-10.
3. Pulsos anchos en todos los canales. Figura 6-11.
4. Pulsos de anchura media en todos los canales. Figura 6-12.

Explicación:

Los pulsos anchos corresponden con niveles de potencia elevada y los pulsos medios, con potencias medias. Recordemos que en el transmisor las diferentes tensiones de salida (potencias) se resuelven por medio de una modulación por ancho de pulso, así un ciclo de trabajo próximo al 90% (pulsos anchos) representa casi el máximo voltaje de salida, mientras que un ciclo de trabajo del 50% (pulso medio) representa aproximadamente la mitad del voltaje de salida.

Recordando la tabla de la figura 3-2 del capítulo 3, la cual reproducimos de nuevo aquí en la figura 6-13, podemos ver que el dato “11” corresponde con una potencia elevada del color rojo y verde (R=’200 y G=’200’, en una escala de ‘0’ a ‘255’), el “01” con una potencia elevada de rojo y media de verde, el “10” con una potencia media de rojo y elevada de verde, y el “00” con una potencia media de ambos.

DATO A TRANSMITIR	POTENCIA RELATIVA R	POTENCIA RELATIVA G	POTENCIA RELATIVA B
“11”	‘200’	‘200’	‘50’
“01”	‘200’	‘100’	‘50’
“10”	‘100’	‘200’	‘50’
“00”	‘100’	‘100’	‘50’

Figura 6-13. Relaciones entre datos y potencias para cada color

Con lo que concluimos que el oscilograma de la figura 6-9 corresponde con una salida de datos de “01”, el de la figura 6-10 con un “10”, el de la figura 6-11 con un “11” y por último el de la figura 6-12 que corresponde con un “00”. También podemos observar en todas las figuras como las entradas coinciden con las salidas, lo que implica que se recibe lo mismo que se transmite.

El pequeño desfase que se observa entre las señales de entrada-salida son debidas en mayor parte al tiempo de respuesta del sensor y una pequeña parte por el tiempo necesario para procesado de éstas en el receptor. Las pequeñas discrepancias observadas entre las mismas son debidas a que el receptor está recibiendo datos cerca de la velocidad límite lo que produce pequeños errores de recepción.

Como vemos en estas cuatro gráficas están representados todos los valores de la constelación usada en el transmisor que podemos ver en la figura 3-1 del capítulo 3 sección 3.2 (transmisor) y que por conveniencia repetimos aquí de nuevo en la figura 6-14

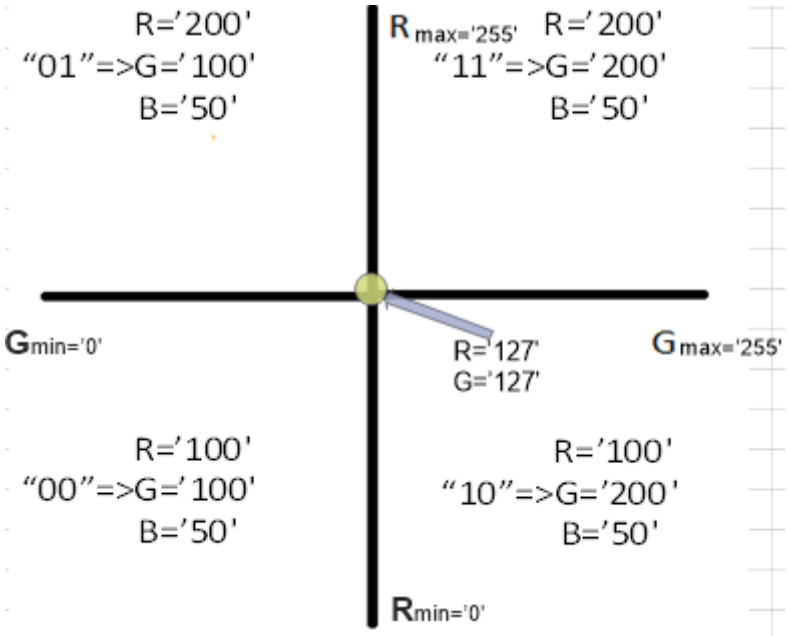


Figura 6-14. Constelación de los datos a transmitir

En la figura 6-15 se muestra la señal de sinronismo (en trazo amarillo) junto a una de las salidas del receptor (trazo azul) para la máxima frecuencia de

respuesta del receptor. Podemos ver en este oscilograma el tiempo de barrido horizontal corresponde con 1 ms/div. Como podemos apreciar entre los pulsos de sincronismo hay un tiempo aproximado de 5 ms. Lo que corresponde a una frecuencia de unos 200 Hz. A partir de esta velocidad la salida no es una reproducción fiel de la entrada, con lo que concluimos que esta es la frecuencia máxima de transmisión.

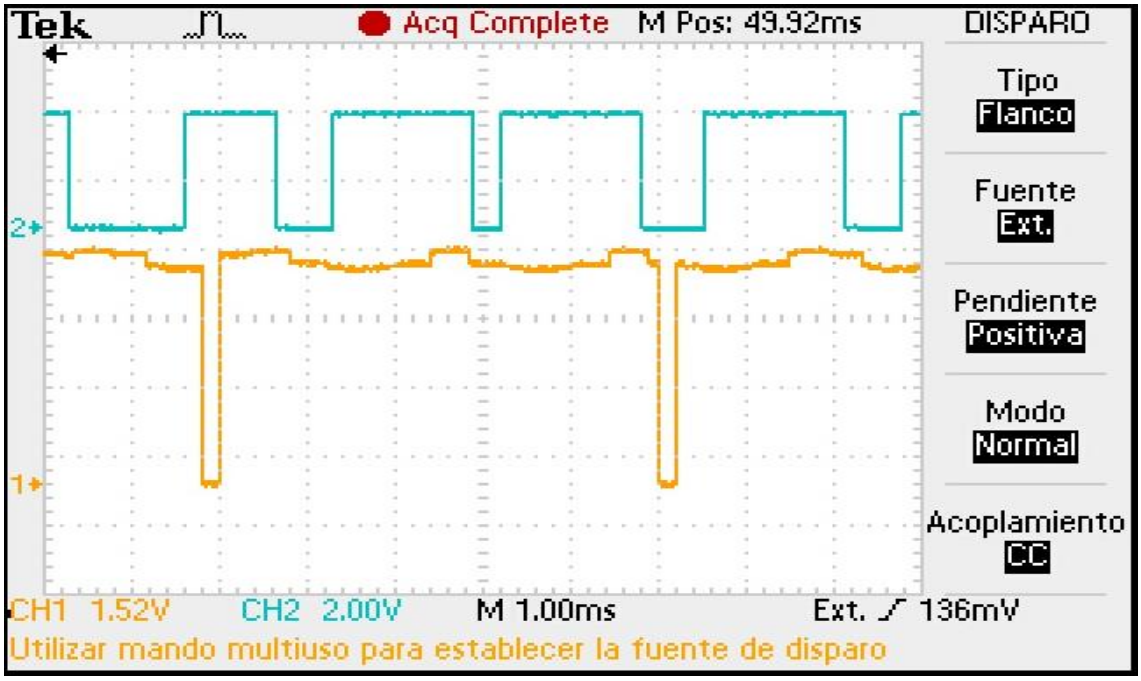


Figura 6-15. Señales de sincronismo y salida

Con la figura anterior se ha calculado la máxima frecuencia de funcionamiento teniendo en cuenta la señal de sincronismo, midiendo el tiempo entre dos pulsos de sincronismo. Si en vez de eso, tenemos en cuenta que los niveles de color están modulados por ancho de pulso (PAM), a los cuales el receptor está respondiendo y cogemos el mínimo ancho de esos pulsos el cual podemos ver en la figura 6-16 (trazo azul) junto a la señal de sincronismo (trazo amarillo) y calculamos su anchura, tendremos el tiempo mínimo que es capaz de seguir el receptor. Teniendo en cuenta que hay que recibir los valores de los tres colores (rojo, verde y azul), podremos calcular una velocidad de respuesta diferente a la anterior, multiplicando este valor mínimo por 3.

$$Tmin = 3\ x\ ancho\ mínimo\ del\ pulso = 3\ x\ 0.32\ ms = 0.96\ ms$$

$$Fmax = \frac{1}{Tmin} \approx 1{,}04\ KHz$$

Vemos que la frecuencia máxima así calculada supera a la anterior en unas 5 veces.

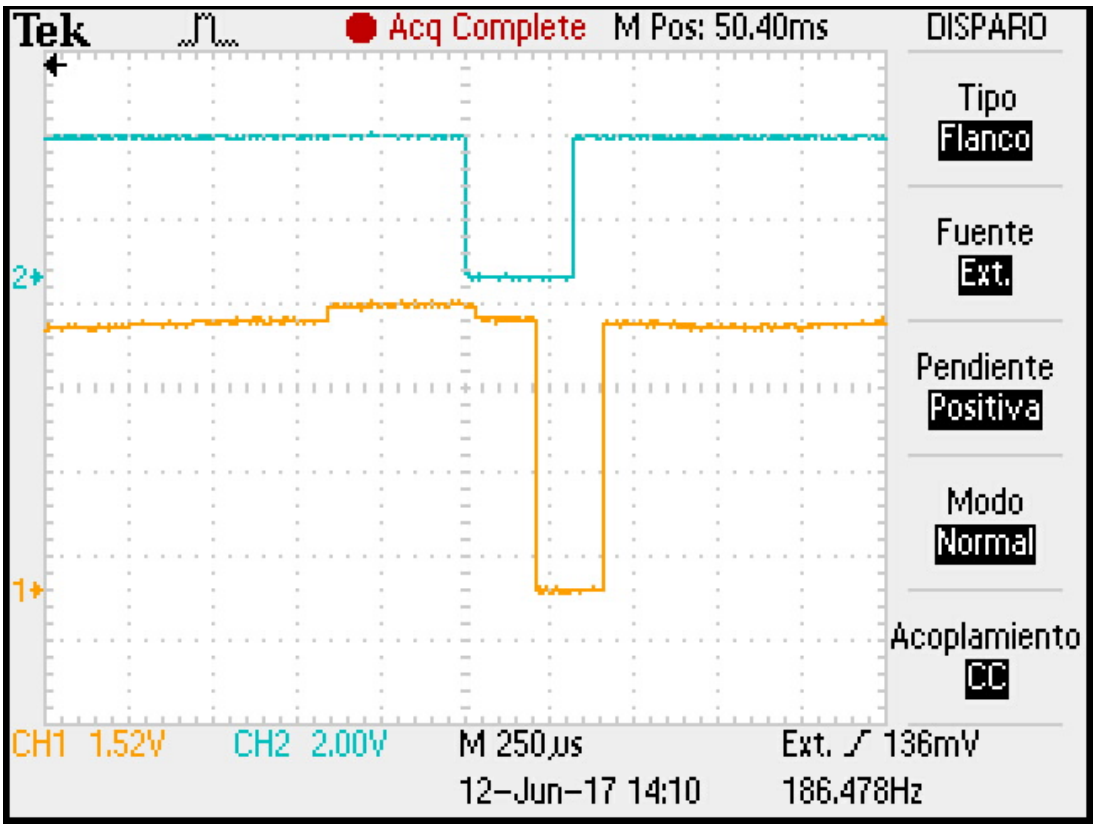


Figura 6-16. Señal sincronismo y salida de pulso muy estrecho

En las siguientes figuras representamos la constelación para diferentes tiempos de transmisión. Cuando nos referimos a 'tiempos de transmisión', nos referimos al tiempo entre impulsos de sincronismo del transmisor. Los tiempos de transmisión varían en estas figuras entre 5 ms y 100 ms.

En la figura 6-17 podemos ver la constelación para un tiempo de transmisión de 100 ms y 5 segundos de persistencia en pantalla, teniendo en cuenta que entre 2 pulsos de sincronismo se transmiten 20 datos, esto se traduce en un envío de

20 datos cada 100 ms ó 200 datos por segundo. La velocidad de trasmisión en este caso sería:

$$Velocidad\left(\frac{Datos}{seg}\right) = \frac{20Datos}{100ms}\ x\frac{1ms}{0{,}001seg} = 200\frac{datos}{seg}\ \text{ó}\ 200\ baudios$$

Como el tiempo de persistencia es de 5 segundos, el número de datos representados son:

$$Número\ de\ datos = 5\ seg\ x\ 200\frac{datos}{seg} = 1.000\ datos$$

Por lo que la figura 6-7 representa unos 1.000 datos. Como podemos ver se mantiene la constelación bien definida, lo que quiere decir que la tasa de error de bit (BER), se mantiene con unos valores aceptables para esta velocidad de transmisión.

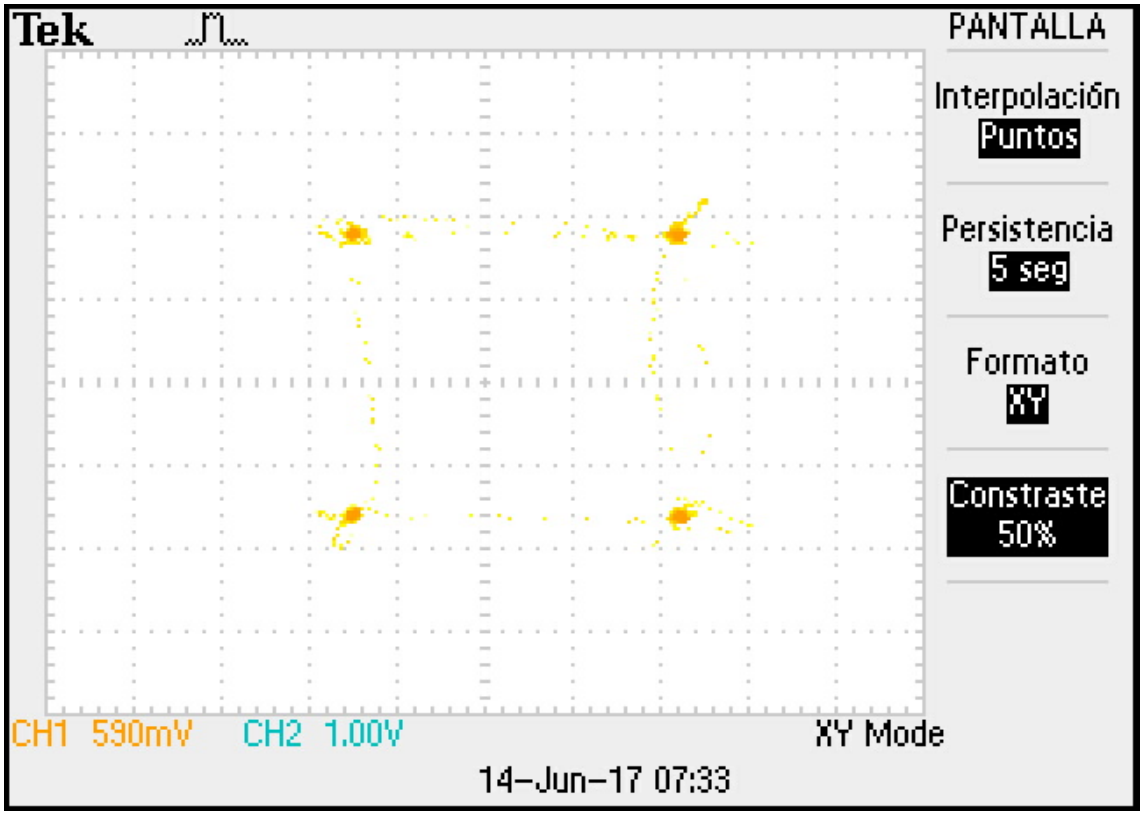


Figura 6-17. Constelación para un tiempo de persistencia 5 segundos y una velocidad transmisión de 100 ms.

En la figura 6-18 podemos ver la constelación para un tiempo de transmisión de 10 ms y 5 segundos de persistencia en pantalla, lo que se traduce en una velocidad de transmisión de:

$$Velocidad\left(\frac{Datos}{seg}\right) = \frac{20Datos}{10ms}\ x\ \frac{1ms}{0{,}001seg} = 2.000\frac{datos}{seg}\ \text{ó}\ 2.000\ baudios$$

Como antes la persistencia en pantalla es de 5 segundos, por lo que el número de datos transmitidos hasta ese instante es de:

$$Número\ de\ datos = 5\ seg\ x\ 2.000\frac{datos}{seg} = 10.000\ datos$$

En este caso la figura 6-18 representa a 10.000 datos. Como podemos ver se mantiene la constelación bien definida, lo que quiere decir que a esta velocidad todavía la BER se mantiene con unos valores aceptables.

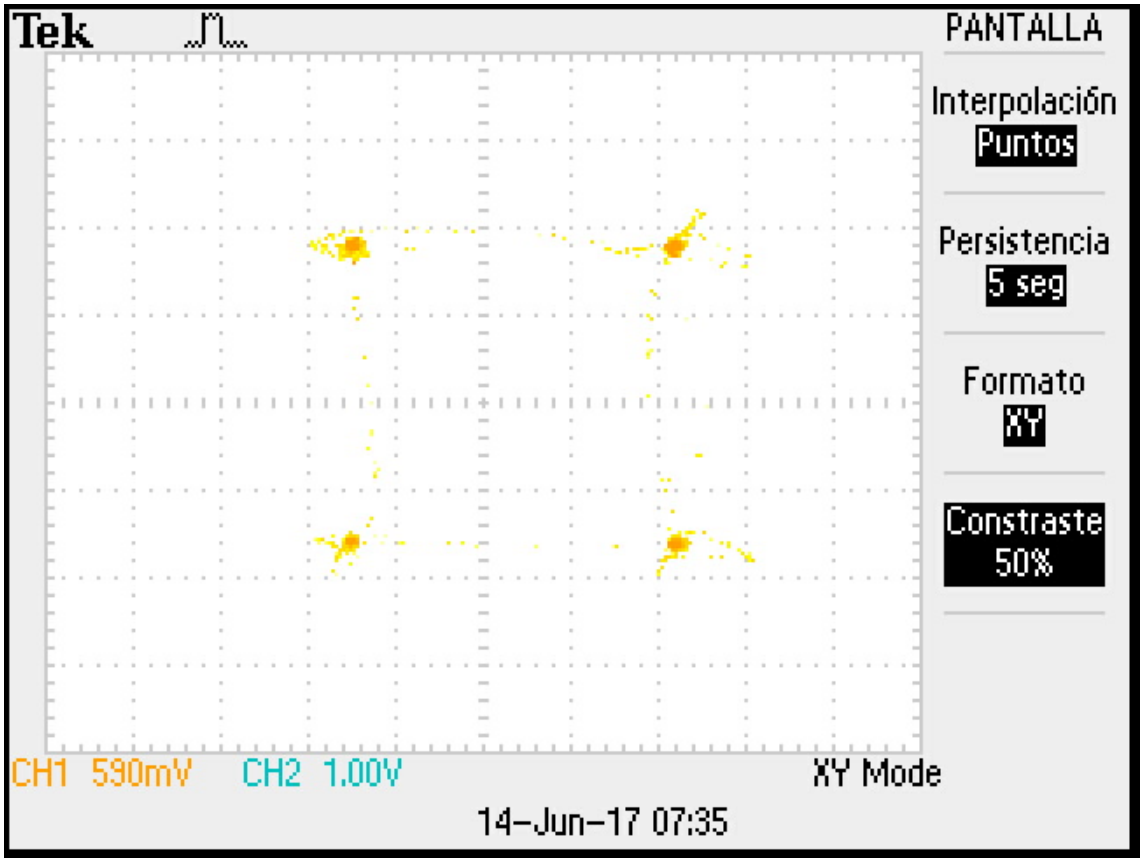


Figura 6-18. Constelación para un tiempo de persistencia 5 segundos y un tiempo de transmisión de 10 ms

En la figura 6-19 podemos ver la constelación para un tiempo de transmisión de 5 ms y 5 segundos de persistencia en pantalla, lo que se traduce en una velocidad de transmisión de:

$$Velocidad\left(\frac{Datos}{seg}\right) = \frac{20Datos}{5ms}\ x\ \frac{1ms}{0{,}001seg} = 4.000\frac{datos}{seg}\ \text{ó}\ 4.000\ baudios$$

Para este caso la cantidad de datos representados en la figura 6-19 es de:

$$Número\ de\ datos = 5\ seg\ x\ 4.000\frac{datos}{seg} = 20.000\ datos$$

La constelación sigue más o menos bien definida por lo que llegamos a la misma conclusión que en los párrafos anteriores.

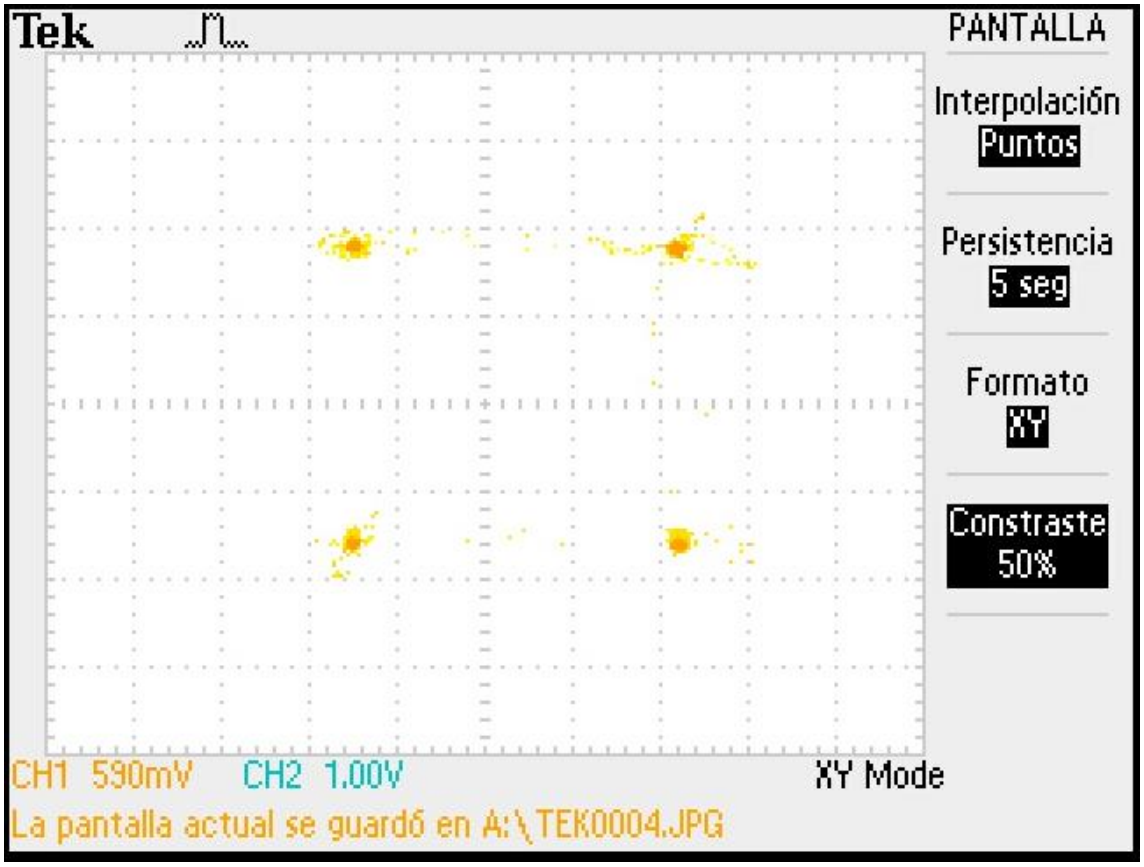


Figura 6-19. Constelación para un tiempo de persistencia 5 segundos y un tiempo de transmisión de 5 ms

En la figura 6-20 podemos ver la constelación para un tiempo de transmisión de 4 ms y 5 segundos de persistencia en pantalla, lo que se traduce en una velocidad de 5.000 datos por segundo por parte del transmisor ó 5.000 baudios, que viene a ser 25.000 datos transmitidos, según se ha comentado anteriormente. Como podemos ver a partir de esta velocidad la constelación no está muy bien definida, lo que quiere decir que a esta velocidad la BER ha aumentado considerablemente, lo cual es congruente con los resultados obtenidos con representación gráfica de las señales de entrada-salida comentadas anteriormente.

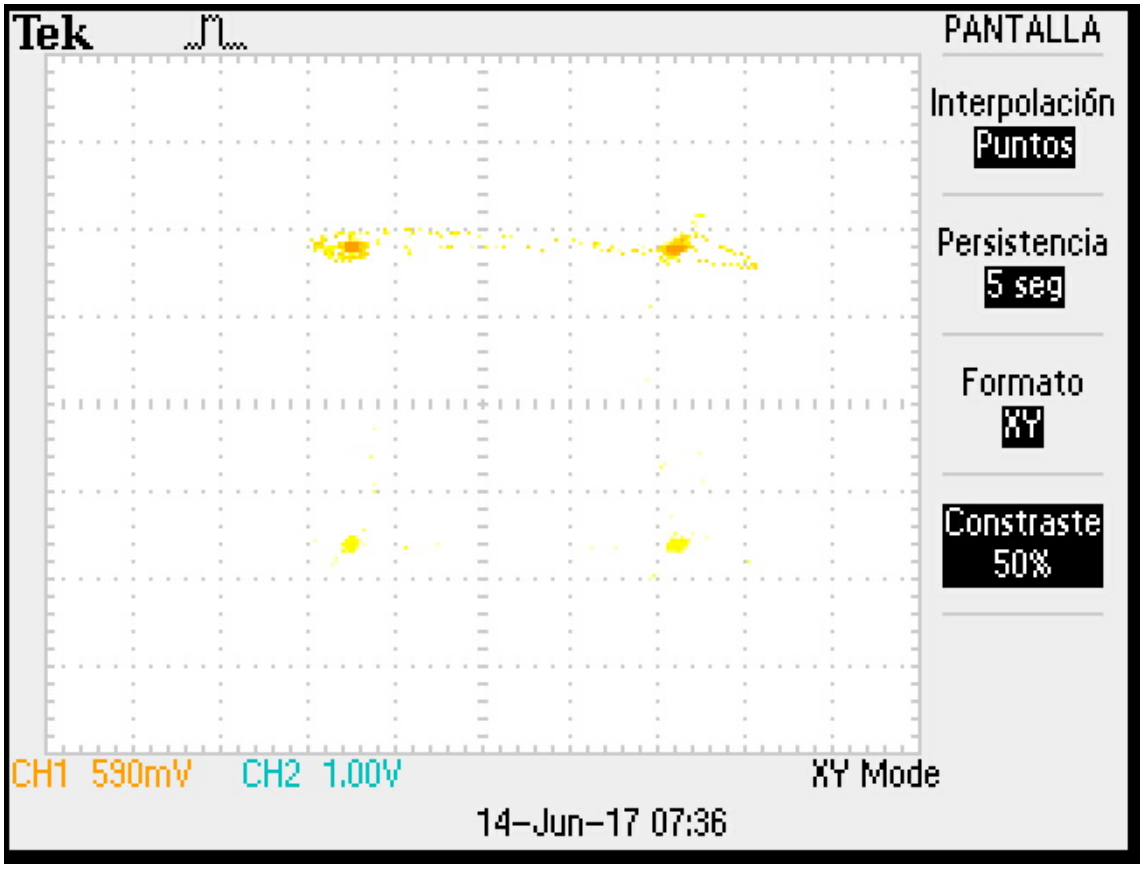


Figura 6-20. Constelación para un tiempo de persistencia 5 segundos y un tiempo de transmisión de 4 ms

En la figura 6-21 podemos ver la constelación para un tiempo de transmisión de 10 ms y 30 segundos de persistencia en pantalla, lo que se traduce en una velocidad de transmisión de 2.000 datos por segundo ó 2.000 baudios tal y como se vio anteriormente. En dicha figura se representan los datos transmitidos que en este caso son 60.000 datos. Como podemos ver a esta velocidad la constelación se mantiene más o menos bien definida, lo que quiere decir que a esta velocidad la BER es aceptable.

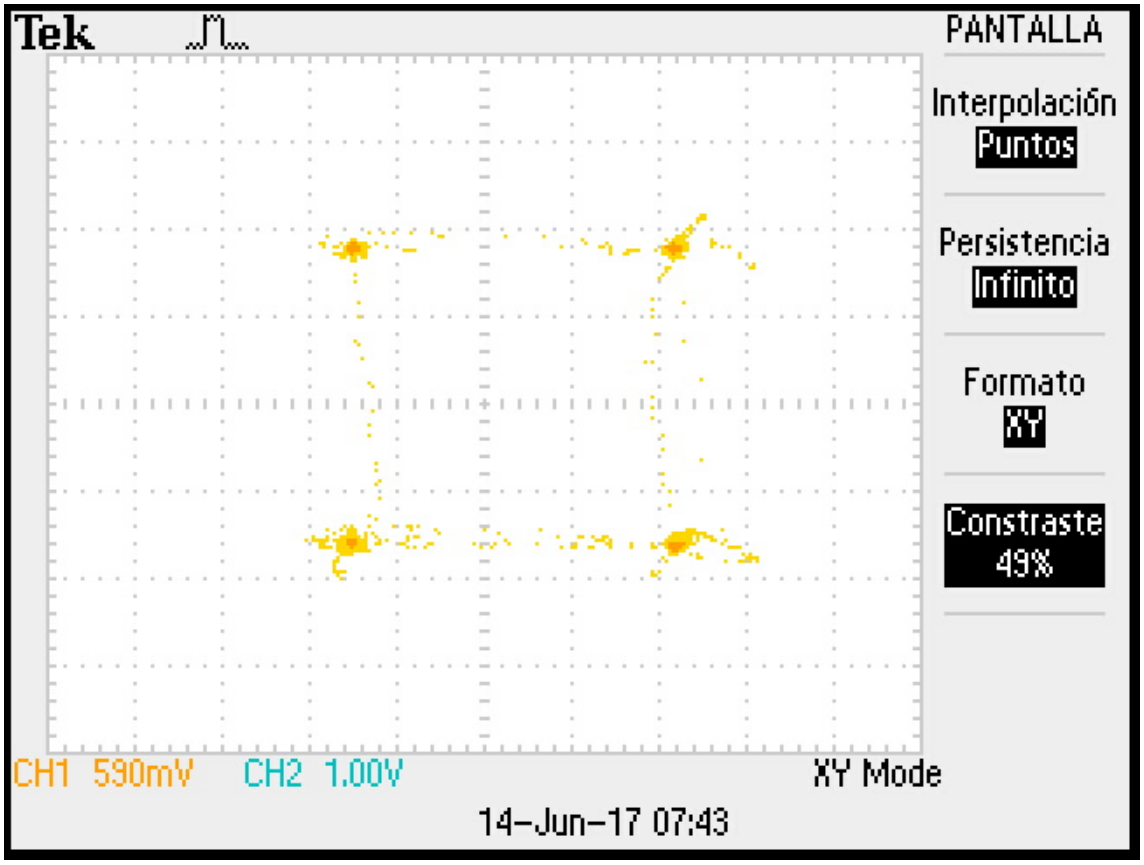


Figura 6-21. Constelación para un tiempo de persistencia unos 30 segundos y un tiempo de transmisión de 10 ms

En la figura 6-22 podemos ver la constelación para un tiempo de transmisión de 5 ms y 30 segundos de persistencia en pantalla, en este caso la velocidad de transmisión es de 4.000 datos por segundo. En dicha figura podemos ver representados 120.000 datos transmitidos. Como podemos ver a esta velocidad la constelación se mantiene más o menos bien definida, lo que quiere decir que a esta velocidad la BER sigue siendo aceptable.

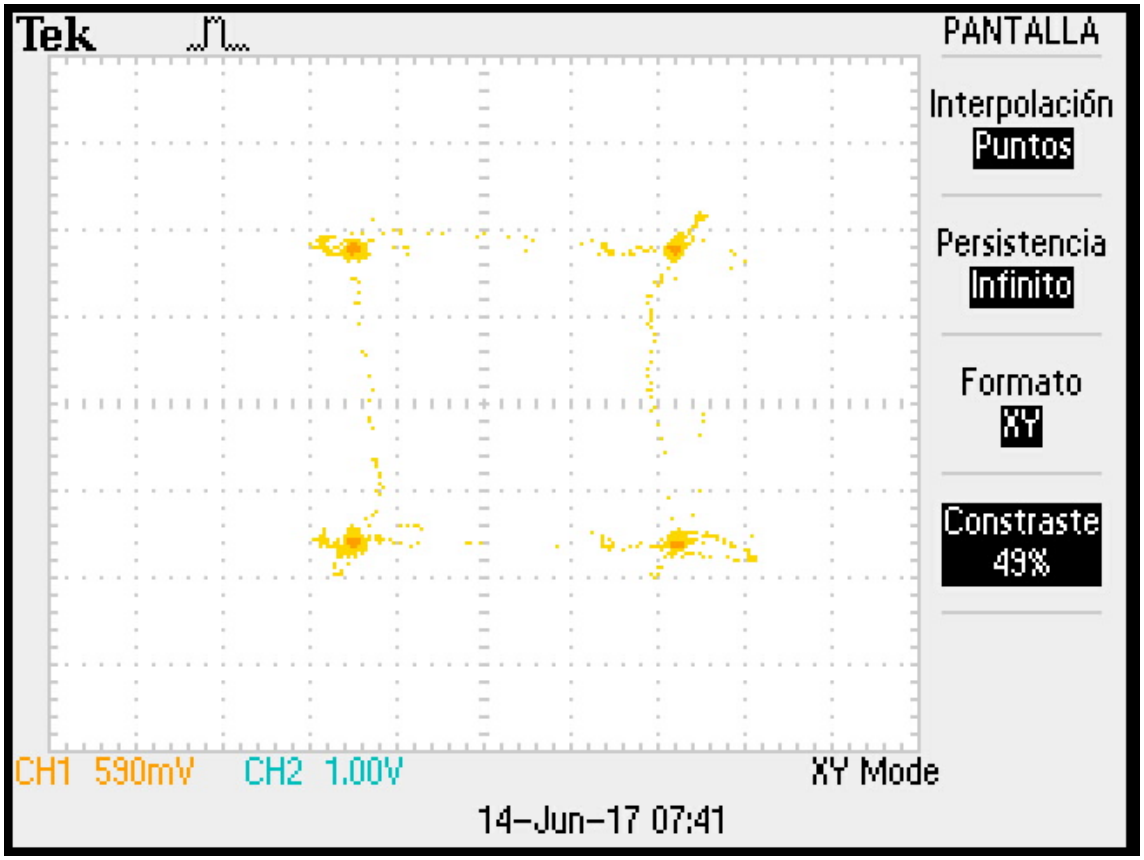


Figura 6-22. Constelación para un tiempo de persistencia 30 segundos y un tiempo de transmisión de 5 ms

En la figura 6-23 podemos ver la constelación para un tiempo de transmisión de 4 ms y 30 segundos de persistencia en pantalla, lo que se traduce en una velocidad de transmisión de 5.000 datos por segundo ó 5.000 baudios. En dicha figura se representan 150.000 datos transmitidos. Como podemos ver a esta velocidad la constelación empieza degradarse, lo que es totalmente compatible con los resultados anteriores.

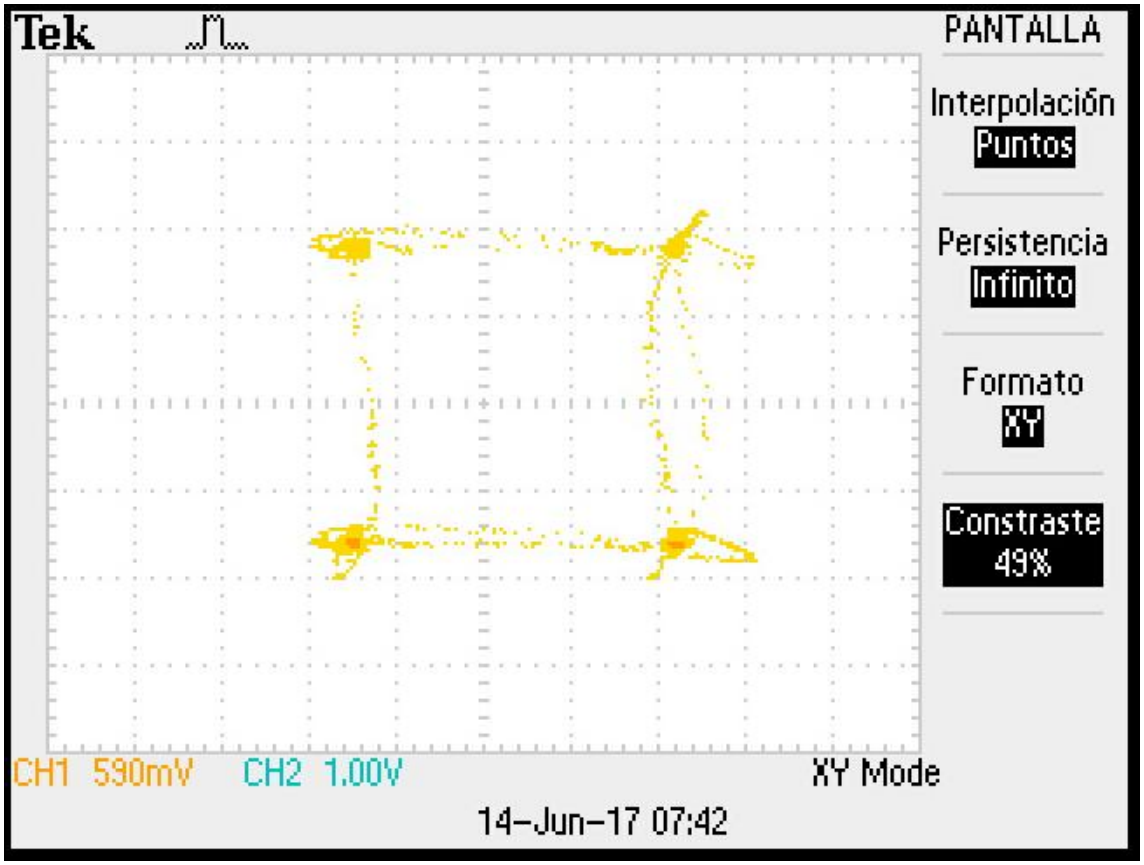


Figura 6-23. Constelación para un tiempo de persistencia 30 segundos y un tiempo de transmisión de 4 ms

No hemos calculado la BER de manera empírica porque no tenemos un registro de los datos transmitidos ni de los recibidos.

De todas las maneras esta no es la intención de este proyecto. En futuros proyectos se puede abordar este tema, cosa que vamos a proponer en el capítulo 6 “conclusiones y líneas futuras de trabajo”.

En las figuras 6-24 y 6-25 podemos ver el montaje para la validación del sistema.

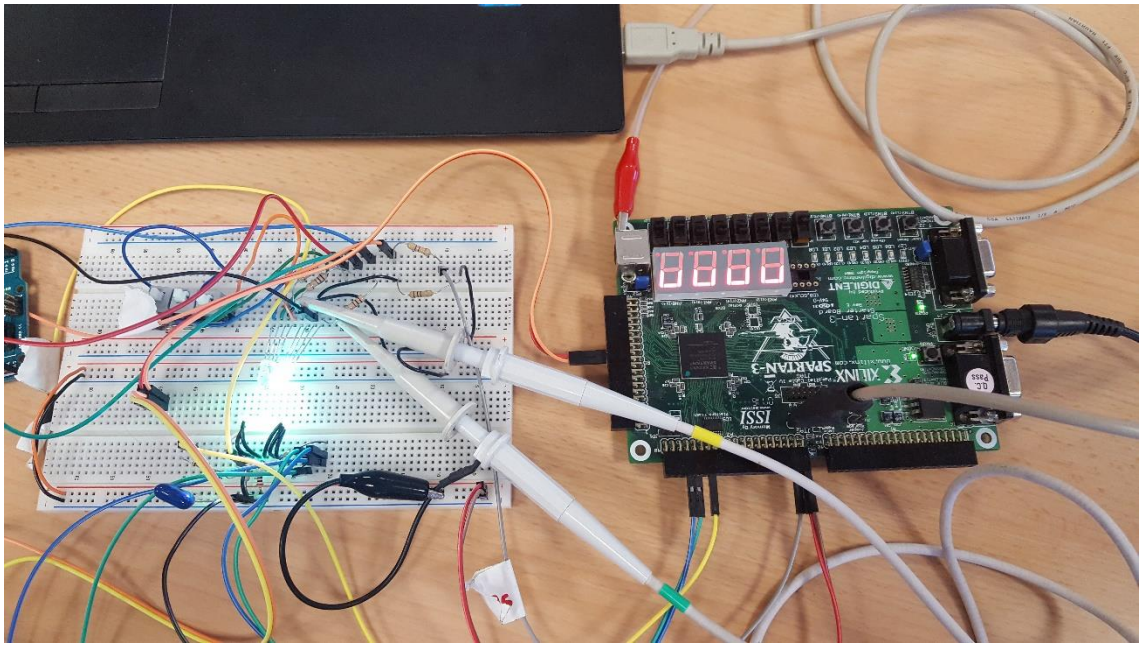


Figura 6-24. Foto de laboratorio 1

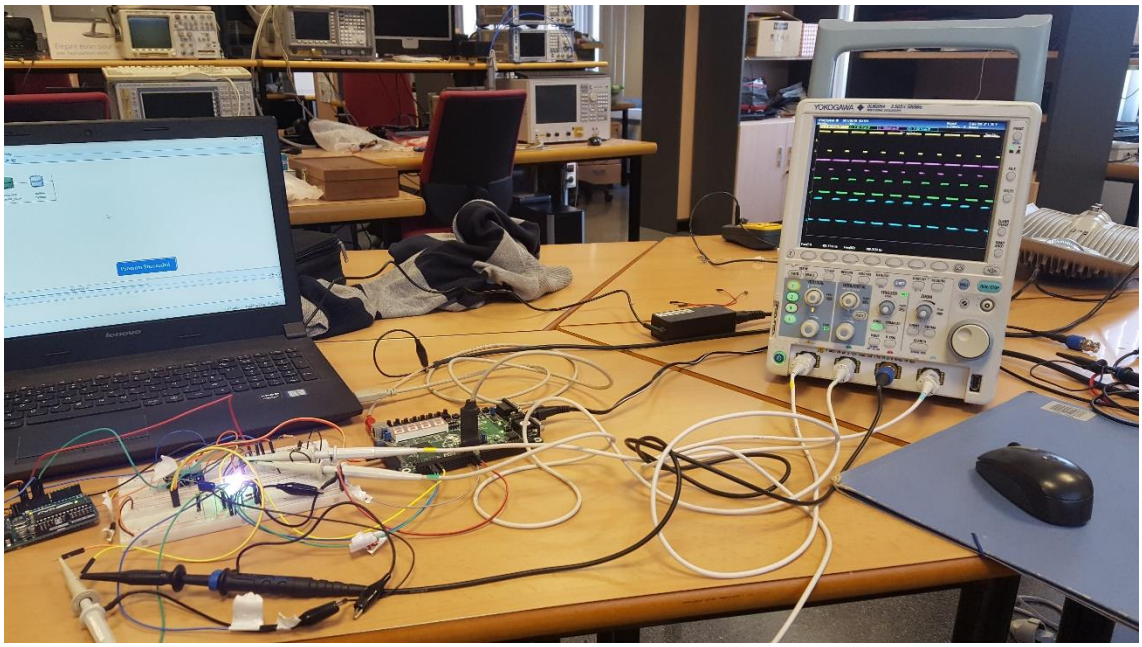


Figura 6-25. Foto de laboratorio 2

Capítulo 7

Conclusiones y líneas futuras de trabajo

Capítulo 7: Conclusiones y líneas futuras de trabajo

7. Conclusiones y líneas futuras de trabajo

7.1 Conclusiones

El objetivo de este proyecto era realizar un receptor que hace uso de un demodulador CSK basado en el valor medio de cada uno de los componentes RGB, tal y como fue propuesto, basado en el transmisor existente al principio del desarrollo del trabajo. A partir del desarrollo de este proyecto se han extraído una serie de conclusiones además de las derivadas del objetivo principal. A continuación se exponen las principales:

- ✓ Se ha conseguido validar el sistema transmisor CSK de forma experimental comprobando la correcta transmisión de datos. En el instante inicial de la realización del trabajo, por parte del grupo proponente, sólo se había validado mediante simulación o con el uso de cámaras CMOS o CCD's.

- ✓ Se ha comprobado que el uso de sensores de color proporciona un incremento en la velocidad del sistema, aumentando la tasa de transmisión al evitar la limitación impuesta por las cámaras existentes en la actualidad en el mercado, al menos de las que habitualmente se encuentran en los dispositivos móviles. Además de ello se ha conseguido un sistema escalable en cuanto a velocidades y a sensibilidad gracias al uso de dispositivos FPGA que permiten mayores prestaciones en cuanto a ancho de banda y cálculo que los microprocesadores empleados, por ejemplo en el transmisor.

- ✓ Se ha comprobado el funcionamiento de una nueva forma de discriminación de los datos mediante comparación con el valor medio de la constelación. De esta manera, los datos son detectados mediante comparación de la señal consigo misma y no mediante comparación con un valor absoluto. Esto hace al sistema más robusto frente las posibles y más que frecuentes variaciones de la iluminación ambiente.

- ✓ Precisamente debido a lo anterior, el sistema no sólo es robusto frente a los cambios de iluminación ambiente, sino a que pueda funcionar de igual manera en diferentes entornos y diferentes condiciones sin necesidad de una fase de entrenamiento como en otros desarrollados anteriormente.

7.2 Líneas Futuras

Durante el desarrollo de este trabajo se han detectado posibles mejoras y algunas carencias que podrían ser fruto de posteriores desarrollos. En primer lugar el transmisor estaba implementado previamente, con lo que la constelación usada era directamente basada en componentes RGB y no en un espacio XY CIE, por ejemplo, como en el citado estándar VLC. La respuesta del receptor demostró que con la misma implementación valdría para ese tipo de emisor también. El desarrollo de otras constelaciones sería objetivo del transmisor, no de este proyecto. Como posibles líneas futuras a partir de este trabajo podrían encontrarse:

- ✓ El uso de lentes o concentradores para dotar al sistema de una mayor ganancia y alcanzar mayores distancias sin necesidad de incrementar la potencia eléctrica. La superficie óptica del sensor de color empleado es muy pequeña, con lo que gran parte de la potencia óptica se desperdicia. El uso de lentes concentradoras ampliaría el área activa del receptor sin apenas incremento del coste del sistema.

- ✓ Mejorar la resolución del sistema empleando todos los bits de color procedentes del sensor. En este sistema no se han empleado los bits menos significativos con la consiguiente pérdida de resolución en la lectura del color. Además esto permitiría incrementar la distancia al necesitar menores niveles de potencia para realizar la discriminación de color.

- ✓ Como consecuencia de lo anterior, se podría, además emplear una constelación de 8-CSK, por ejemplo al disponer de mayor resolución, con lo que aumentaría la tasa de datos al aumentar el número de bits por símbolo.

- ✓ Otra línea futura de trabajo sería implementar un receptor con sensores de respuesta más rápida para obtener unas frecuencias de trabajo mayores.

- ✓ Por otro lado habría también que hacer un estudio de la degradación de la señal recibida frente a las interferencias producidas por elementos externos como pueden ser la luz solar o alumbrado artificial (luces fluorescentes, lámparas de descarga, etc) y buscar algún tipo de solución para hacer a estos sistemas más robustos frente a este tipo de complicaciones.

Bibliografía

Bibliografía

[1] IEEE Standard for Local and Metropolitan Area Networks------Part 15.7: Short---Range Wireless Optical Communication Using Visible Light.

[2] H. Elgala and T.D.C. Little, “Reverse polarity optical-OFDM (RP-OFDM): dimming compatible OFDM for gigabit VLC links,” Optics Express, Vol. 21, Issue 20, pp. 24288-24299 (2013)

[3] http://purelifi.com/li-fi-mobile-world-congress-2014/

[4] M. Kavehrad, P. Amirshahi, “Hybrid MV-LV Power Lines and White Light Emitting Diodes for Triple-Play Broadband Access Communications,” IEC Comprehensive Report on Achieving the Triple Play: Technologies and Business Models for Success, ISBN 1-931695-51-2, pp. 167-178, January 2006.

[5] P. Haigh, F. Bausi, Z. Ghassemlooy, I. Papakonstantinou, H. Le Minh, C. Fléchon, and F. Cacialli, "Visible light communications: real time 10 Mb/s link with a low bandwidth polymer light-emitting diode," Opt. Express 22, 2830-2838 (2014)

[6] "Philips. Hue: Personal Wireless Lighting. (2013). [Online]. Available:", [online] Available: https://www.meethue.com/

[7] K.-I. Ahn and J. K. Kwon "Color intensity modulation for multicolored visible light communications", *IEEE Photon. Technol. Lett.*, vol. 24, no. 24, pp.2254 - 2257 2012
Abstract | Full Text: PDF (574KB) | Full Text: HTML

[8] "IEEE Standard for Local and Metropolitan Area Networksâ”Part 15.7: Short-Range Wireless Optical Communication Using Visible Light, IEEE Standard 802.15.7, 2011.",

[9] **Prat Viñas, Lluís.(2006)** Dispositivos electrónicos y fotónicos: fundamentos

[10] P. Butala , J. Chau and T. D. C. Little "Metameric modulation for diffuse visible light communications with constant ambient lighting", *Proc. Int. Workshop Opt. Wireless Commun.*, pp.1 -3

[11] Grupo de Nuevas Actividades Profecionales (2004) La situación de las tecnologías WLAN basadas en el estándar IEEE 802.11 y sus variantes (“Wi-Fi”)

[12] K. Choi, Y. Jang, J. Noh, M. Ju and Y. Park "Visible light communications with color and dimming control by employing VPPM coding", *Proc. 4th Int. Conf. Ubiquitous Future Netw.*, pp.10 -12
Abstract | Full Text: PDF (1037KB) | Full Text: HTML

[13] Serafín A. Pérez, Enrique Soto, Santiago Fernández. 2002. Diseño de sistemas digitales con VHDL

[14] Delores M. Etter,1997. Solución de problemas de ingeniería con MATLAB

[15] Yokoi, A., et. al., “More description about CSK constellation,” March 2011, IEEE 802.15 contribution
15-11-0247-00-0007, https://mentor.ieee.org/802.15/dcn/11/15-11-0247-00-0007-csk-constellation-in-allcolor-band-combinations.pdf.

Pliego de condiciones

Pliego de condiciones generales:

Para la realización de este proyecto se ha necesitado de material hardware, software y varias sesiones de laboratorio. A continuación describimos cada uno de ellos.

Pliegos de especificaciones técnicas:

1.Especificaciones de materiales y equipos

Material Hardware

- 1 placa de prototipado de digilent con FPGA de la casa Xilinx modelo xs3s1000-4ft256 donde se implementa el receptor VLC.
- 1 protoboard para el montaje del sensor y LED RGB
- 1 Cable de programación Digilent JTAG-USB que se usa para la programación de la FPGA.
- 1 Fuente de alimentación 5 voltios, 2.5 amperios. Para la alimentación de la placa de prototipado.
- 1 módulo arduino uno con el cual se generan las señales del transmisor.
- 1 cable USB-USB para la conexión del módulo arduino uno al ordenador.
- 1 osciloscopio Tektronix modelos TDS 2012 con el que se llevan a cabo la mayoría de las pruebas prácticas del diseño.
- Sondas para osciloscopio y cables de conexión para poder efectuar el cableado de los circuitos y las pruebas pertinentes.
- Un generador de funciones PROMAX modelo GF- 232 para pruebas de respuesta del sensor a diferentes estimulaciones.
- Un ordenador portátil Lenovo modelo B51-80 con el cual se llevan a cabo las simulaciones y programación de la FPGA.
- 1 sensor RGB de la casa Hamamatsu modelo S9706 como transductor luminoso-eléctrico para la recepción de señales luminosas tipo RGB.
- 1 diodo LED RGB que genera los diferentes colores RGB a transmitir.
- Resistencias, condensadores, interruptores, cables de conexión como elementos necesarios para conexionar las diferentes partes de los

componentes así como para diferentes pruebas relacionadas con el sensor.

Material software

- ModelSim de la casa Mentor Graphics como simulador digital de señales
- ISE Project Navigator Versión 14.6 herramienta software para la programación de la FPGA entre otras cosas.
- Microsoft Windows 10 como sistema operativo.

2. Especificaciones de ejecución

El proceso de ejecución ha sido el siguiente:

1. Recopilación de información necesaria (bibliografía, artículos…).
2. Diseño del proyecto donde se especifican los diferentes pasos para la ejecución del mismo.
3. Simulación de cada una de las partes y global para comprobar el correcto funcionamiento a nivel software.
4. Implementación de cada una de las partes que integran el diseño.
5. Obtención de los resultados prácticos a partir de medidas hechas en laboratorio con señales reales.
6. Redacción del proyecto.

Presupuesto

PRESUPUESTO

En el pliego de condiciones se ha hecho una descripción detallada de todos los recursos necesarios para la ejecución de este proyecto. En base a estos recursos elaboraremos el presupuesto del proyecto según las pautas que el Colegio Oficial de Ingenieros de Telecomunicaciones (COIT) establece.

El COIT ha estado publicando hasta el año 2008, una lista de honorarios orientativos que se denominaban Costes Estimados de Trabajos Profesionales. Por modificación de la Ley de Colegios Profesionales, mediante la Ley 25/2009 del 22 de diciembre, no es posible seguir publicando estas listas. El Colegio no puede elaborar baremos de honorarios, ni siquiera orientativos, salvo que sean con la finalidad de tasar costes en los procedimientos judiciales.

Sin embargo, el Colegio Oficial de Ingenieros de Telecomunicación (COIT) dispone de una herramienta de cálculo para que el colegiado pueda valorar por sí mismo los trabajos profesionales que realiza. Esta herramienta se encuentra en la web del COIT en Ejercicio Profesional / Apoyo y Desarrollo Técnico /Información General.

Siguiendo las recomendaciones del Colegio Oficial de Ingenieros de Telecomunicación (COIT) y haciendo uso de la herramienta orientativa para trabajos profesionales 2011, el presupuesto se ha desglosado en varias secciones que se muestran a continuación:

P1. Recursos materiales
P2. Trabajo tarifado por tiempo empleado
P3. Costes de redacción del Trabajo Final de Carrera
P4. Material fungible
P5. Derechos de visado del COITT
P6. Aplicación de impuestos

P1. Recursos Materiales

Para llevar a cabo este proyecto se ha hecho uso tanto de recursos *hardware* como de recursos *software,* según se vio en el capítulo de "*pliego de condiciones"* del presente proyecto.
Se estipula el coste de amortización para un período de 6 meses. Para ello, se utilizará un sistema de amortización lineal o constante, en el que se supone que el inmovilizado material se deprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula usando la siguiente fórmula:

$$Cuota\ Anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Números\ de\ años\ de\ vida\ útil}$$

El resultado esta operación tendrá que ser dividido por 2 para que corresponda a un período de 6 meses (tiempo empleado en la elaboración de este proyecto). El "valor residual" es el valor teórico que se supone que tendrá cada recurso después de su vida útil.

P1.1 Recursos *Hardware*

En la tabla de abajo podemos ver los recursos *Hardware* empleados, así como el coste de cada uno de ellos en el momento de la compra. También podemos ver el valor residual después de unos 3 años y por último el valor de amortización durante un período de seis meses.

Coste de los recursos *hardware*			
Recurso	Valor adquisición	Valor residual	Valor Amortización
1 placa de prototipado	350,00 €	50,00 €	30,00 €
1 protoboard	17,00 €	5,00 €	0,60 €
1 Cable de programación	70,00 €	20,00 €	5,00 €
1 Fuente de alimentación	25,00 €	10,00 €	0,75 €
1 osciloscopio	3.500,00 €	1.500,00 €	75,00 €
3 Sondas osciloscopio	180,00 €	50,00 €	6,50 €
Un generador de funciones	550,00 €	250,00 €	15,00 €
Un ordenador portátil	750,00 €	200,00 €	91,66 €
1 sensor RGB	25,00 €	5,00 €	0,40 €
Pequeño material (resistencias, condensadores,…)	0.80 €	0,00 €	0,04 €
COSTE TOTAL			224,91 €

P1.2 Recursos *software*

Los recursos software empleados en la realización de este proyecto son gratuitos. El ModelSim versión estudiante está disponible en la página web oficial de Mentor Graphics (https://www.mentor.com/products/fv/modelsim) y ISE Project Navigator en la de Xilinx (https://www.xilinx.com/support/download.htm).

P2. Trabajo tarifado por tiempo empleado

Para el cálculo de los honorarios obtenidos durante la elaboración de este proyecto empleamos la siguiente fórmula según las recomendaciones del COIT.

$$H = Ct\ x\ 74{,}88\ x\ Hn + Ct\ x\ 96{,}72\ x\ He\ \text{€}$$

Donde:

- *H* son los honorarios totales por el tiempo dedicado
- *Hn* son las horas normales trabajadas (dentro de la jornada laboral).

- *He* son las horas especiales.
- *Ct e*s un factor de corrección función del número de horas trabajadas.

Para la elaboración de este proyecto se ha invertido un total de 6 meses con una dedicación de 10 horas por día, lo que hace un total de 1.800 horas de trabajo.

Según el COIT, el coeficiente '*Ct'* tiene un valor variable en función del número de horas empleadas de acuerdo con la siguiente tabla:

Horas empleadas	Factor de corrección
Hasta 36	1,0
De 36 a 72	0,9
De 72 a 108	0,8
De 108 a 144	0,7
De 144 a 180	0,65
De 180 a 360	0,6
De 360 a 540	0,55
De 540 a 720	0,5
De 720 a 1080	0,45
Más de 1080	0,4

Como se citó anteriormente el número de horas totales empleadas han sido 1.800, por lo tanto el coeficiente '*Ct'* que se aplica según la tabla de arriba es de 0,4. Introduciendo los datos en la fórmula obtenemos:

$$H = 0{,}4\ x\ 74{,}88\ x\ 1.800 + 0{,}4\ x\ 96{,}72\ x\ 0 = 53.913{,}60\ €$$

Por lo que los honorarios totales por tiempo dedicado libres de impuestos ascienden a cincuenta y tres mil novecientos trece con sesenta céntimos (53.913,60 €).

P3. Costes de redacción del Trabajo Final de Carrera

Para el cáculo del coste de redacción del Trabajo Final de Carrera utilizamos la siguiente expresión:

Presupuesto

$$R = 0{,}07\ x\ P\ x\ Cn$$

Donde:

- *P* es el presupuesto del proyecto
- *Cn* es el coeficiente de ponderación en función del presupuesto

En la siguiente tabla podemos ver los gastos generados hasta el momento:

Recursos	Costes
Recursos Hardware	224,91 €
Recursos software	0,00 €
Trabajo tarifado por tiempo empleado	53.913,60 €
TOTAL	54.138,51 €

Como el presupuesto excede el valor de 30.050 €, el coeficiente de ponderación aplicable en este caso es de '*Cn'* = 0,9, por lo tanto el coste de redacción asciende a:

$$R = 0{,}07\ x\ 54.138{,}51\ x\ 0{,}9 = 3.410{,}72\ €$$

P4. Material fungible

Además de los recursos *hardware* y *software*, en este proyecto se han empleado otros materiales, como son los folios y el tóner de la impresora entre otros, que se nombran en la siguiente tabla.

Materiales	Costes
Folios	4,50 €
Tóner impresora	45,00 €
Encuadernación	3,75 €
TOTAL	53,25 €

P5. Derechos de visado del COITT

Los gastos de visado del COITT se tarifan mediante la siguiente expresión:

$$V = 0{,}006\ x\ P\ x\ Cv$$

Donde:

- *P* es el presupuesto del proyecto
- *Cv* es el coeficiente reductor en función del presupuesto del proyecto.

El presupuesto calculado hasta el momento asciende a la suma de los costes de ejecución material, de redacción y de material fungible.

$$P = 224{,}91\ € + 54.138{,}51\ € + 53{,}25\ € = 54.416{,}67\ €$$

El coeficiente de ponderación para presupuestos mayores de 30.050 € viene definido por el COITT con un valor de 0,9 el coste de los derechos de visado del proyecto asciende a la cantidad de:

$$V = 0{,}006\ x\ 54.416{,}67\ x\ 0{,}9 = 293{,}85\ €$$

P6. Aplicación de impuestos

El coste total del proyecto, antes de aplicarle los correspondientes impuestos, asciende a un total de 57.896,37 €, a lo que hay que sumarle el 7% de IGIC, con lo que el coste definitivo del Proyecto Final de Carrera es:

Descripción	Costes
Recursos materiales	224,95 €
Trabajo tarifado por tiempo empleado	53.913,60 €
Redacción del proyecto final de carrera	3.410,72 €
Material fungible	53,25 €
Derechos de visado	293,85 €
SUBTOTAL	57.896,37 €
Aplicación de impuestos (IGIC 7%)	4.052,74 €
TOTAL DE COSTES	**61.949,11 €**

El coste total del presupuesto asciende a la cantidad de *sesenta y un mil novecientos cuarenta y nueve euros con once céntimos (61.949,11 €).*

Las Palmas de Gran Canaria a 3 Julio del 2017

Fdo: Miguel Ángel Hernández Martínez

Anexo I

Descripción básica de la placa Digilent

ANEXO I

Descripción básica de la placa Digilent

Introduction

The Xilinx Spartan-3 Starter Kit provides a low-cost, easy-to-use development and evaluation platform for Spartan-3 FPGA designs.

Key Components and Features

Figure 1-1 shows the Spartan-3 Starter Kit board, which includes the following components and features:

- 200,000-gate Xilinx Spartan-3 XC3S200 FPGA in a 256-ball thin Ball Grid Array package (XC3S200FT256) (1)
  - 4,320 logic cell equivalents
  - Twelve 18K-bit block RAMs (216K bits)
  - Twelve 18x18 hardware multipliers
  - Four Digital Clock Managers (DCMs)
  - Up to 173 user-defined I/O signals
- 2Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM (2)
  - 1Mbit non-volatile data or application code storage available after FPGA configuration
  - Jumper options allow FPGA application to read PROM data or FPGA configuration from other sources (3)
- 1M-byte of Fast Asynchronous SRAM (bottom side of board, see Figure 1-3) (4)
  - Two 256Kx16 ISSI IS61LV25616AL-10T 10 ns SRAMs
  - Configurable memory architecture
    - Single 256Kx32 SRAM array, ideal for MicroBlaze code images
    - Two independent 256Kx16 SRAM arrays
  - Individual chip select per device
  - Individual byte enables
- 3-bit, 8-color VGA display port (5)
- 9-pin RS-232 Serial Port (6)
  - DB9 9-pin female connector (DCE connector)
  - RS-232 transceiver/level translator (7)
  - Uses straight-through serial cable to connect to computer or workstation serial port
  - Second RS-232 transmit and receive channel available on board test points (8)

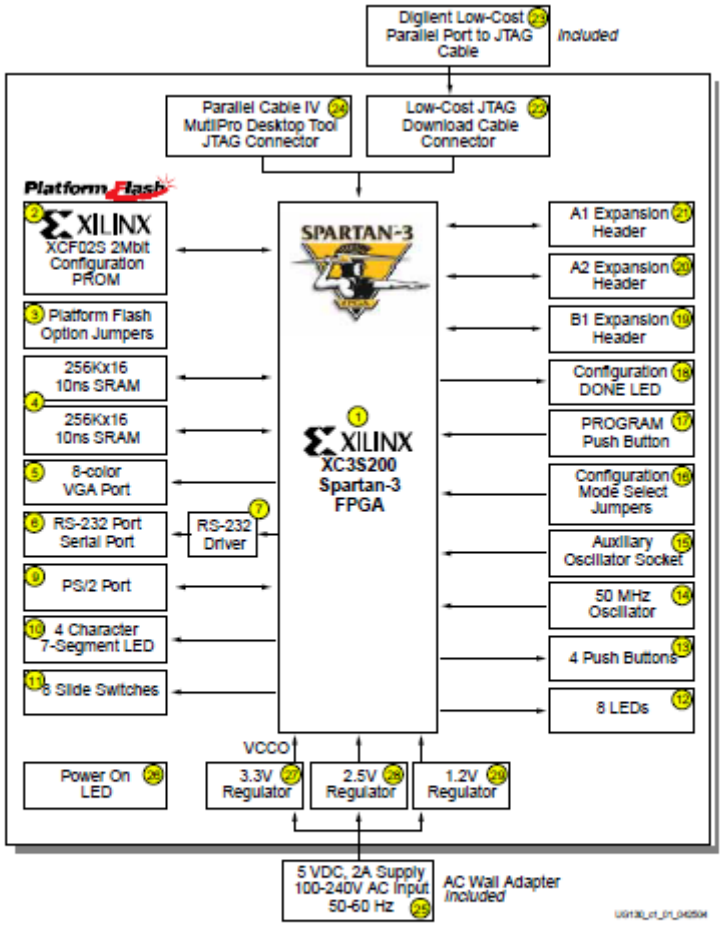


Figure 1-1: **Xilinx Spartan-3 Starter Kit Board Block Diagram**

- PS/2-style mouse/keyboard port (9)
- Four-character, seven-segment LED display (10)
- Eight slide switches (11)
- Eight individual LED outputs (12)

- 50 MHz crystal oscillator clock source (bottom side of board, see Figure 1-3) (14)
- Socket for an auxiliary crystal oscillator clock source (15)
- FPGA configuration mode selected via jumper settings (16)
- Push button switch to force FPGA reconfiguration (FPGA configuration happens automatically at power-on) (17)
- LED indicates when FPGA is successfully configured (18)
- Three 40-pin expansion connection ports to extend and enhance the Spartan-3 Starter Kit Board (19) (20) (21)
  - See www.xilinx.com/s3boards for compatible expansion cards
  - Compatible with Digilent, Inc. peripheral boards https://digilent.us/Sales/boards.cfm#Peripheral
  - FPGA serial configuration interface signals available on the A2 and B1 connectors
    - PROG_B, DONE, INIT_B, CCLK, DONE
- JTAG port (22) for low-cost download cable (23)
- Digilent JTAG download/debugging cable connects to PC parallel port (22)
- JTAG download/debug port compatible with the Xilinx Parallel Cable IV and MultiPRO Desktop Tool (24)
- AC power adapter input for included international unregulated +5V power supply (25)
- Power-on indicator LED (26)
- On-board 3.3V (27), 2.5V (28), and 1.2V (29) regulators

Component Locations

Figure 1-2 and Figure 1-3 indicate the component locations on the top side and bottom side of the board, respectively.

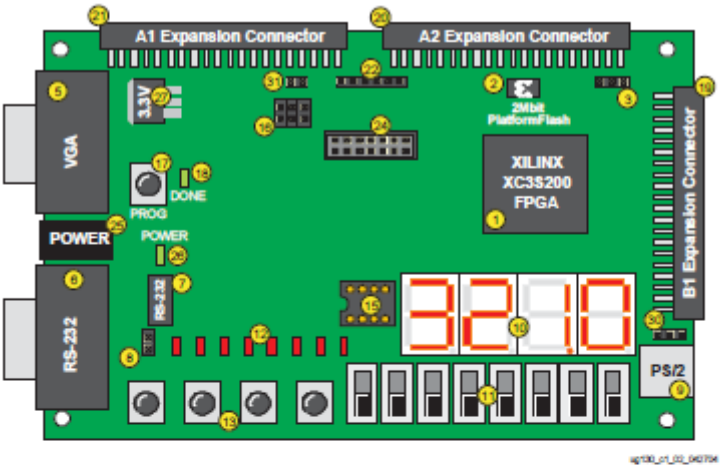


Figure 1-2: **Xilinx Spartan-3 Starter Kit Board (Top Side)**

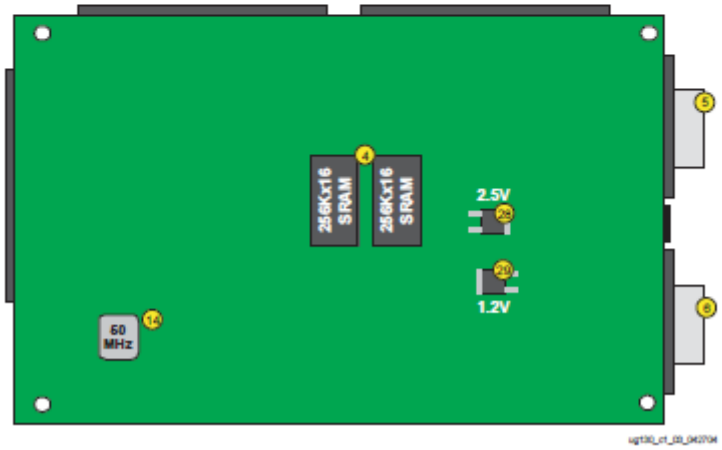


Figure 1-3: **Xilinx Spartan-3 Starter Kit Board (Bottom Side)**

Expansion Connectors

The Spartan-3 Starter Kit board has three 40-pin expansion connectors labeled A1, A2, and B1. The A1 and A2 connectors, indicated as (21) and (20), respectively, in Figure 1-2, are on the top edge of the board. Connector A1 is on the top left, and A2 is on the top right. The B1 connector, indicated as (19) in Figure 1-2, is along the right edge of the board.

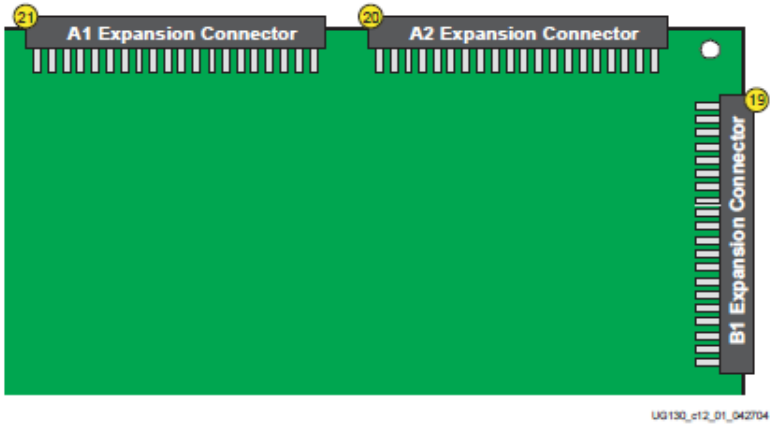


Figure 13-1: **Spartan-3 Starter Kit Board Expansion Connectors**

Table 13-1 summarizes the capabilities of each expansion port. Port A1 supports a maximum of 32 user I/O pins, while the other ports provide up to 34 user I/O pins. Some pins are shared with other functions on the board, which may reduce the effective I/O count for specific applications. For example, pins on the A1 port are shared with the SRAM address signals, with the SRAM OE# and WE# control signals, and with the eight least-significant data signals to SRAM IC10 only.

Table 13-1: **Expansion Connector Features**

Connector	User I/O	SRAM	JTAG	Serial Configuration	Parallel Configuration
A1	32	Address OE#, WE# Data[7:0] to IC10 only	√		
A2	34			√	
B1	34			√	√

Each port offers some ability to program the FPGA on the Spartan-3 Starter Kit Board. For example, port A1 provides additional logic to drive the FPGA and Platform Flash JTAG chain. Similarly, ports A2 and B1 provide connections for Master or Slave Serial mode configuration. Finally, port B1 also offers Master or Slave Parallel configuration mode.

Each 40-pin expansion header, shown in Figure 13-2, uses 0.1-inch (100 mil) DIP spacing. Pin 1 on each connector is always GND. Similarly, pin 2 is always the +5V DC output from the switching power supply. Pin 3 is always the output from the +3.3V DC regulator.

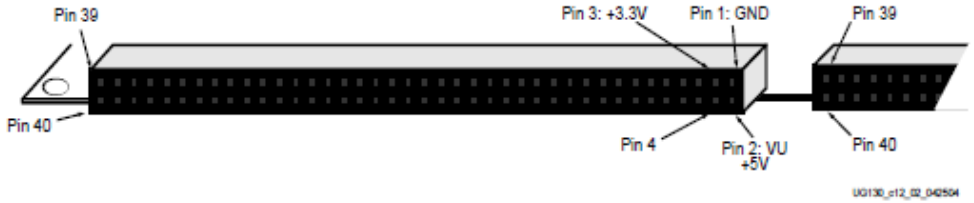


Figure 13-2: **40-pin Expansion Connector**

The pinout information for each connector appears below. The tables include the connections between the FPGA and the expansion connectors plus the signal names used in the detailed schematic in Figure A-1.

A1 Connector Pinout

The A1 expansion connector is located along the top edge of the board, on the left, as indicated by (21) in Figure 1-2. Table 13-2 provides the pinout for the A1 connector. The FPGA connections are specified in parentheses.

Table 13-2: **Pinout for A1 Expansion Connector**

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(N8)	ADR0
DB0	(N7) SRAM IC10 IO0	5	6	(L5) SRAM A0	ADR1
DB1	(T8) SRAM IC10 IO1	7	8	(N3) SRAM A1	ADR2
DB2	(R6) SRAM IC10 IO2	9	10	(M4) SRAM A2	ADR3
DB3	(T5) SRAM IC10 IO3	11	12	(M3) SRAM A3	ADR4
DB4	(R5) SRAM IC10 IO4	13	14	(L4) SRAM A4	ADR5
DB5	(C2) SRAM IC10 IO5	15	16	(G3) SRAM WE#	WE
DB6	(C1) SRAM IC10 IO6	17	18	(K4) SRAM OE#	OE
DB7	(B1) SRAM IC10 IO7	19	20	(P9) FPGA DOUT/BUSY	CSA
LSBCLK	(M7)	21	22	(M10)	MA1-DB0
MA1-DB1	(F3) SRAM A6	23	24	(G4) SRAM A5	MA1-DB2
MA1-DB3	(E3) SRAM A8	25	26	(F4) SRAM A7	MA1-DB4
MA1-DB5	(G5) SRAM A10	27	28	(E4) SRAM A9	MA1-DB6
MA1-DB7	(H4) SRAM A12	29	30	(H3) SRAM A11	MA1-ASTB
MA1-DSTB	(J3) SRAM A14	31	32	(J4) SRAM A13	MA1-WRITE
MA1-WAIT	(K5) SRAM A16	33	34	(K3) SRAM A15	MA1-RESET
MA1-INT	(L3) SRAM A17	35	36	JTAG Isolation	JTAG Isolation
TMS	(C13) FPGA JTAG TMS	37	38	(C14) FPGA JTAG TCK	TCK
TDO-ROM	Platform Flash JTAG TDO	39	40	Header J7, pin 3	TDO-A

The A1 expansion connector shares connections with the 256Kx16 SRAM devices, specifically the SRAM address lines, the OE# and WE# control signals, and the eight least-significant data lines to SRAM IC10 only. Similarly, the JTAG chain is available on pins 36 through 40. Pin 20 is the FPGA DOUT/BUSY configuration signal and toggles during the FPGA configuration process.

A2 Connector Pinout

The A2 expansion connector is located along the top edge of the board, on the right, as indicated by (20) in Figure 1-2. Figure 13-3 provides the pinout for the A2 connector. The FPGA connections are specified in parentheses.

Most of the A2 expansion connector pins connect only with the FPGA and are not shared. Pin 35 connects to the auxiliary clock socket, if an oscillator is installed in the socket. Pins 36 through 40 include the signals required to configure the FPGA in Master or Slave Serial mode.

Table 13-3: **Pinout for A2 Expansion Connector**

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3
PA-IO4	(D6)	7	8	(C6)	PA-IO5
PA-IO6	(E7)	9	10	(C7)	PA-IO7
PA-IO8	(D7)	11	12	(C8)	PA-IO9
PA-IO10	(D8)	13	14	(C9)	PA-IO11
PA-IO12	(D10)	15	16	(A3)	PA-IO13
PA-IO14	(B4)	17	18	(A4)	PA-IO15
PA-IO16	(B5)	19	20	(A5)	PA-IO17
PA-IO18	(B6)	21	22	(B7)	MA2-DB0
MA2-DB1	(A7)	23	24	(B8)	MA2-DB2
MA2-DB3	(A8)	25	26	(A9)	MA2-DB4
MA2-DB5	(B10)	27	28	(A10)	MA2-DB6
MA2-DB7	(B11)	29	30	(B12)	MA2-ASTB
MA2-DSTB	(A12)	31	32	(B13)	MA2-WRITE
MA2-WAIT	(A13)	33	34	(B14)	MA2-RESET
MA2-INT/GCK4	(D9) Oscillator socket	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

B1 Connector Pinout

The B1 expansion connector is located on the right edge of the board, as indicated by (19) in Figure 1-2. Table 13-4 provides the pinout for the B1 connector. The FPGA connections are specified in parentheses.

Most of the B1 expansion connector pins connect only with the FPGA and are not shared. Pins 36 through 40 include the signals required to configure the FPGA in Master or Slave Serial mode. These same pins plus pins 5, 7, 9, 11, 13, 15, 17, 19, and 20 provide the signals required to configure the FPGA in Master or Slave Parallel mode.

Table 13-4: **Pinout for B1 Expansion Connector**

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(C10)	PB-ADR0
PB-DB0	(T3) FPGA RD_WR_B config	5	6	(E10)	PB-ADR1
PB-DB1	(N11) FPGA D1 config	7	8	(C11)	PB-ADR2
PB-DB2	(P10) FPGA D2 config	9	10	(D11)	PB-ADR3
PB-DB3	(R10) FPGA D3 config	11	12	(C12)	PB-ADR4
PB-DB4	(T7) FPGA D4 config	13	14	(D12)	PB-ADR5
PB-DB5	(R7) FPGA D5 config	15	16	(E11)	PB-WE
PB-DB6	(N6) FPGA D6 config	17	18	(B16)	PB-OE
PB-DB7	(M6) FPGA D7 config	19	20	(R3) FPGA CS_B config	PB-CS
PB-CLK	(C15)	21	22	(C16)	MB1-DB0
MB1-DB1	(D15)	23	24	(D16)	MB1-DB2
MB1-DB3	(E15)	25	26	(E16)	MB1-DB4
MB1-DB5	(F15)	27	28	(G15)	MB1-DB6
MB1-DB7	(G16)	29	30	(H15)	MB1-ASTB
MB1-DSTB	(H16)	31	32	(J16)	MB1-WRITE
MB1-WAIT	(K16)	33	34	(K15)	MB1-RESET
MB1-INT	(L15)	35	36	(B3) FPGA PROG_B	PROG-B
DONE	(R14) FPGA DONE	37	38	(N9) FPGA INIT_B	INIT
CCLK	(T15) FPGA CCLK Connects to (A14) via 390Ω resistor	39	40	(M11)	DIN

Anexo II

Data sheet del sensor RGB

ANEXO II

Data sheet del sensor RGB

HAMAMATSU

PHOTON IS OUR BUSINESS

Digital color sensor

S9706

12-bit digital output

The S9706 is a digital color sensor sensitive to red (λ=615 nm), green (λ=540 nm) and blue (λ=465 nm) regions of the spectrum. Detected signals are serially output as 12-bit digital data. Built-in three 12-bit registers allow simultaneous measurement of RGB three colors. Sensitivity level is adjustable in two steps to cover a wide photometric range.

Features

- 12-bit digital output
- Simultaneous measurement of RGB three colors
- 2-step sensitivity switching (sensitivity ratio of 1 : 9)
- Low voltage (3.3 V) operation
- CMOS monolithic photo IC
- No external components required

Applications

- Display color adjustment
- Various applications involving color detection

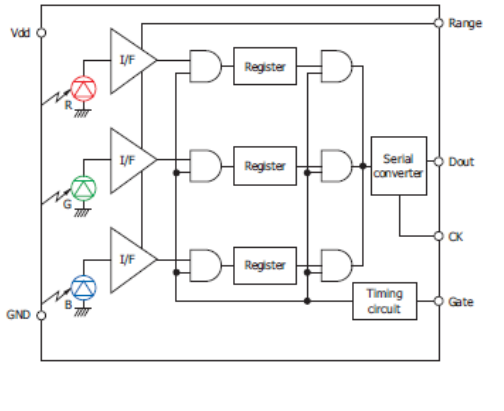
Feature 01 12-bit digital output

Light signals detected by the photodiode are amplified and converted into 12-bit digital signals. An amplifier is also formed for each of the RGB photodiode elements arrayed in the mosaic pattern, allowing simultaneous accurate measurement of the RGB components of incident light.

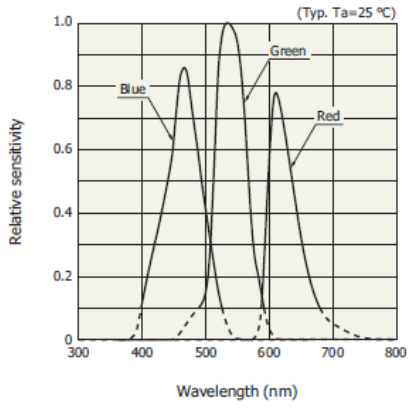
Feature 02 Simultaneous measurement of RGB three colors

The photodiode consists of 9 × 9 elements arrayed in a mosaic pattern. Each element has an on-chip filter that it sensitive to one color of light, either red (λp=615 nm), green (λp=540 nm) or blue (λp=465 nm).

Block diagram



Spectral response



This product does not support lead-free soldering. Solder it by hand.

Digital color sensor | **S9706**

Feature 03 2-step sensitivity switching

To enable measurement over a wide range of illuminance, the photodiode sensitivity can be selected from two setting modes (high sensitivity mode and low sensitivity mode). The photodiode photosensitive area used to detect light differs depending on which sensitivity mode is selected (high sensitivity mode: 9 × 9 elements, low sensitivity mode: 3 × 3 elements in center).

Sensitivity setting

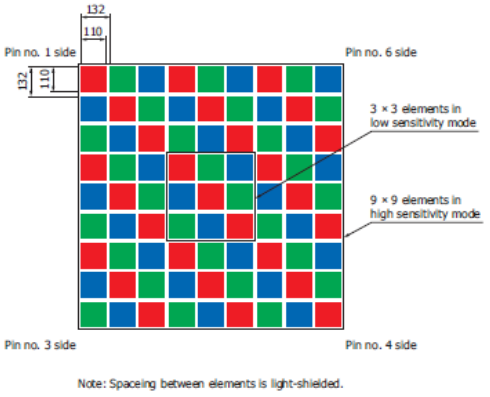
Range	Mode	Effective photosensitive area*
High	High sensitivity	9 × 9 elements
Low	Low sensitivity	3 × 3 elements

* The photosensitive area of S9706 consists of 9 × 9 elements in a mosaic pattern.

The effective photosensitive area changes depending on which sensitivity mode is used, "high" or "low", as explained below.

- High sensitivity mode: 9 × 9 elements
- Low sensitivity mode: 3 × 3 elements in center

Details of photosensitive area (unit: µm)



Absolute maximum ratings

Parameter	Symbol	Condition	Value	Unit
Supply voltage	Vdd	Ta=25 °C	-0.3 to 6	V
Load current	Io	Ta=25 °C	±10	mA
Power dissipation	P	Ta=25 °C	100	mW
Operating temperature	Topr		-20 to +85	°C
Storage temperature	Tstg		-20 to +85	°C

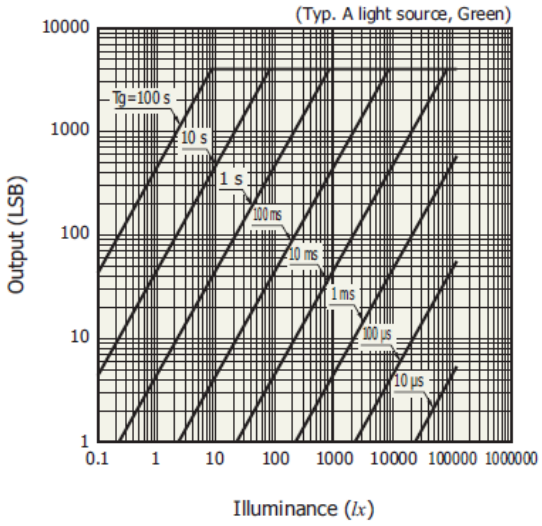
Note: Exceeding the absolute maximum ratings even momentarily may cause a drop in product quality. Always be sure to use the product within the absolute maximum ratings.

Electrical and optical characteristics (Ta=25 °C, Vdd=5 V, Tg=100 ms, A light source, unless otherwise noted)

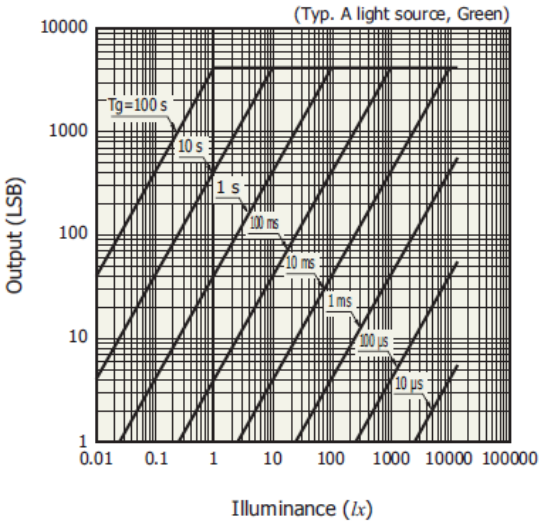
Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Photosensitive area size	-	All elements (9 × 9 elements)	-	1.2 × 1.2	-	mm
Effective photosensitive area	-	Per 1 color, High range	-	0.32	-	mm^2
Spectral response range	λ	Blue	-	400 to 540	-	nm
		Green	-	480 to 600	-	
		Red	-	590 to 720	-	
Peak sensitivity wavelength	λp	Blue	-	465	-	nm
		Green	-	540	-	
		Red	-	615	-	
Supply voltage	Vdd		3.0	-	5.5	V
Current consumption	Idd	Dark state, no load	-	5	10	mA
Photosensitivity	Sbl	Blue, Low range	0.15	0.21	0.27	LSB/lx
	Sgl	Green, Low range	0.32	0.45	0.59	
	Srl	Red, Low range	0.45	0.64	0.83	
	Sbh	Blue, High range	1.3	1.9	2.5	
	Sgh	Green, High range	2.8	4.1	5.4	
	Srh	Red, High range	4.0	5.8	7.6	
Incident light power (Conversion value in A light source)	Ibl	Blue, Low range	-	-	240	klx
	Igl	Green, Low range	-	-	110	
	Irl	Red, Low range	-	-	78	
	Ibh	Blue, High range	-	-	26	
	Igh	Green, High range	-	-	12	
	Irh	Red, High range	-	-	8.6	
Dark output	Dark	Tg=0.5 s	-	-	1	LSB
Input high level	Vih		Vdd × 0.82	-	-	V
Input low level	Vil		-	-	Vdd × 0.18	V
High level output voltage	Voh	Ioh=-0.5 mA	4.5	-	-	V
Low level output voltage	Vol	Iol=0.5 mA	-	-	0.5	V
Integration time	Tg		Refer to "Output vs. illuminance"			-
Hold time	t1		4	-	-	µs
	t2		3	-	-	µs
	t3		3	-	-	µs
	t4		2000	-	-	µs
	t5		3	-	-	µs
Readout clock period	tck		500	-	-	ns
Readout pulse width (positive)	tw		200	-	-	ns
Readout pulse width (negative)	tck-tw		200	-	-	ns

Output vs. illuminance

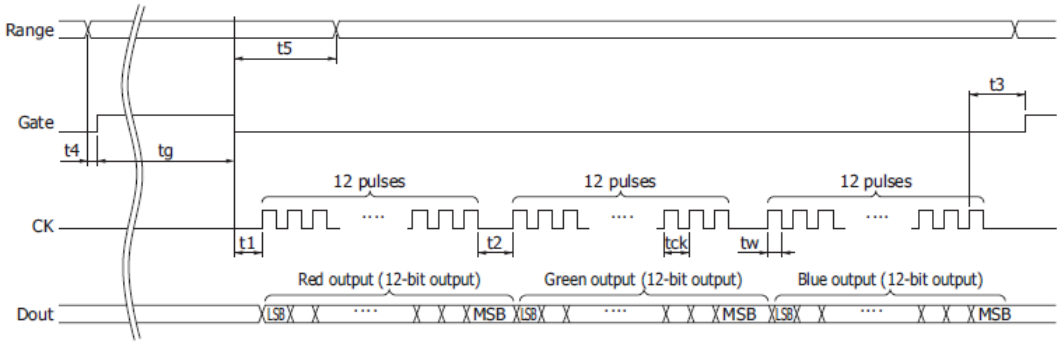
Low range



High range



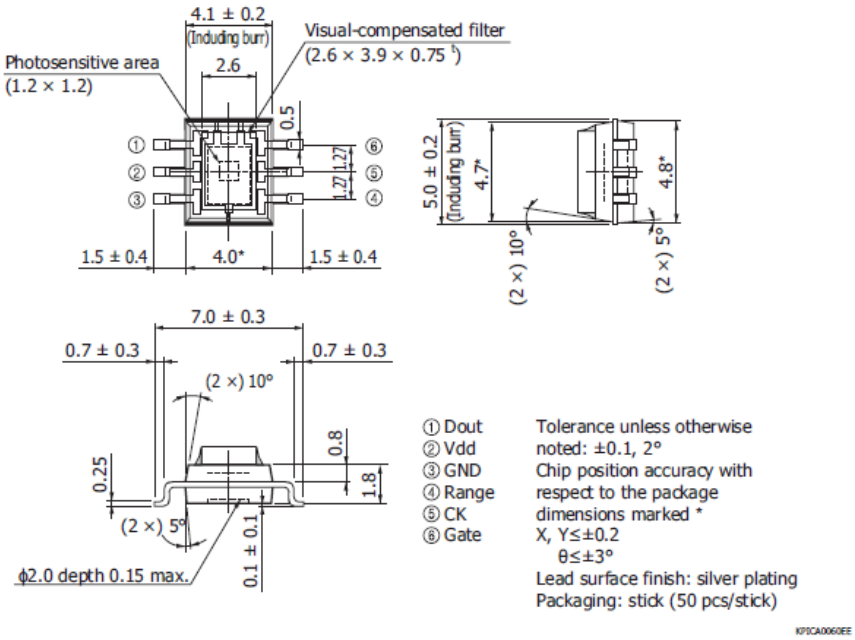
Timing chart



Operating sequence
(1) Set the Gate terminal and CK terminal to "Low".
(2) Select the desired sensitivity with the Range terminal.
(3) Set the Gate terminal from "Low" to "High", to start integrating the light intensity.
(4) After the desired integration time (tg) has passed, set the Gate terminal from "High" to "Low" to end the light intensity integration.
(5) Measurement data is output from the Dout terminal by inputting 36 CK pulses to the CK terminal.

Note 1: A total of 36 CK pulses are required to read out 3-color measurement data. Red data is output by the first 12 pulses, green data by the next 12 pulses, and blue data by the last 12 pulses. Measurement data is output from the LSB side.
Note 2: Measurement data changes at the CK pulse rising edge.
Note 3: Do not switch the Range terminal during integration time (tg).

Dimensional outline (unit: mm)



Note: If excessive vibration is continuously applied to the glass filter, there is a risk that the filter may come off, so secure the glass filter with a holder.

Line-up of RGB color sensors

Type no.	Type	Photosensitive area (mm)	Package (mm)	Peak sensitivity wavelength (nm)		Photosensitivity						Photo
S9032-02	Photodiode	ϕ2.0	4 × 4.8 × 1.8t 6 pin (filter 0.75t)	B	460	B	0.18 (A/W) [λ=460 nm]					
				G	540	G	0.23 (A/W) [λ=540 nm]					
				R	620	R	0.16 (A/W) [λ=620 nm]					
S9702	Photodiode	1.0 × 1.0	3 × 4 × 1.3t 4 pin (filter 0.75t)	B	460	B	0.18 (A/W) [λ=460 nm]					
				G	540	G	0.23 (A/W) [λ=540 nm]					
				R	620	R	0.16 (A/W) [λ=620 nm]					
S10917-35GT	Photodiode	1.0 × 1.0	3 × 1.6 × 1.0t COB (on-chip filter)	B	460	B	0.2 (A/W) [λ=460 nm]					
				G	540	G	0.23 (A/W) [λ=540 nm]					
				R	620	R	0.17 (A/W) [λ=620 nm]					
S10942-01CT	Photodiode	1.0 × 1.0	3 × 1.6 × 1.0t COB (on-chip filter)	*		B	0.21 (A/W) [λ=460 nm]					
						G	0.25 (A/W) [λ=540 nm]					
						R	0.45 (A/W) [λ=640 nm]					
S9706	Digital photo IC	1.2 × 1.2	4 × 4.8 × 1.8t 6 pin (filter 0.75t)	B	465	Low	B	0.21 (LSB/lx)	High	B	1.9 (LSB/lx)	
				G	540		G	0.45 (LSB/lx)		G	4.1 (LSB/lx)	
				R	615		R	0.64 (LSB/lx)		R	5.8 (LSB/lx)	
S11012-01CR	Digital photo IC	1.2 × 1.2	3.43 × 3.8 × 1.6t COB (on-chip filter)	*		Low	B	0.3 (LSB/lx)	High	B	2.6 (LSB/lx)	
							G	0.6 (LSB/lx)		G	5.3 (LSB/lx)	
							R	1.4 (LSB/lx)		R	12.9 (LSB/lx)	
S11059-02DT /-03DS	I²C compatible color sensor	0.56 × 1.22	3 × 4.2 ×1.3t 10 pin (on-chip filter)	B	460	Low	B	4.4 (count/lx)	High	B	44.8 (count/lx)	
				G	530		G	8.3 (count/lx)		G	85.0 (count/lx)	
				R	615		R	11.2 (count/lx)		R	117.0 (count/lx)	
				IR	855		IR	3.0 (count/lx)		IR	30.0 (count/lx)	
S11059-01WT	I²C compatible color sensor	1.22 × 0.56	1.68 × 1.18 ×0.58t WL-CSP (on-chip filter)	B	460	Low	B	3.35 (count/lx)	High	B	31.7 (count/lx)	
				G	530		G	7.61 (count/lx)		G	76.2 (count/lx)	
				R	615		R	9.48 (count/lx)		R	94.5 (count/lx)	
				IR	855		IR	1.66 (count/lx)		IR	15.3 (count/lx)	

* Refer to the spectral response of each product's datasheet.

Anexo III

Códigos VHDL

ANEXIO III

Códigos VHDL

CLK SCL

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity clkscale is
    Port ( clk : in  STD_LOGIC;
           divsel : in  STD_LOGIC_VECTOR (2 downto 0);
           rst : in  STD_LOGIC;
           clk_out : out  STD_LOGIC);
end clkscale;

architecture Behavioral of clkscale is
-- como tenemos que dividir por múltiplos de 2 hasta 128, escogemos
--un contador binario de 7 bits. El peso de cada bit nos dará la división
-- en múltiplos de 2 según dicho peso. Por ejemplo el bit nº 7 nos dá una
--división de 128 y el bit 1º por 2.
signal contador : std_logic_vector (6 downto 0) := "0000000";

begin

        process (clk, rst)
        begin

                if rst = '1' then
                        clk_out <= '0';
                        contador <= "0000000";

                elsif clk'event and clk = '1' then
                        contador <= contador + "0000001";
                        case divsel is
                                when "001" =>
                                        clk_out <= contador(0);-- dividimos por dos
                                when "010" =>
                                        clk_out <= contador(1);-- dividimos por 4
                                when "011" =>
                                        clk_out <= contador(2);-- dividimos por 8
                                when "100" =>
                                        clk_out <= contador(3);-- dividimos por 16
                                when "101" =>
                                        clk_out <= contador(4);-- dividimos por 32
                                when "110" =>
                                        clk_out <= contador(5);-- dividimos por 64
                                when "111" =>
                                        clk_out <= contador(6);-- dividimos por 128
                                when others =>
                                        clk_out <= 'Z';
                        end case;

                end if;
end process;


end Behavioral;
```

COLOR READER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity color_reader is

   Port ( rst_CR       : in  STD_LOGIC;
            sinc_in : in  STD_LOGIC;
            data_in_CR    : in  STD_LOGIC;
            clk_CR       : in  STD_LOGIC;
          --SEÑALES DE SALIDA DE DEMUX
            data_out_demux  : out  STD_LOGIC_VECTOR (2 DOWNTO 0);

          --SEÑALES DE ENABLE REG DESP COLOR READER
            R_en_demux_CR     : out STD_LOGIC;
            G_en_demux_CR     : out STD_LOGIC;
            B_en_demux_CR     : out STD_LOGIC;

            sel_out_CR   : out STD_LOGIC_VECTOR(1 downto 0);


            gate_CR      : out  STD_LOGIC;
            ck_CR        : out  STD_LOGIC;

            data_out_R_CR : out STD_LOGIC_VECTOR (7 downto 0);
            data_out_G_CR : out STD_LOGIC_VECTOR (7 downto 0);
            data_out_B_CR : out STD_LOGIC_VECTOR (7 downto 0));

end color_reader;

architecture Behavioral of color_reader is


  component re_desp_8
          Port ( rst_desp    : in  STD_LOGIC;
            en_desp : in  STD_LOGIC;
            clk_desp     : in STD_LOGIC;
            D_in_desp     : in STD_LOGIC;
            D_out_desp    : out STD_LOGIC_VECTOR(7 downto 0));
  end component;

  component control_reader1
           Port (clk : in  STD_LOGIC;
            sync  : in  STD_LOGIC;
            rst  : in STD_LOGIC;
            gate : out  STD_LOGIC;
            ck : out  STD_LOGIC;
            R_en : out  STD_LOGIC;
            G_en : out  STD_LOGIC;
            B_en : out  STD_LOGIC;
            num_pulsos : out STD_LOGIC_VECTOR(3 DOWNTO 0);
            sel : out  STD_LOGIC_VECTOR(1 DOWNTO 0));
  end component;

  component demux1_3
          PORT( data_in_demux   : in  STD_LOGIC;
              sel_demux          : in  STD_LOGIC_VECTOR (1 DOWNTO 0);
              data_out_demux  : out  STD_LOGIC_VECTOR (2 DOWNTO 0));
  end component;
```

```
        --reg_desp
        signal s_rst_desp :   STD_LOGIC;
        signal s_en_desp_R :   STD_LOGIC;
        signal s_en_desp_G :   STD_LOGIC;
        signal s_en_desp_B :   STD_LOGIC;
        signal s_clk_desp :  STD_LOGIC;
        signal s_D_in_desp_R :  STD_LOGIC;
        signal s_D_in_desp_G :  STD_LOGIC;
        signal s_D_in_desp_B :  STD_LOGIC;
        signal s_D_out_desp_R    : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_out_desp_G    : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_out_desp_B    : STD_LOGIC_VECTOR(7 downto 0);



        --demux
        signal s_data_in_demux : std_logic;
        signal s_sel_demux : std_logic_vector(1 downto 0);
        signal s_data_out_demux : std_logic_vector(2 downto 0);

        signal s_R_out_mux_CR : std_logic;
        signal s_G_out_mux_CR : std_logic;
        signal s_B_out_mux_CR : std_logic;



        --control_reader
          signal s_clk :   STD_LOGIC;
          signal s_sync :  STD_LOGIC;
          signal s_rst : STD_LOGIC;
          signal s_gate :  STD_LOGIC;
          signal s_ck : STD_LOGIC;
          signal s_R_en : STD_LOGIC;
          signal s_G_en : STD_LOGIC;
          signal s_B_en : STD_LOGIC;
          signal s_num_pulsos : STD_LOGIC_VECTOR(3 DOWNTO 0);
          signal s_sel :  STD_LOGIC_VECTOR(1 DOWNTO 0);

begin

  cont_reader: control_reader1 port map
(s_clk,s_sync,s_rst,s_gate,s_ck,s_R_en,s_G_en,s_B_en,s_num_pulsos,s_sel);
  r_desp_R: re_desp_8 port map(s_rst_desp, s_en_desp_R,s_clk_desp,s_D_in_desp_R,s_D_out_desp_R);
  r_desp_G: re_desp_8 port map(s_rst_desp, s_en_desp_G,s_clk_desp,s_D_in_desp_G,s_D_out_desp_G);
  r_desp_B: re_desp_8 port map(s_rst_desp, s_en_desp_B,s_clk_desp,s_D_in_desp_B,s_D_out_desp_B);
  demux: demux1_3 port map(s_data_in_demux,s_sel,s_data_out_demux);

 --entradas color_reader

  s_rst <= rst_CR;
  s_sync <= sinc_in;
  s_data_in_demux <= data_in_CR;
  s_clk <= clk_CR;



-- salidas color reader
  data_out_demux <= s_data_out_demux;
  R_en_demux_CR <= s_R_en;
  G_en_demux_CR <= s_G_en;
  B_en_demux_CR <= s_B_en;
  sel_out_CR <= s_sel;
  data_out_R_CR <= s_D_out_desp_R;
  data_out_G_CR<= s_D_out_desp_G;
  data_out_B_CR <= s_D_out_desp_B;
  gate_CR <= s_gate;
  ck_CR <= s_ck;
```

```
-- salidas demux
   s_D_in_desp_R <= s_data_out_demux(0);
   s_D_in_desp_G <= s_data_out_demux(1);
   s_D_in_desp_B <= s_data_out_demux(2);


-- entradas re_desp
   s_rst_desp <= s_rst;
   s_clk_desp <= s_ck;
   s_en_desp_R <= s_R_en;
   s_en_desp_G <= s_G_en;
   s_en_desp_B <= s_B_en;

-- salidas re_desp
   data_out_R_CR <= s_D_out_desp_R;
   data_out_G_CR <= s_D_out_desp_G;
   data_out_B_CR <= s_D_out_desp_B;

end Behavioral;
```

CONTROL READER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity control_reader1 is
    Port ( clk : in  STD_LOGIC;
                sync : in  STD_LOGIC;
                rst : in STD_LOGIC;
                gate : out  STD_LOGIC;
                ck : out  STD_LOGIC;
                R_en : out  STD_LOGIC;
                G_en : out  STD_LOGIC;
                B_en : out  STD_LOGIC;
                num_pulsos : out STD_LOGIC_VECTOR(3 DOWNTO 0);
                sel : out  STD_LOGIC_VECTOR(1 DOWNTO 0));
end control_reader1;

architecture Behavioral of control_reader1 is

type estados is (S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10);

signal estado:estados;
signal contador : std_logic_vector(8 downto 0):="000000000";
signal clk_out : std_logic;
signal sync_in : std_logic;

signal contador_pulsos : std_logic_vector(3 downto 0):="0000";
signal sel_out : std_logic_vector(1 downto 0):="00";
signal s_num_pulsos : std_logic_vector(3 downto 0):="0000";
signal s_cont_sel : std_logic_vector(3 downto 0):="0000";
signal s_cont_ck : std_logic_vector(3 downto 0):="0000";
```

```
begin


--salidas

sel <= sel_out;
num_pulsos <= s_num_pulsos;



	process(clk, rst, sync)
	begin
		if (rst='1') then
			estado <= S0;
			gate <='1';
			ck <='0';
			R_en <='0';
			B_en <='0';
			G_en <='0';
			sel_out <="00";
		elsif (clk='1' and clk'event) then
		case estado is
			when S0 =>
				if (sync = '1') then
					estado <= S1;
				else
					ck <='0';
					R_en <='0';
					B_en <='0';
					G_en <='0';
					sel_out <="00";
					estado <= S0;
				end if;
			when S1 =>
				if (sync = '0') then
					estado <= S2;
				else

					estado <= S1;
				end if;
			when S2 =>

				contador <= contador + 1;
				if(contador = "111110100" )then  --tg=10us (contador 500 pulsos de reloj
f=50 Mhz, T=0'02us)
					contador <= "000000000"; --reseteamos contador
					gate <='0';
					estado <= S3;			-- pasamos a S2

				else
					estado <= S2;			 -- seguimos en S1

				end if;

			when S3 =>

				contador <= contador + 1;
				if (contador = "011001000") then --t1=4us (contador 200 pulsos de reloj
f=50 Mhz, T=0'02us)
					contador <= "000000000"; --reseteamos contador
					estado <= S4;			-- pasamos a S3
				else

					estado <= S3;

				end if;

			when S4 =>
				s_cont_sel <= s_cont_sel+1;
```

```
                    if (s_cont_sel = "011") then
                        s_cont_sel <= "0000";

                        estado <= S10;
                    else
                        estado <= S5;
                    end if;

                when S5 =>

                    ck <='1';

                    if (contador = "000001000") then   -- tw= 0.26us (contador = 8)
                        contador <= "000000000";

                        estado <= S6;
                    else
                        contador <= contador + 1;
                        estado <= S5;
                    end if;


                when S6 =>
                    s_num_pulsos <= s_num_pulsos + 1;
                    if (s_num_pulsos = "1011") then  -- num pulsos = 12 ya se leyeron los 12
bits del sensor
                        R_en <= '0';
                        G_en <= '0';
                        B_en <= '0';
                        sel_out <= "00";
                        ck <= '0';
                        s_num_pulsos <= "0000";
                        s_cont_ck <= "0000";
                        estado <= S9;
                    else
                        estado <= S7;

                    end if;

                when S7 =>


                    s_cont_ck <= s_cont_ck+1;
                    if (s_cont_ck="0011") then   -- ck=4


                        if (s_cont_sel = "0001") then  -- Rojo

                            sel_out <= "01";
                            R_en <= '1';
                            G_en <= '0';
                            B_en <= '0';

                            estado <= S8;

                        elsif (s_cont_sel = "0010") then -- Verde
                            sel_out <= "10";
                            R_en <= '0';
                            G_en <= '1';
                            B_en <= '0';

                            estado <= S8;
                        elsif (s_cont_sel = "0011") then -- Azul
                            sel_out <= "11";
                            R_en <= '0';
                            G_en <= '0';
                            B_en <= '1';

                            estado <= S8;
```

```
                                        else
                                                estado <= S8;

                                        end if;
                                else

                                        estado <= S8;

                                end if;



                        when S8 =>
                                ck <='0';

                                if (contador = "000001000") then   -- tw= 0.26us (contador = 8)
                                        contador <= "000000000";

                                        estado <= S5;
                                else
                                        contador <= contador + 1;
                                        estado <= S8;
                                end if;


                        when S9 =>

                                contador <= contador + 1;
                                if (contador = "010010110") then --t2=3us (contador 150 pulsos de reloj
f=50 Mhz, T=0'02us)
                                        contador <= "000000000"; --reseteamos contador
                                        estado <= S4;                   -- pasamos a S3
                                else

                                        estado <= S9;

                                end if;

                        when S10 =>

                                if (contador = "010010110") then   -- t3= 3us (contador = 150)
                                        contador <= "000000000";
                                        gate <= '1';
                                        estado <= S0;
                                else
                                        contador <= contador + 1;
                                        estado <= S10;
                                end if;
                end case;
                end if;
        end process;


end Behavioral;
```

DATA PROCESS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
```

```
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity data_process is

   Port ( rst_DP       : in  STD_LOGIC;

               gate_in_DP    : in  STD_LOGIC;
               clk_in_DP     : in  STD_LOGIC;

                     --salidas DE COLOR READER R G B
               data_in_R_DP  : in  STD_LOGIC_VECTOR (7 downto 0);
               data_in_G_DP  : in  STD_LOGIC_VECTOR (7 downto 0);
               data_in_B_DP  : in  STD_LOGIC_VECTOR (7 downto 0);

                     --SALIDAS DE MATRIX R G B
               D_out_m_R0  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R7  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R15  : out STD_LOGIC_VECTOR(7 downto 0);

                     --SALIDAS DE MEDIA R G B
               D_in_media_R  : OUT STD_LOGIC_VECTOR(7 downto 0);
               D_in_media_G  : OUT STD_LOGIC_VECTOR(7 downto 0);
               D_in_media_B  : OUT STD_LOGIC_VECTOR(7 downto 0);

                     --SALIDA GATE HACIA COLOR READER
               read_color_DP : in  STD_LOGIC;

                     --SALIDA DE DATOS
               data_out_DP   : out  STD_LOGIC_VECTOR(1 downto 0));

end data_process;

architecture Behavioral of data_process is


  component matrix_R_G_B
            Port ( rst_m    : in  STD_LOGIC;
               en_m : in  STD_LOGIC;
               clk_m     : in STD_LOGIC;

               D_in_m_R     : in STD_LOGIC_VECTOR(7 downto 0);
               D_in_m_G     : in STD_LOGIC_VECTOR(7 downto 0);
               D_in_m_B     : in STD_LOGIC_VECTOR(7 downto 0);

               D_out_m_R0  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R1  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R2  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R3  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R4  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R5  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R6  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R7  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R8  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R9  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R10 : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R11 : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_R12   : out STD_LOGIC_VECTOR (7 downto 0);
               D_out_m_R13   : out STD_LOGIC_VECTOR (7 downto 0);
               D_out_m_R14   : out STD_LOGIC_VECTOR (7 downto 0);
               D_out_m_R15   : out STD_LOGIC_VECTOR (7 downto 0);

               D_out_m_G0  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G1  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G2  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G3  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G4  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G5  : out STD_LOGIC_VECTOR(7 downto 0);
               D_out_m_G6  : out STD_LOGIC_VECTOR(7 downto 0);
```

```
            D_out_m_G7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G11 : out STD_LOGIC_VECTOR(7 downto 0);
           D_out_m_G12   : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_G13   : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_G14   : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_G15   : out STD_LOGIC_VECTOR (7 downto 0);


            D_out_m_B0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B12  : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_B13  : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_B14  : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_B15  : out STD_LOGIC_VECTOR (7 downto 0));

end component;

component media_R_G_B
          Port ( rst_m_R_G_B : in  STD_LOGIC;
            en_m_R_G_B  : in  STD_LOGIC;

            D_in_m_R0    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R1    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R2    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R3    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R4    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R5    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R6    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R7    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R8    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R9    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R10   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R11   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R12   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R13   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R15   : in STD_LOGIC_VECTOR (7 downto 0);

            D_in_m_G0    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G1    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G2    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G3    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G4    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G5    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G6    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G7    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G8    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G9    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G10   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G11   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G12   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G13   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G15   : in STD_LOGIC_VECTOR (7 downto 0);

            D_in_m_B0    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B1    : in STD_LOGIC_VECTOR (7 downto 0);
```

```
            D_in_m_B2    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B3    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B4    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B5    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B6    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B7    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B8    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B9    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B10   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B11   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B12   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B13   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B15   : in STD_LOGIC_VECTOR (7 downto 0);


            D_out_m_R    : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_G    : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_B    : out STD_LOGIC_VECTOR (7 downto 0));


  end component;

  component decisor
          Port ( rst_dec    : in  STD_LOGIC;
            en_dec     : in  STD_LOGIC;

            D_in_media_R  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_G  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_B  : in STD_LOGIC_VECTOR(7 downto 0);

            D_in_color_R  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_color_G  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_color_B  : in STD_LOGIC_VECTOR(7 downto 0);

            D_out_dec    : out STD_LOGIC_VECTOR(1 downto 0));
  end component;

component data_process_control
           Port ( rst_C    : in  STD_LOGIC;
            clk_C       : in  STD_LOGIC;
            gate_C      : in  STD_LOGIC;
            read_color_C  : in  STD_LOGIC;
            clk_mtx_C          : out STD_LOGIC;
            en_mtx_C           : out STD_LOGIC;
            med_en_C           : out STD_LOGIC;
            dec_en_C           : out STD_LOGIC);
  end component;

          --reset
          signal s_rst_DP      : STD_LOGIC;

          --matrix_R_G_B
          signal s_en_m      : STD_LOGIC;
          signal s_clk_m     : STD_LOGIC;

          signal s_in_mtx_R_DP   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_mtx_G_DP   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_mtx_B_DP   : STD_LOGIC_VECTOR(7 downto 0);

          signal s_in_media_R0   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R1   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R2   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R3   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R4   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R5   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R6   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R7   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R8   : STD_LOGIC_VECTOR(7 downto 0);
          signal s_in_media_R9   : STD_LOGIC_VECTOR(7 downto 0);
```

```
signal s_in_media_R10  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_R11  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_R12  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_R13  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_R14  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_R15  : STD_LOGIC_VECTOR(7 downto 0);


signal s_in_media_G0   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G1   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G2   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G3   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G4   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G5   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G6   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G7   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G8   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G9   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G10  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G11  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G12  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G13  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G14  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_G15  : STD_LOGIC_VECTOR(7 downto 0);

signal s_in_media_B0   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B1   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B2   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B3   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B4   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B5   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B6   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B7   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B8   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B9   : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B10  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B11  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B12  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B13  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B14  : STD_LOGIC_VECTOR(7 downto 0);
signal s_in_media_B15  : STD_LOGIC_VECTOR(7 downto 0);

--media
signal s_en_m_R_G_B : STD_LOGIC;

signal s_media_en : STD_LOGIC;


--data process

signal s_clk_C        : STD_LOGIC;
signal s_gate_C       : STD_LOGIC;
signal s_read_color_C  : STD_LOGIC;
signal s_clk_mtx_C     : STD_LOGIC;
signal s_en_mtx_C      : STD_LOGIC;
signal s_med_en_C      : STD_LOGIC;
signal s_dec_en_C      : STD_LOGIC;

signal s_D_out_m_R0 : STD_LOGIC_VECTOR (7 downto 0);
          signal s_D_out_m_R7 : STD_LOGIC_VECTOR (7 downto 0);
          signal s_D_out_m_R15 : STD_LOGIC_VECTOR (7 downto 0);

--decisor

signal s_en_dec    : STD_LOGIC;

signal s_D_in_media_R  : STD_LOGIC_VECTOR(7 downto 0);
signal s_D_in_media_G  : STD_LOGIC_VECTOR(7 downto 0);
signal s_D_in_media_B  : STD_LOGIC_VECTOR(7 downto 0);
```

```
        signal s_D_in_color_R  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_color_G  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_color_B  : STD_LOGIC_VECTOR(7 downto 0);

        signal s_D_out_dec    : STD_LOGIC_VECTOR(1 downto 0);

        --entradas data process


        signal s_gate_in_DP    : STD_LOGIC;
        signal s_clk_in_DP     : STD_LOGIC;

        signal s_data_in_R_DP  : STD_LOGIC_VECTOR (7 downto 0);
        signal s_data_in_G_DP  : STD_LOGIC_VECTOR (7 downto 0);
        signal s_data_in_B_DP  : STD_LOGIC_VECTOR (7 downto 0);



        --salidas data process
        signal s_read_color_DP : STD_LOGIC;
        signal s_data_out_DP   : STD_LOGIC_VECTOR(1 downto 0);

        signal s_dec_en : STD_LOGIC;

begin

  mtx_R_G_B: matrix_R_G_B port map (s_rst_DP,s_en_mtx_C,s_clk_mtx_C,
                                    s_in_mtx_R_DP,s_in_mtx_G_DP,s_in_mtx_B_DP,


s_in_media_R0,s_in_media_R1,s_in_media_R2,s_in_media_R3,s_in_media_R4,

s_in_media_R5,s_in_media_R6,s_in_media_R7,s_in_media_R8,s_in_media_R9,

s_in_media_R10,s_in_media_R11,s_in_media_R12,s_in_media_R13,s_in_media_R14,
                                  s_in_media_R15,


s_in_media_G0,s_in_media_G1,s_in_media_G2,s_in_media_G3,s_in_media_G4,

s_in_media_G5,s_in_media_G6,s_in_media_G7,s_in_media_G8,s_in_media_G9,

s_in_media_G10,s_in_media_G11,s_in_media_G12,s_in_media_G13,s_in_media_G14,
                                    s_in_media_G15,


s_in_media_B0,s_in_media_B1,s_in_media_B2,s_in_media_B3,s_in_media_B4,

s_in_media_B5,s_in_media_B6,s_in_media_B7,s_in_media_B8,s_in_media_B9,

s_in_media_B10,s_in_media_B11,s_in_media_B12,s_in_media_B13,s_in_media_B14,
                                    s_in_media_B15);

  m_R_G_B  : media_R_G_B port map(s_rst_DP, s_med_en_C,


s_in_media_R0,s_in_media_R1,s_in_media_R2,s_in_media_R3,s_in_media_R4,

s_in_media_R5,s_in_media_R6,s_in_media_R7,s_in_media_R8,s_in_media_R9,

s_in_media_R10,s_in_media_R11,s_in_media_R12,s_in_media_R13,s_in_media_R14,
                                    s_in_media_R15,


s_in_media_G0,s_in_media_G1,s_in_media_G2,s_in_media_G3,s_in_media_G4,

s_in_media_G5,s_in_media_G6,s_in_media_G7,s_in_media_G8,s_in_media_G9,

s_in_media_G10,s_in_media_G11,s_in_media_G12,s_in_media_G13,s_in_media_G14,
                                    s_in_media_G15,
```

```
s_in_media_B0,s_in_media_B1,s_in_media_B2,s_in_media_B3,s_in_media_B4,

s_in_media_B5,s_in_media_B6,s_in_media_B7,s_in_media_B8,s_in_media_B9,

s_in_media_B10,s_in_media_B11,s_in_media_B12,s_in_media_B13,s_in_media_B14,
                                        s_in_media_B15,

                                        s_D_in_media_R , s_D_in_media_G, s_D_in_media_B);

  dec: decisor port map(s_rst_DP, s_dec_en_C,

                                    s_in_mtx_R_DP,s_in_mtx_G_DP,s_in_mtx_B_DP,

                                   s_D_in_media_R,s_D_in_media_G,s_D_in_media_B,s_D_out_dec);

  dat_pros_ctrl: data_process_control port map(s_rst_DP, s_clk_in_DP,s_gate_in_DP,s_clk_mtx_C,
                                                        s_en_mtx_C,s_med_en_C,s_dec_en_C);


 --entradas data process
          s_rst_DP <=rst_DP;


          s_gate_in_DP <= gate_in_DP;
          s_clk_in_DP  <= clk_in_DP;

          s_in_mtx_R_DP <= data_in_R_DP;
          s_in_mtx_G_DP <= data_in_G_DP;
          s_in_mtx_B_DP <= data_in_B_DP;
--salidas data process

          D_out_m_R0 <= s_in_media_R0;
          D_out_m_R7 <= s_in_media_R7;
          D_out_m_R15 <= s_in_media_R15;

          D_in_media_R <= s_D_in_media_R;
          D_in_media_G <= s_D_in_media_G;
          D_in_media_B <= s_D_in_media_B;


          data_out_DP <= s_D_out_dec;
end Behavioral;
```

DATA PROCESS CONTROL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity data_process_control is
   Port ( rst_C          : in  STD_LOGIC;
          clk_C          : in  STD_LOGIC;
          gate_C         : in  STD_LOGIC;
          read_color_C   : in  STD_LOGIC;
          clk_mtx_C      : out STD_LOGIC;
          en_mtx_C       : out STD_LOGIC;
          med_en_C       : out STD_LOGIC;
          dec_en_C       : out STD_LOGIC);
```

```
end data_process_control;

architecture Behavioral of data_process_control is

type estados is (S0,S1,S2);

signal estado:estados;


signal s_gate_C  : STD_LOGIC;

signal s_clk_mtx_C    : STD_LOGIC;
signal s_en_mtx_C     : STD_LOGIC;
signal s_med_en_C     : STD_LOGIC;
signal s_dec_en_C     : STD_LOGIC;

signal gate_in_DP_C   : STD_LOGIC;

signal t_read_data : std_logic_vector(7 downto 0):="00000000";



begin


-- entradas

s_gate_C <= gate_C;

--salidas

clk_mtx_C <= s_clk_mtx_C;
en_mtx_C <= s_en_mtx_C;
med_en_C <=  s_med_en_C;
dec_en_C <= s_dec_en_C;

        process(clk_C, rst_C)
        begin
                if (rst_C='1') then
                        estado <= S0;
                        s_clk_mtx_C <='0';
                        s_en_mtx_C <='0';
                        s_med_en_C <='0';
                        s_dec_en_C <='0';


                elsif (clk_C='1' and clk_C'event) then
                case estado is
                        when S0 =>
                                if (s_gate_C='0' ) then

                                        estado <= S1;
                                else

                                        estado <= S0;
                                end if;



                        when S1 =>


                                if ( s_gate_C  ='1') then   -- esperamos a gate='1' (se ha hecho la medida)
                                        s_clk_mtx_C <= '1';
                                        s_en_mtx_C <= '1';
                                        s_med_en_C <='1';
                                        s_dec_en_C <='1';
                                        estado <= S2;
```

```
                                        else

                                                estado <= S1;
                                        end if;
                                when S2 =>


                                        if (t_read_data = "00110010") then  -- tiempo de lectura datos (50 periodos
de reloj (1 us))
                                                s_en_mtx_C <='0';
                                                s_clk_mtx_C <= '0';
                                                s_med_en_C <= '0';
                                                s_dec_en_C <='0';
                                                t_read_data <= "00000000";  --resetamos tiempo de lectura dato
                                                estado <= S0;


                                        else
                                                t_read_data <= t_read_data + 1;
                                                estado <= S2;

                                        end if;


                                end case;
                        end if;
                end process;

end Behavioral;
```

DECISOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity decisor is

   Port ( rst_dec    : in  STD_LOGIC;
          en_dec     : in  STD_LOGIC;

          D_in_media_R  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_G  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_B  : in STD_LOGIC_VECTOR(7 downto 0);

          D_in_color_R  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_color_G  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_color_B  : in STD_LOGIC_VECTOR(7 downto 0);

          D_out_dec    : out STD_LOGIC_VECTOR(1 downto 0));
end decisor;


architecture Behavioral of decisor is

signal s_D_out_dec : STD_LOGIC_VECTOR(1 downto 0);

begin
```

```
D_out_dec <= s_D_out_dec;

process(rst_dec, en_dec)
begin

	if (rst_dec = '1') then

		s_D_out_dec <= "ZZ";

	elsif (en_dec'event and en_dec = '1') then

		if (D_in_color_R > D_in_media_R) then   -- si true => cuadrante superior

			if (D_in_color_G > D_in_media_G) then  -- -- si true cuadrante sup-dech

				s_D_out_dec <=  "00";
			else
				s_D_out_dec <=  "01";          --   cuadrante sup-izq

			end if;

		elsif (D_in_color_R < D_in_media_R) then  -- si true => cuadrante inferior


			if (D_in_color_G > D_in_media_G) then  -- si true cuadrante inf-dech

				s_D_out_dec <=  "10";
			else
				s_D_out_dec <=  "11";         -- cuadrante inf-izq
			end if;
		else
			null;

		end if;


	else
		null;

	end if;

end process;

end Behavioral;
```

DEMUX1_3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY demux1_3 IS
	PORT( data_in_demux : in  STD_LOGIC;
	      sel_demux             : in  STD_LOGIC_VECTOR (1 DOWNTO 0);
	      data_out_demux    : out  STD_LOGIC_VECTOR (2 DOWNTO 0));
END demux1_3;

ARCHITECTURE Behavioral OF demux1_3 IS


BEGIN
```

```
    process(data_in_demux,sel_demux)    -- demultiplexor 1-3
          begin
            case sel_demux is
                  when "01" =>
                        data_out_demux(0) <= data_in_demux ; --salida del red
                        data_out_demux(1) <= 'Z';
                        data_out_demux(2) <= 'Z';
                  when "10" =>
                        data_out_demux (1) <= data_in_demux;  --salida del green
                        data_out_demux(0) <= 'Z';
                        data_out_demux(2) <= 'Z';
                  when "11" =>
                        data_out_demux (2) <= data_in_demux;  --salida del blue
                        data_out_demux(0) <= 'Z';
                        data_out_demux(1) <= 'Z';

                  when others  =>
                        data_out_demux (0) <= 'Z';
                        data_out_demux (1) <= 'Z';
                        data_out_demux (2) <= 'Z';
                  end case;
  end process;

end Behavioral;
```

MATRIX R_G_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity matrix_R_G_B is

   Port ( rst_m    : in  STD_LOGIC;
            en_m : in  STD_LOGIC;
            clk_m     : in STD_LOGIC;

        D_in_m_R     : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_m_G     : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_m_B     : in STD_LOGIC_VECTOR(7 downto 0);

            D_out_m_R0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R10 : out STD_LOGIC_VECTOR(7 downto 0);
```

```
            D_out_m_R11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R12  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R13  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R14 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R15 : out STD_LOGIC_VECTOR(7 downto 0);

            D_out_m_G0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G12  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G13  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G14 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_G15 : out STD_LOGIC_VECTOR(7 downto 0);

            D_out_m_B0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B12  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B13  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B14 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_B15 : out STD_LOGIC_VECTOR(7 downto 0));

end matrix_R_G_B;

architecture Behavioral of matrix_R_G_B is


  component matrix_reg_desp
            Port ( rst_desp    : in  STD_LOGIC;
             en_desp : in  STD_LOGIC;
             clk_desp     : in STD_LOGIC;
        D_in_desp     : in STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_12 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_13 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_14 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_15 : out STD_LOGIC_VECTOR(7 downto 0));

  end component;
```

```
        signal s_rst_m :   STD_LOGIC;
        signal s_en_m :   STD_LOGIC;
        signal s_clk_m :  STD_LOGIC;

    signal s_D_in_m_R : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_m_G : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_m_B : STD_LOGIC_VECTOR(7 downto 0);



begin

  mat_R: matrix_reg_desp port map(s_rst_m, s_en_m,s_clk_m,s_D_in_m_R,
                                  D_out_m_R0,D_out_m_R1,D_out_m_R2,D_out_m_R3,D_out_m_R4,
                                  D_out_m_R5,D_out_m_R6,D_out_m_R7,D_out_m_R8,D_out_m_R9,

D_out_m_R10,D_out_m_R11,D_out_m_R12,D_out_m_R13,D_out_m_R14,D_out_m_R15);

  mat_G: matrix_reg_desp port map(s_rst_m, s_en_m,s_clk_m,s_D_in_m_G,
                                  D_out_m_G0,D_out_m_G1,D_out_m_G2,D_out_m_G3,D_out_m_G4,
                                  D_out_m_G5,D_out_m_G6,D_out_m_G7,D_out_m_G8,D_out_m_G9,

D_out_m_G10,D_out_m_G11,D_out_m_G12,D_out_m_G13,D_out_m_G14,D_out_m_G15);


  mat_B: matrix_reg_desp port map(s_rst_m, s_en_m,s_clk_m,s_D_in_m_B,
                                  D_out_m_B0,D_out_m_B1,D_out_m_B2,D_out_m_B3,D_out_m_B4,
                                  D_out_m_B5,D_out_m_B6,D_out_m_B7,D_out_m_B8,D_out_m_B9,

D_out_m_B10,D_out_m_B11,D_out_m_B12,D_out_m_B13,D_out_m_B14,D_out_m_B15);



 --entradas

  s_rst_m <= rst_m;
  s_en_m <= en_m;
  s_clk_m <= clk_m;

  s_D_in_m_R <= D_in_m_R;
  s_D_in_m_G <= D_in_m_G;
  s_D_in_m_B <= D_in_m_B;


end Behavioral;
```

MATRIX_REG_DESP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity matrix_reg_desp is

   Port ( rst_desp    : in  STD_LOGIC;
          en_desp : in  STD_LOGIC;
          clk_desp     : in STD_LOGIC;
```

```
            D_in_desp    : in STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_1  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_2  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_3  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_4  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_5  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_6  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_8  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_9  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_10 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_11 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_12 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_13 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_14 : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_desp_15 : out STD_LOGIC_VECTOR(7 downto 0));

end matrix_reg_desp;


architecture Behavioral of matrix_reg_desp is

component reg_8_ent_8_sal

    Port ( rst   : in  STD_LOGIC;
           en    : in  STD_LOGIC;
           clk   : in  STD_LOGIC;
           D_in  : in STD_LOGIC_VECTOR(7 DOWNTO 0);
           D_out : out STD_LOGIC_VECTOR(7 DOWNTO 0));

end component;


signal s_reg_out_0 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_1 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_2 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_3 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_4 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_5 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_6 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_7 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_8 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_9 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_10 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_11 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_12 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_13 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_14 : STD_LOGIC_VECTOR(7 downto 0);
signal s_reg_out_15 : STD_LOGIC_VECTOR(7 downto 0);

begin


D_out_desp_0 <= s_reg_out_0;
D_out_desp_1 <= s_reg_out_1;
D_out_desp_2 <= s_reg_out_2;
D_out_desp_3 <= s_reg_out_3;
D_out_desp_4 <= s_reg_out_4;
D_out_desp_5 <= s_reg_out_5;
D_out_desp_6 <= s_reg_out_6;
D_out_desp_7 <= s_reg_out_7;
D_out_desp_8 <= s_reg_out_8;
D_out_desp_9 <= s_reg_out_9;
D_out_desp_10 <= s_reg_out_10;
D_out_desp_11 <= s_reg_out_11;
D_out_desp_12 <= s_reg_out_12;
D_out_desp_13 <= s_reg_out_13;
D_out_desp_14 <= s_reg_out_14;
D_out_desp_15 <= s_reg_out_15;
```

```
r_0: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,D_in_desp,s_reg_out_0);
r_1: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_0,s_reg_out_1);
r_2: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_1,s_reg_out_2);
r_3: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_2,s_reg_out_3);
r_4: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_3,s_reg_out_4);
r_5: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_4,s_reg_out_5);
r_6: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_5,s_reg_out_6);
r_7: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_6,s_reg_out_7);
r_8: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_7,s_reg_out_8);
r_9: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_8,s_reg_out_9);
r_10: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_9,s_reg_out_10);
r_11: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_10,s_reg_out_11);
r_12: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_11,s_reg_out_12);
r_13: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_12,s_reg_out_13);
r_14: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_13,s_reg_out_14);
r_15: reg_8_ent_8_sal port map (rst_desp,en_desp,clk_desp,s_reg_out_14,s_reg_out_15);

end Behavioral;
```

MEDIA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity media is

   Port ( rst_media    : in  STD_LOGIC;
          en_media     : in  STD_LOGIC;

          D_in_media_0  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_1  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_2  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_3  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_4  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_5  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_6  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_7  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_8  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_9  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_10 : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_11 : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_12  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_13  : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_14 : in STD_LOGIC_VECTOR(7 downto 0);
          D_in_media_15 : in STD_LOGIC_VECTOR(7 downto 0);

          D_out_media    : out STD_LOGIC_VECTOR(7 downto 0));
end media;


architecture Behavioral of media is


signal s_D_out_media : STD_LOGIC_VECTOR(7 downto 0):="00000000";

begin
```

```
D_out_media <= s_D_out_media;

process(rst_media, en_media)
begin

        if (rst_media = '1') then

                s_D_out_media <= "ZZZZZZZZ";

        elsif (en_media'event and en_media = '1') then

                s_D_out_media<=conv_std_logic_vector((conv_integer(D_in_media_0) +conv_integer(
D_in_media_1)+
                                                        conv_integer(D_in_media_2) +
                                                        conv_integer(D_in_media_3) +
conv_integer(D_in_media_4) +
                                                        conv_integer(D_in_media_5) +
conv_integer(D_in_media_6) +
                                                        conv_integer(D_in_media_7) +
conv_integer(D_in_media_8) +
                                                        conv_integer(D_in_media_9) +
conv_integer(D_in_media_10) +
                                                        conv_integer(D_in_media_11) +
conv_integer(D_in_media_12) +
                                                        conv_integer(D_in_media_13) +
conv_integer(D_in_media_14) +
                                                        conv_integer(D_in_media_15))/16,8);


        end if;

end process;


end Behavioral;
```

MEDIA R_G_B

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity media_R_G_B is

   Port ( rst_m_R_G_B : in  STD_LOGIC;
          en_m_R_G_B  : in  STD_LOGIC;

          D_in_m_R0    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R1    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R2    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R3    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R4    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R5    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R6    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R7    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R8    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R9    : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R10   : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R11   : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R12   : in STD_LOGIC_VECTOR (7 downto 0);
          D_in_m_R13    : in STD_LOGIC_VECTOR (7 downto 0);
```

```
            D_in_m_R14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_R15   : in STD_LOGIC_VECTOR (7 downto 0);


            D_in_m_G0    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G1    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G2    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G3    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G4    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G5    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G6    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G7    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G8    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G9    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G10   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G11   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G12    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G13    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_G15   : in STD_LOGIC_VECTOR (7 downto 0);

            D_in_m_B0    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B1    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B2    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B3    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B4    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B5    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B6    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B7    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B8    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B9    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B10   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B11   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B12    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B13    : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B14   : in STD_LOGIC_VECTOR (7 downto 0);
            D_in_m_B15   : in STD_LOGIC_VECTOR (7 downto 0);



            D_out_m_R   : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_G   : out STD_LOGIC_VECTOR (7 downto 0);
            D_out_m_B   : out STD_LOGIC_VECTOR (7 downto 0));

end media_R_G_B;

architecture Behavioral of media_R_G_B is


  component media
          Port ( rst_media    : in  STD_LOGIC;
            en_media      : in  STD_LOGIC;

            D_in_media_0  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_1  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_2  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_3  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_4  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_5  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_6  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_7  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_8  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_9  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_10 : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_11 : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_12  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_13  : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_14 : in STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_15 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
            D_out_media    : out STD_LOGIC_VECTOR(7 downto 0));

  end component;


          --media
          signal s_rst_m :   STD_LOGIC;
          signal s_en_m :   STD_LOGIC;




begin


  media_R: media port map(s_rst_m, s_en_m,
                                D_in_m_R0,D_in_m_R1,D_in_m_R2,D_in_m_R3,D_in_m_R4,
                            D_in_m_R5,D_in_m_R6,D_in_m_R7,D_in_m_R8,D_in_m_R9,D_in_m_R10,D_in_m_R11,
                                D_in_m_R12,D_in_m_R13,D_in_m_R14,D_in_m_R15,
                                D_out_m_R);
  media_G: media port map(s_rst_m, s_en_m,
                                D_in_m_G0,D_in_m_G1,D_in_m_G2,D_in_m_G3,D_in_m_G4,

D_in_m_G5,D_in_m_G6,D_in_m_G7,D_in_m_G8,D_in_m_G9,D_in_m_G10,D_in_m_G11,
                                D_in_m_G12,D_in_m_G13,D_in_m_G14,D_in_m_G15,
                                D_out_m_G);
  media_B: media port map(s_rst_m, s_en_m,
                                D_in_m_B0,D_in_m_B1,D_in_m_B2,D_in_m_B3,D_in_m_B4,

D_in_m_B5,D_in_m_B6,D_in_m_B7,D_in_m_B8,D_in_m_B9,D_in_m_B10,D_in_m_B11,
                                D_in_m_B12,D_in_m_B13,D_in_m_B14,D_in_m_B15,
                                D_out_m_B);


-- entradas re_desp
  s_rst_m <= rst_m_R_G_B;
  s_en_m <= en_m_R_G_B;

end Behavioral;
```

REG_DESP

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity re_desp is

   Port ( rst_desp    : in  STD_LOGIC;
          en_desp : in  STD_LOGIC;
          clk_desp     : in STD_LOGIC;
          D_in_desp     : in STD_LOGIC;
          D_out_desp    : out STD_LOGIC_VECTOR(15 downto 0));
end re_desp;
```

```
architecture Behavioral of re_desp is

component registro
            Port ( rst    : in  STD_LOGIC;
              en : in  STD_LOGIC;
              clk      : in  STD_LOGIC;
              D_in     : in STD_LOGIC;
              D_out    : out STD_LOGIC);
end component;



signal s_Q0 : STD_LOGIC;
signal s_Q1 : STD_LOGIC;
signal s_Q2: STD_LOGIC;
signal s_Q3 : STD_LOGIC;
signal s_Q4 : STD_LOGIC;
signal s_Q5: STD_LOGIC;
signal s_Q6: STD_LOGIC;
signal s_Q7: STD_LOGIC;
signal s_Q8: STD_LOGIC;
signal s_Q9: STD_LOGIC;
signal s_Q10: STD_LOGIC;
signal s_Q11: STD_LOGIC;
signal s_Q12: STD_LOGIC;
signal s_Q13: STD_LOGIC;
signal s_Q14: STD_LOGIC;
signal s_Q15: STD_LOGIC;


begin

--salidas

D_out_desp(0) <= s_Q15;
D_out_desp(1) <= s_Q14;
D_out_desp(2) <= s_Q13;
D_out_desp(3) <= s_Q12;
D_out_desp(4) <= s_Q11;
D_out_desp(5) <= s_Q10;
D_out_desp(6) <= s_Q9;
D_out_desp(7) <= s_Q8;
D_out_desp(8) <= s_Q7;
D_out_desp(9) <= s_Q6;
D_out_desp(10) <= s_Q5;
D_out_desp(11) <= s_Q4;
D_out_desp(12) <= s_Q3;
D_out_desp(13) <= s_Q2;
D_out_desp(14) <= s_Q1;
D_out_desp(15) <= s_Q0;



r_0: registro port map (rst_desp,en_desp,clk_desp,D_in_desp,s_Q0);
r_1: registro port map (rst_desp,en_desp,clk_desp,s_Q0,s_Q1);
r_2: registro port map (rst_desp,en_desp,clk_desp,s_Q1,s_Q2);
r_3: registro port map (rst_desp,en_desp,clk_desp,s_Q2,s_Q3);
r_4: registro port map (rst_desp,en_desp,clk_desp,s_Q3,s_Q4);
r_5: registro port map (rst_desp,en_desp,clk_desp,s_Q4,s_Q5);
r_6: registro port map (rst_desp,en_desp,clk_desp,s_Q5,s_Q6);
r_7: registro port map (rst_desp,en_desp,clk_desp,s_Q6,s_Q7);
r_8: registro port map (rst_desp,en_desp,clk_desp,s_Q7,s_Q8);
r_9: registro port map (rst_desp,en_desp,clk_desp,s_Q8,s_Q9);
r_10: registro port map (rst_desp,en_desp,clk_desp,s_Q9,s_Q10);
r_11: registro port map (rst_desp,en_desp,clk_desp,s_Q10,s_Q11);
r_12: registro port map (rst_desp,en_desp,clk_desp,s_Q11,s_Q12);
r_13: registro port map (rst_desp,en_desp,clk_desp,s_Q12,s_Q13);
r_14: registro port map (rst_desp,en_desp,clk_desp,s_Q13,s_Q14);
```

```
r_15: registro port map (rst_desp,en_desp,clk_desp,s_Q14,s_Q15);

end Behavioral;
```

RECEPTOR VLC

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity receptor_vlc is

   Port ( rst      : in  STD_LOGIC;
          clk      : in  STD_LOGIC;
          data_in  : in  STD_LOGIC;

         --SEÑAL DE ENTRADA DE SINCRONISMO

          sinc_in  : in  STD_LOGIC;

                --SEÑALES DE SALIDA DE DEMUX
          data_out_demux  : out  STD_LOGIC_VECTOR (2 DOWNTO 0);

                --SEÑALES DE ENABLE REG DESP COLOR READER
          R_en_demux_CR     : out STD_LOGIC;
          G_en_demux_CR     : out STD_LOGIC;
          B_en_demux_CR     : out STD_LOGIC;

                --DATOS SALIDA READ COLOR
          data_out_R_CR : out STD_LOGIC_VECTOR (7 downto 0);
          data_out_G_CR : out STD_LOGIC_VECTOR (7 downto 0);
          data_out_B_CR : out STD_LOGIC_VECTOR (7 downto 0);

                --SALIDAS DE MATRIX R G B
          D_out_m_R0  : out STD_LOGIC_VECTOR(7 downto 0);
          D_out_m_R7  : out STD_LOGIC_VECTOR(7 downto 0);
          D_out_m_R15  : out STD_LOGIC_VECTOR(7 downto 0);

                --DATOS SALIDA MEDIA
          D_out_m_R   : out STD_LOGIC_VECTOR (7 downto 0);
          D_out_m_G   : out STD_LOGIC_VECTOR (7 downto 0);
          D_out_m_B   : out STD_LOGIC_VECTOR (7 downto 0);

         sel_out   : out STD_LOGIC_VECTOR(1 DOWNTO 0);


         gate     : out STD_LOGIC;
         ck       : out  STD_LOGIC;

          data_out : out  STD_LOGIC_VECTOR(1 downto 0));

end receptor_vlc;

architecture Behavioral of receptor_vlc is


  component data_process
            Port ( rst_DP       : in  STD_LOGIC;
```

```
            gate_in_DP    : in  STD_LOGIC;
            clk_in_DP     : in  STD_LOGIC;



                  --salidas DE COLOR READER R G B
            data_in_R_DP  : in  STD_LOGIC_VECTOR (7 downto 0);
            data_in_G_DP  : in  STD_LOGIC_VECTOR (7 downto 0);
            data_in_B_DP  : in  STD_LOGIC_VECTOR (7 downto 0);

                  --SALIDAS DE MATRIX R G B
            D_out_m_R0  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R7  : out STD_LOGIC_VECTOR(7 downto 0);
            D_out_m_R15  : out STD_LOGIC_VECTOR(7 downto 0);

                  --SALIDAS DE MEDIA R G B
        D_in_media_R  : OUT STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_G  : OUT STD_LOGIC_VECTOR(7 downto 0);
            D_in_media_B  : OUT STD_LOGIC_VECTOR(7 downto 0);


            data_out_DP   : out  STD_LOGIC_VECTOR(1 downto 0));

  end component;

  component color_reader
          Port ( rst_CR       : in  STD_LOGIC;
            sinc_in    : in  STD_LOGIC;
            data_in_CR    : in  STD_LOGIC;
            clk_CR       : in  STD_LOGIC;



          --SEÑALES DE SALIDA DE DEMUX
            data_out_demux  : out  STD_LOGIC_VECTOR (2 DOWNTO 0);

          --SEÑALES DE ENABLE REG DESP COLOR READER
            R_en_demux_CR     : out STD_LOGIC;
            G_en_demux_CR     : out STD_LOGIC;
            B_en_demux_CR     : out STD_LOGIC;

            sel_out_CR   : out STD_LOGIC_VECTOR(1 downto 0);


            gate_CR      : out  STD_LOGIC;
            ck_CR        : out  STD_LOGIC;

            data_out_R_CR : out STD_LOGIC_VECTOR (7 downto 0);
            data_out_G_CR : out STD_LOGIC_VECTOR (7 downto 0);
            data_out_B_CR : out STD_LOGIC_VECTOR (7 downto 0));

end component;


component clkscale
          Port ( clk : in  STD_LOGIC;
           divsel : in  STD_LOGIC_VECTOR (2 downto 0);
           rst : in  STD_LOGIC;
          clk_out : out  STD_LOGIC);
end component;

          --señale clkscale
          signal s_clk_out    : STD_LOGIC;   --señal reloj



          signal s_divsel :  STD_LOGIC_VECTOR (2 downto 0):="011";
          signal s_hold : STD_LOGIC;
          signal s_debon_clk_out :  STD_LOGIC;
```

```
        signal s_clksacale_out :  STD_LOGIC;
        signal s_out_elim_reb :  STD_LOGIC;

    --señales de entrada receptor vlc

        signal s_rst    : STD_LOGIC;      --reset
        signal s_clk    : STD_LOGIC;   --señal reloj

        signal s_data_in : STD_LOGIC;   --señal de entrada datos

        --señales de salida receptor vlc

        signal s_gate  : STD_LOGIC;
        signal s_ck    : STD_LOGIC;

        signal s_data_out  : STD_LOGIC_VECTOR(1 downto 0);   -- salida de datos receptor



        --señales de entrada color_ reade


        signal s_read_color : STD_LOGIC;

        -- SALIDAS COLOR READER
        signal s_data_out_demux  : STD_LOGIC_VECTOR (2 DOWNTO 0);


                --SEÃALES DE ENABLE RG DESP COLOR READER
        signal s_R_en_demux_CR    : STD_LOGIC;
        signal s_G_en_demux_CR    : STD_LOGIC;
        signal s_B_en_demux_CR    : STD_LOGIC;

        signal s_sel_out_CR   :  STD_LOGIC_VECTOR(1 downto 0);
        signal s_data_R  : STD_LOGIC_VECTOR (7 downto 0);   --salida datos RGB
        signal s_data_G  : STD_LOGIC_VECTOR (7 downto 0);
        signal s_data_B  : STD_LOGIC_VECTOR (7 downto 0);

                --SALIDAS DE MATRIX R G B
        signal s_D_out_m_R0  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_out_m_R7  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_out_m_R15  : STD_LOGIC_VECTOR(7 downto 0);

                --SALIDAS DE MEDIA R G B
        signal s_D_in_media_R  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_media_G  : STD_LOGIC_VECTOR(7 downto 0);
        signal s_D_in_media_B  : STD_LOGIC_VECTOR(7 downto 0);

        --señal de sinc
        signal s_sinc_in  : STD_LOGIC;
        signal s_sinc_in1  : STD_LOGIC;
        begin

  col_reader: color_reader port map (s_rst,s_sinc_in,s_data_in,s_clk_out,s_data_out_demux,
                                                s_R_en_demux_CR,s_G_en_demux_CR,s_B_en_demux_CR,
                                                s_sel_out_CR,s_gate,s_ck,
                                                s_data_R,s_data_G,s_data_B);

  dat_pros: data_process port map(s_rst,s_gate,s_clk_out,

                                        s_data_R,s_data_G,s_data_B,
                                        s_D_out_m_R0,s_D_out_m_R7,s_D_out_m_R15,
                                        s_D_in_media_R,s_D_in_media_G,s_D_in_media_B,
                                        --read_color_C,
                                                s_data_out);

clksc : clkscale port map (s_clk,s_divsel,s_rst, s_clk_out);
```

```
 --entradas receptor vIC

            s_rst  <= rst;
            s_clk  <= clk;
            s_data_in <= data_in;


            --SALIDA DEMUX
            data_out_demux <= s_data_out_demux;

            --SEÑALES DE ENABLE REG DESP COLOR READER
              R_en_demux_CR <= s_R_en_demux_CR;
              G_en_demux_CR <= s_G_en_demux_CR;
              B_en_demux_CR <= s_B_en_demux_CR;


            -- SALIDAS COLOR READER
            data_out_R_CR <= s_data_R  ;
            data_out_G_CR <= s_data_G  ;
            data_out_B_CR <= s_data_B ;

            --SALIDAS DE MATRIX R G B
            D_out_m_R0 <= s_D_out_m_R0;
            D_out_m_R7 <= s_D_out_m_R7;
            D_out_m_R15 <= s_D_out_m_R15;


            --SALIDAS DE MEDIA R G B
            D_out_m_R  <=  s_D_in_media_R;
            D_out_m_G  <=  s_D_in_media_G;
            D_out_m_B  <=  s_D_in_media_B;

             sel_out <= s_sel_out_CR;

            s_sinc_in <= sinc_in;
            gate     <= s_gate;
            ck       <= s_ck;
            data_out <= s_data_out;


end Behavioral;
```

REGISTRO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity registro is
   Port ( rst   : in  STD_LOGIC;
               en      : in  STD_LOGIC;
               clk     : in  STD_LOGIC;
               D_in    : in STD_LOGIC;
               D_out   : out STD_LOGIC);
end registro;


architecture Behavioral of registro is
```

```
begin

        process(rst,clk,en)
        begin


          IF (clk='1' and clk'event  and en='1') then
                D_out <= D_in;
          end if;

        end process;

end Behavioral;
```

REG 8 ENT 8 SAL

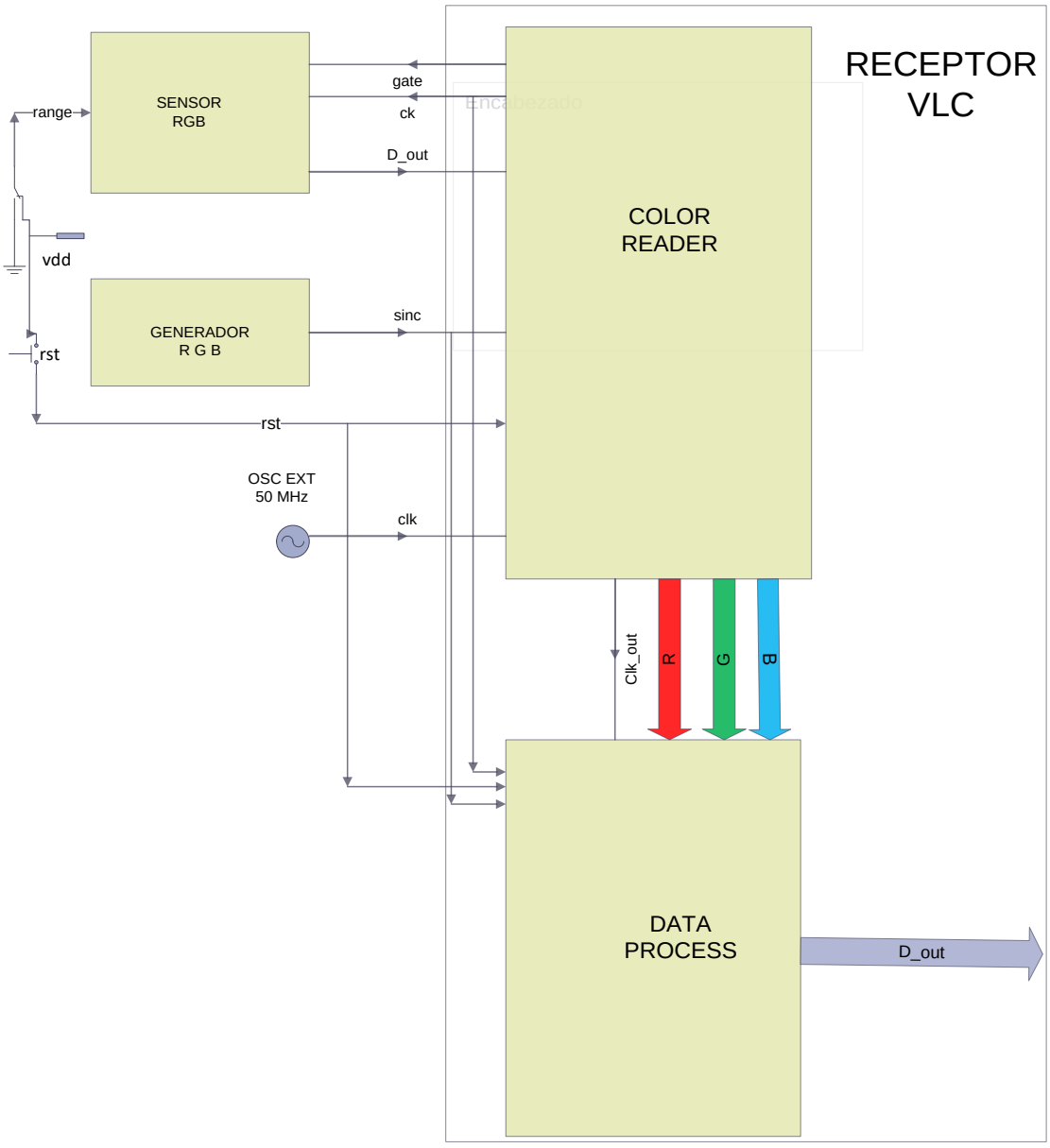
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity reg_8_ent_8_sal is
   Port ( rst   : in  STD_LOGIC;
          en      : in  STD_LOGIC;
          clk     : in  STD_LOGIC;
          D_in    : in STD_LOGIC_VECTOR(7 DOWNTO 0);
          D_out   : out STD_LOGIC_VECTOR(7 DOWNTO 0));
end reg_8_ent_8_sal;


architecture Behavioral of reg_8_ent_8_sal is


begin

        process(rst,clk,en)
        begin


          IF (clk='1' and clk'event  and en='1') then
                D_out <= D_in;
          end if;

        end process;

end Behavioral;
```

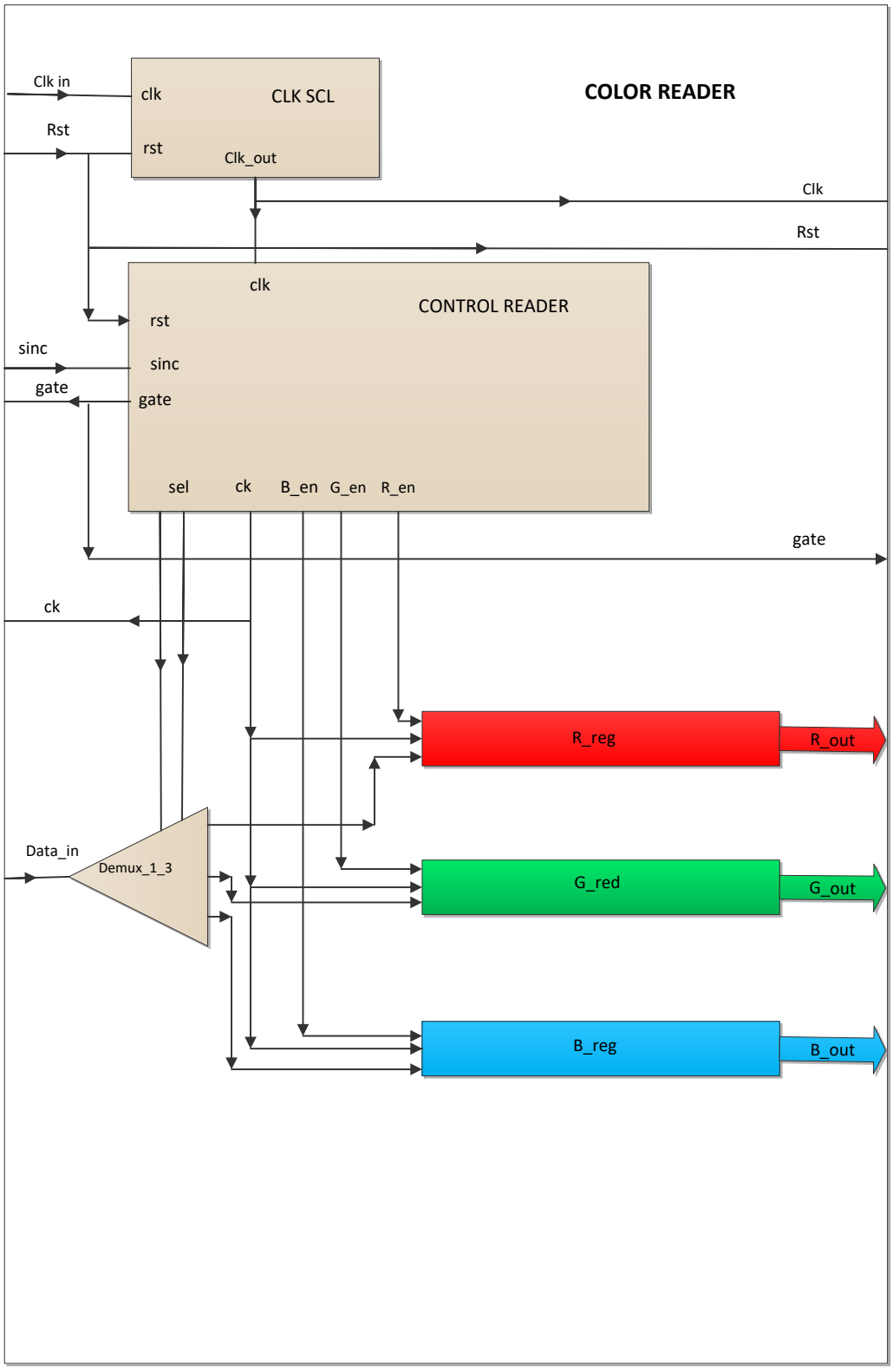
Anexo IV

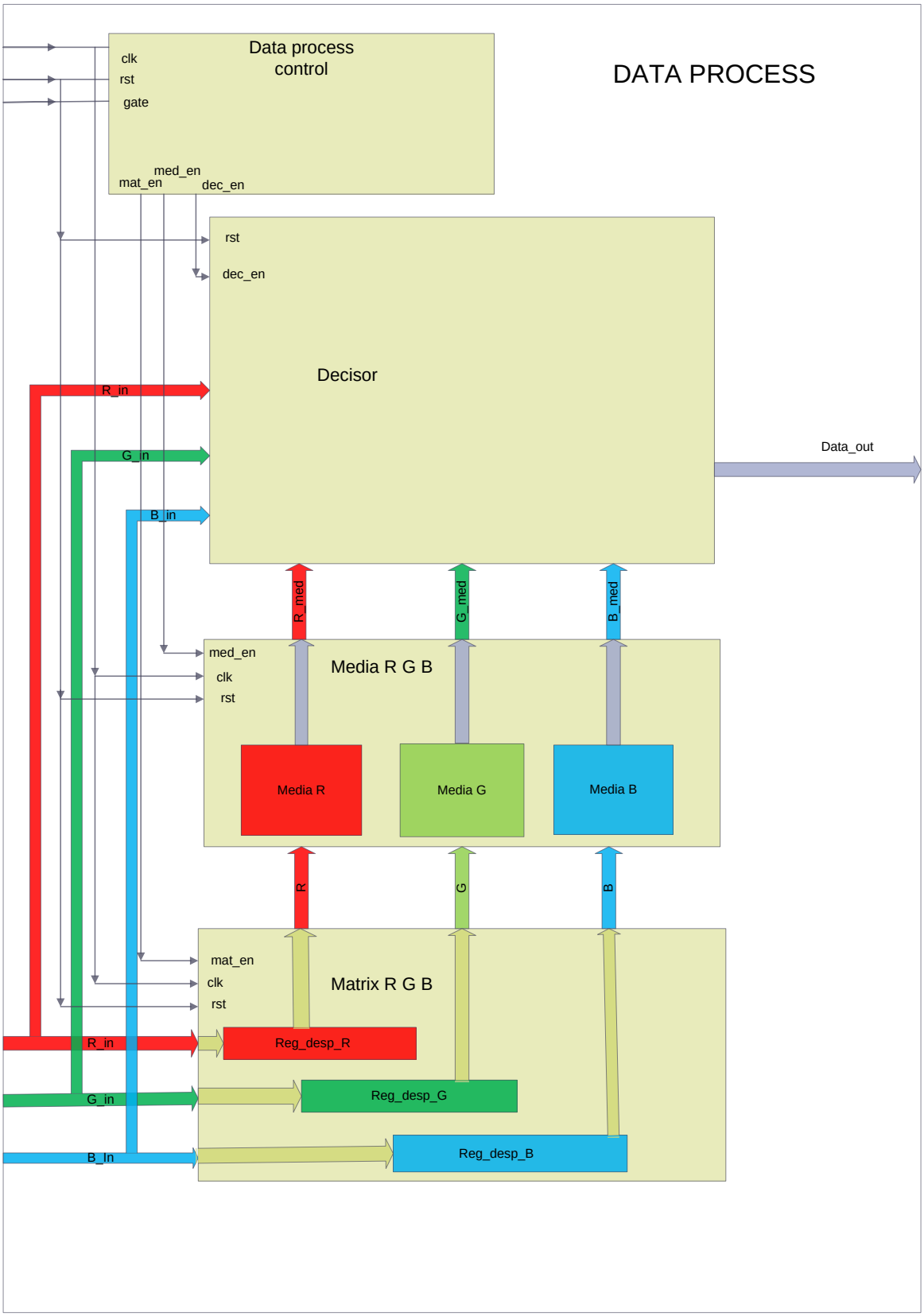
DIAGRAMAS DE BLOQUE

ANEXO IV

Diagramas de bloques







Anexo V

ESQUEMA ELÉCTRICO

ANEXO V

Esquema eléctrico