Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO DE FIN DE GRADO:

Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Alumnos: David Dorta Pimentel
Sergio Hernández Romera

Tutores: Oswaldo B. González Hernández
José Carlos San Luis Leal

Julio, 2017

Agradecimientos

Queremos agradecer todo el apoyo durante todos estos años por darnos la oportunidad de estudiar, además de mostrarnos cada día el camino a seguir a nuestra familia. En especial a nuestros padres, hermanos, cuñados y sobrinos que siempre están para lo bueno como para lo malo.

A todos aquellos profesores que nos han aportado los conocimientos suficientes para desarrollar el trabajo final de grado. Con especial dedicación a nuestros tutores Oswaldo Bernabé González Hernández y José Carlos San Luis Leal, esenciales en la consecución del trabajo. Además, a Fernando Luis Rosa González por todos sus consejos en el laboratorio.

A nuestros amigos y compañeros por haber compartido tantos momentos juntos, por toda la ayuda y cariño recibido por su parte, especialmente a Elena, Iván, Carlos y Julio.

Contenido

Índice de Figuras

Resumen del trabajo

El trabajo consiste en realizar el diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED. Para el correcto desarrollo, se deben perfeccionar los distintos elementos que componen el canal de transmisión (emisor y receptor).

En primer lugar, realizamos el tratamiento de señales en la FPGA, diseñando los bloques en VHDL (*Very high-speed integrated circuit Hardware Description Language*) asociados al emisor y al receptor. Estos bloques tienen la finalidad de mejorar la señal de nuestro medio de comunicación (tratándola a través de filtros digitales, derivadores, retardadores…) consiguiendo que la recepción del mensaje sea el esperado, además permite controlar el mensaje a reproducir, adquiriendo un amplio abanico de posibilidades de transmisión.

Para poder visualizar el comportamiento de los elementos lógicos creados debemos utilizar un convertidor digital a analógico (CAD). Para ello, programamos el funcionamiento del CAD contenido en nuestra FPGA.

Una vez comprobado toda la lógica digital, pasamos a la fabricación de los elementos físicos del enlace óptico. Estos dispositivos PCB *(Printed Circuit Board*), serán los encargados de emitir y recepcionar el haz de luz que contiene la información del mensaje, por lo que crearemos una placa emisora (que contendrá el LED como medio de transmisión, el cual reproducirá el mensaje definido) y una placa receptora, que por medio de un foto-diodo captará las variaciones de intensidades provenientes del LED y las transformará a una señal analógica a su salida.

Para lograr que todos los elementos del prototipo se puedan enlazar, ya que los bloques implementados en la FPGA trabajan con señales digitales y la salida del enlace óptico físico es una señal analógica. Fabricamos una placa que contendrá un ADC (convertidor analógico a digital), que será el nexo entre la placa receptora y el procesado final de la señal a través de la FPGA.

Por último, se realiza el montaje completo de nuestro sistema de comunicación. Primeramente, la PCB emisora se conectará a la FPGA, que reproducirá el mensaje definido en la lógica digital. Este mensaje lo recogerá el foto-diodo asociado a la placa receptora, que a su salida representará la señal analógica del mensaje. Tras ello, el ADC convertirá dicha señal

y la enlazará de nuevo a la FPGA, que realizará el tratamiento final de la misma. Una vez tratado vemos el resultado de la comunicación a través de un conversor digital a analógico (CAD).

Abstract

The aim of this project is to make a communication channel with light-emitting diode (LED) lamps through a FPGA board. To get this we must implement the elements that compose the transmission channel (emitter and receiver).

Firstly, we design the VHDL modules. These modules will process the signals that we want to transmit. The signal processing will improve the emission and the reception of the message, preventing from signal failures and noise. Also, this treatment allows us to control many characteristics of the signal, which helps to reproduce it correctly.

While we are processing the signals, we have to visualize the different states of them, because of that we use a digital-to-analog converter contained in the FPGA (Field-Programmable Gate Array) board, where all the logic circuits are implemented.

Secondly, we create the physical elements of the optical link in two independent printed circuit boards (PCBs). These devices will emit and receive the beam of light which contains the message information. The first element of the link is the emitter. It will have a LED diode that will modify the intensity of the light according to the message. The second element is the receiver based on a photodiode which is able to convert the light variations into an electrical current.

After ensuring the link is working properly, we join together all the components we described before. In order to do that we must create a PCB with analog-to-digital converter (ADC) that will prepare the signal to be ready for the final processing by the FPGA.

Finally, we set up the final assembly. The emitter PCB will be connected to the FPGA, it will send the message created in the VHDL code. The sent message will be gathered up by the photodiode generating an analog signal at the output of the receiver board. The ADC will convert this signal to a digital one and transmit it again to the FPGA, which will make the final processing of it. After that we can send the message to another device or visualize it by using another DAC and an oscilloscope.

Capítulo I:

Introducción

1. Capítulo I. Introducción.

1.1. Introducción general

Desde el principio de los tiempos el ser humano ha necesitado comunicarse con sus semejantes y para ello ha hecho uso de diversos sistemas de comunicación que le permitieran transmitir la información. Dichos sistemas han tomado diferentes formas a lo largo de la historia, desde el papel en la comunicación escrita hasta las actuales redes que permiten la comunicación audiovisual. En las últimas décadas estos sistemas han evolucionado de forma exponencial ayudando, en gran parte, al fenómeno de la globalización.

En la actualidad, el ser humano cuenta con distintos y avanzados métodos para comunicarse, como por ejemplo las comunicaciones a través de cables en las cuales destaca la fibra óptica, que permite un efectivo transporte de información a grandes distancias, pero con grandes costes y la dificultad de su instalación. En el caso de las comunicaciones inalámbricas se debe nombrar la telefonía móvil y su actual tecnología 4G, la cual permite alcanzar elevadas velocidades de transmisión de hasta 100 Mbps.

Pero hay una tecnología en desarrollo que, pese a que no ha podido aplicarse de forma práctica, se le augura un futuro brillante. Esta es la tecnología VLC (*Visible Light Communication* o comunicación por luz visible, por sus siglas en inglés) que se podría aplicar en las fuentes lumínicas convencionales para que estas, aparte de servir para la iluminación del lugar, pudieran ser utilizadas como canales de comunicación inalámbricos de gran ancho de banda.

La tecnología VLC se basa en la modulación de la intensidad lumínica. Dicha modulación se basa en la variación de la intensidad emitida según los datos que se deseen transmitir. Estos cambios se detectan a través de un foto-detector que transforma dicha información en una señal eléctrica de la cual se extraerán los datos posteriormente. Las altas velocidades de transmisión alcanzadas por esta tecnología en el laboratorio la sitúan por delante de la gran mayoría de las tecnologías de comunicación inalámbrica de la actualidad, siendo, por ejemplo, cien veces más rápida que el Wi-Fi.

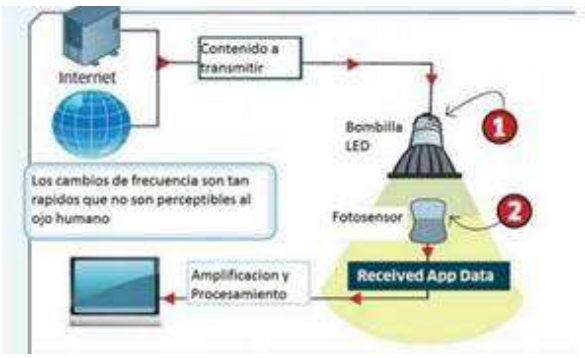


Figura 1.1. Tecnología VLC [1].

Si hay algo que está claro respecto a esta tecnología es que permite una gran cantidad de aplicaciones entre las que cabe destacar el Li-Fi, que en un futuro próximo permitirá la conexión a Internet con velocidades de hasta 1 Gbps "mientras se ilumina el entorno con iluminancias que cumplen con los estándares ISO" tal y como afirman varios investigadores coreanos [IETE Journal of Research, 2013].

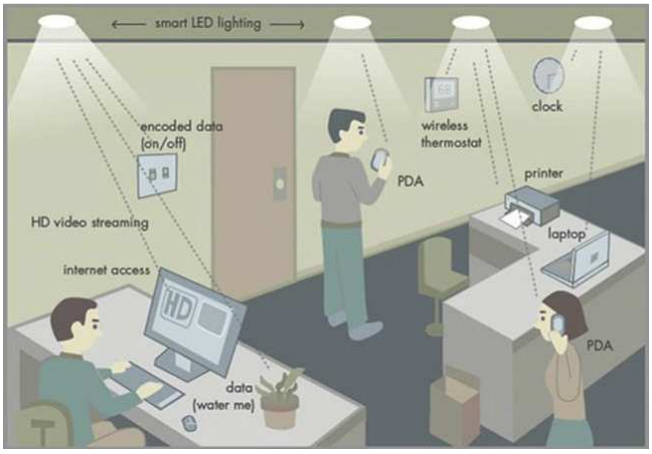


Figura 1.2. Tecnología Li-Fi [Boston University 16].

1.2. Objetivos del proyecto. Descripción general del sistema.

La figura 3 muestra el diagrama de bloques del sistema creado. Dicho sistema se encuentra controlado por la FPGA (*Field Programmable Gate Array*) que se encuentra en una tarjeta SPARTAN 3AN FPGA *Starter Kit Board*, la cual realizará el pre-procesado de la señal que se emitirá a través del circuito emisor y también el procesado de la señal recibida por el circuito receptor. En otras palabras, la FPGA realizará las distintas técnicas de procesado digital de las señales que se transmiten a través del enlace óptico.

Con este prototipo se pretende crear un canal de comunicación inalámbrica que permita la transmisión de información a partir de la modulación de la intensidad lumínica del haz generado por un diodo LED (*Light-Emitting Diode*), su correcta recepción en un fotodiodo y el procesado de la señal eléctrica generada por el mismo.

La modulación que se ha utilizado es la conocida como PAM (*Pulse-Amplitude Modulation* o modulación por amplitud de pulsos) en la cual el desfase y la frecuencia quedan fijas y la amplitud es la que varía. Se ha utilizado esta modulación ya que diversas investigaciones han demostrado que gracias a ella se reduce la secuencia de entrenamiento, reduciendo la limitación que supone la baja velocidad de conmutación del LED [IETE Journal of Research, 2013].

La conexión entre los distintos bloques se realiza a través de los distintos módulos que se pueden encontrar en la tarjeta, además de la utilización de un convertidor analógico-digital (ADC, *Analog-to-Digital Converter*) para conversión de la señal analógica generada en el fotodiodo a una señal digital que pueda leer de forma correcta la tarjeta.

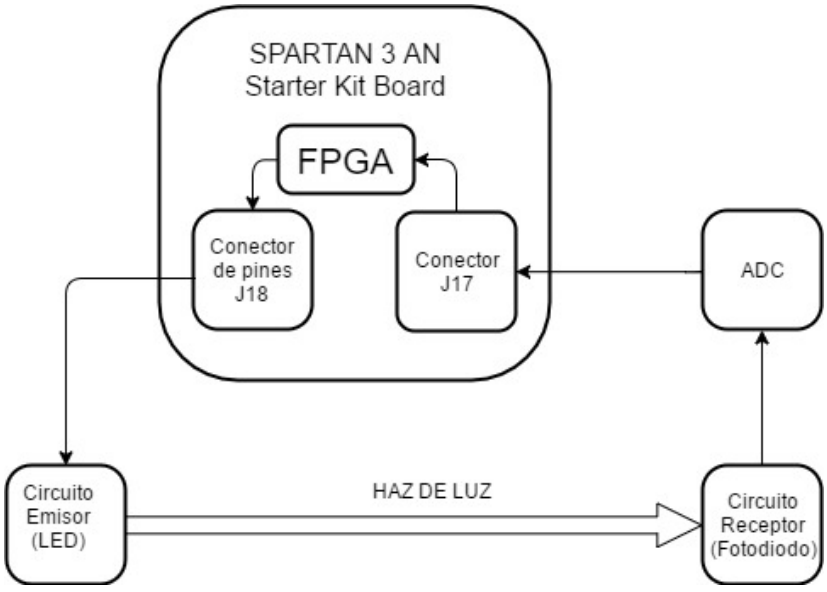


Figura 1.3. Diagrama de bloques del proyecto.

1.3. Estructura general del trabajo.

La memoria está dividida a lo largo de seis capítulos en los cuales se exponen todos los aspectos relacionados con el proyecto. De esta manera se ha seguido la siguiente distribución:

Tras esta introducción que da una visión global del proyecto e indica la finalidad del mismo, en el capítulo II se realiza una breve descripción de los distintos recursos utilizados durante este trabajo. Dicha explicación se realizará tanto de las diversas herramientas *software* como de los elementos *hardware* presentes en el prototipo.

El Capítulo III se centra en comentar los distintos módulos de programación necesarios para realizar el procesado de la señal de forma esquematizada, así como también presentar las simulaciones realizadas sobre los mismos.

A lo largo del Capítulo IV se hace acopio de los módulos de programación explicados en el capítulo anterior componiendo los distintos bloques emisor y receptor. Además de esto, se explican los circuitos diseñados para crear el enlace óptico y su conexión con la tarjeta y, por último, se explica de forma detallada y definitiva el funcionamiento del proyecto.

Los resultados experimentales, verificaciones y demostraciones del funcionamiento del sistema se exponen en el Capítulo V.

En el Capítulo VI se presenta el presupuesto del proyecto.

Finalmente, se exponen las principales conclusiones obtenidas tras la realización de este trabajo.

Capítulo II:

Materiales y recursos utilizados.

2. Capítulo II. Materiales y recursos utilizados.

2.1. Introducción

En este capítulo nos centraremos en los materiales y recursos utilizados para lograr el desarrollo y la implementación del enlace óptico, describiendo aquellas herramientas *software* y elementos *hardware* fundamentales en cada parte del proceso.

Se detallará las partes activas en el tratamiento, simulación y observación de las señales, tales como el dispositivo FPGA, el osciloscopio digital, el analizador lógico y la herramienta *software* de simulación **Isim**, además de abarcar los materiales y procesos necesarios para la fabricación de las PCB *(Printed Circuit Board*).

Por último, señalaremos el *software* utilizado en el desarrollo de la programación del comportamiento del canal de transmisión, conjuntamente con el programa necesario para la creación de los circuitos impresos.

2.2. FPGA

Una FPGA es un dispositivo que permite ser programado mediante un lenguaje de descripción especializado (en nuestro caso VHDL, *Very high-speed integrated circuit Hardware Description Language*), logrando reproducir las funciones determinadas en el código del lenguaje.

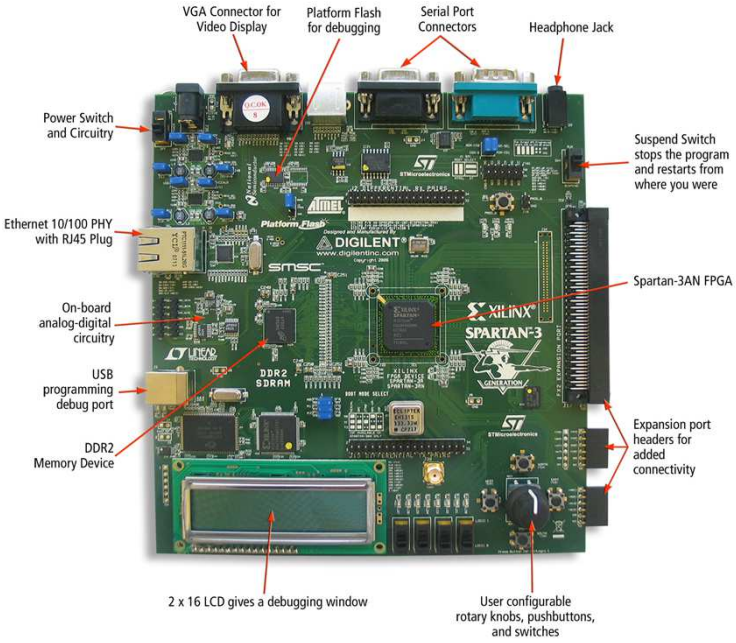


Figura 2.1. Spartan-3AN Starter Kit [Xilinx07].

2.2.1. Tarjeta de desarrollo utilizada

Para la realización del trabajo utilizamos la Spartan-3AN *Starter Kit Board* (figura 4) de Xilinx, basada en una FPGA de la familia Spartan-3AN como su nombre indica. Esta tarjeta de desarrollo cuenta con multitud de conectores, y puertos de entrada y de salida que nos permiten el tratamiento de las señales.

2.2.2. Utilidad

La FPGA será parte fundamental del desarrollo de la transmisión y recepción de la señal. Nos permitirá incorporar los bloques que compondrán ambos circuitos, cuyo comportamiento será descrito en VHDL y se cargará en la FPGA a través de la herramienta de diseño. Los puertos de E/S (entrada/salida) de la FPGA nos permiten visualizar si el funcionamiento interno se produce como se desea, mediante su monitorización, dado que dichos puertos comúnmente son accesibles desde algún conector de la tarjeta de desarrollo.

Una vez tratado el correcto funcionamiento de los diseños descritos en VHDL a nivel digital, utilizando los conectores de expansión de la tarjeta de desarrollo es posible llevar a cabo la transmisión de la señal y su recepción a través del enlace óptico, y nuevamente su digitalización mediante un dispositivo ADC (ver figura 1.3) para que la señal recibida sea procesada por la FPGA. Como vemos, la FPGA es un dispositivo de procesado digital y cualquier señal analógica sobre la que se desee trabajar, en este caso la señal PAM captada por el detector óptico, debe convertirse a digital previamente.

En conclusión, la utilidad de este dispositivo es primordial para el desarrollo del sistema de transmisión, pues lleva a cabo todo el procesado de señales excluyendo únicamente la realizada por los circuitos de transmisión y recepción óptica.

2.2.3. Módulos del kit utilizados

La FPGA está acompañada de módulos con conectores de entrada y salida. En el transcurso del trabajo hemos utilizado estos puertos para la reproducción de las señales. Para poder asociar las ondas a los conectores, creamos un archivo. UCF (*User Constraints File*) donde asignamos las señales a los pines de E/S deseados.

Los puertos manipulados son:

- *Convertidor digital-analógico*:

Permite ver las señales lógicas (palabras de datos) implementadas en los bloques VHDL. El convertidor digital a analógico contenido en la Spartan-3AN *Starter Kit Board* tiene cuatro canales de salida compatibles con el bus SPI (*Serial Peripheral Interface*), los cuales son accesibles desde el conector J21 (Figura 2.2).

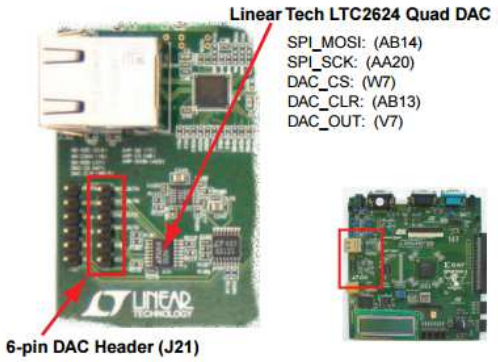


Figura 2.2. Localización del DAC y su conector [Xilinx07].

Para asociar este conector con las señales deseadas creamos el archivo. UCF con las direcciones de pines mostradas a continuación:

Señal	***Dirección Pin FPGA***
SPI_MOSI	AB14
SPI_SCK	AA20
DAC_CS	W7
DAC_CLR	AB13

- *Conectores de Expansión*:

La Spartan-3AN *Starter Kit Board* dispone de múltiples conectores de expansión (figura 2.3). Utilizaremos varios de estos conectores para distintos propósitos. El primer propósito es poder visualizar las señales lógicas a través del analizador lógico, utilizando el conector J17 (conector de 100 pines).

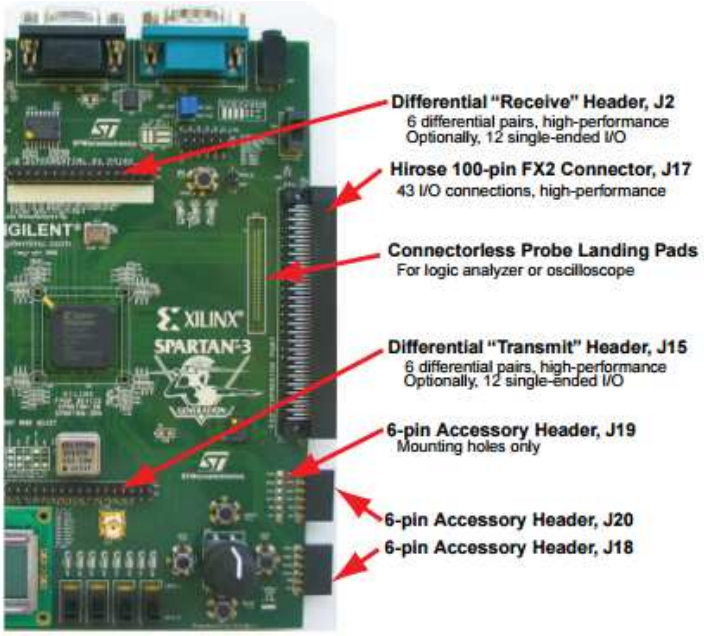


Figura 2.3. Conectores de expansión [Xilinx07].

Signal Name	FPGA Pin	Shared	FX2 Connector		FPGA Pin	Signal Name
		J34	A (top)	B (bottom)		
Supply to FPGA I/O Banks 0, 1, 2	VCCO_012		1	1		SHIELD
	VCCO_012		2	2	GND	GND
TMS_B			3	3		TDO_XC2C
JTSEL			4	4		TCK_B
TDO_FX2			5	5	GND	GND
FX2_IO1	A13	◆	6	6	GND	GND
FX2_IO2	B13	◆	7	7	GND	GND
FX2_IO3	A14	◆	8	8	GND	GND
FX2_IO4	B15	◆	9	9	GND	GND
FX2_IO5	A15	◆	10	10	GND	GND
FX2_IO6	A16	◆	11	11	GND	GND
FX2_IO7	A17	◆	12	12	GND	GND
FX2_IO8	B17	◆	13	13	GND	GND
FX2_IO9	A18	◆	14	14	GND	GND
FX2_IO10	C18	◆	15	15	GND	GND
FX2_IO11	A19	◆	16	16	GND	GND
FX2_IO12	B19	◆	17	17	GND	GND
FX2_IO13	A20	◆	18	18	GND	GND
FX2_IO14	B20	◆	19	19	GND	GND
FX2_IO15	C19	◆	20	20	GND	GND
FX2_IO16	D19	◆	21	21	GND	GND
FX2_IO17	D18	◆	22	22	GND	GND
FX2_IO18	E17	◆	23	23	GND	GND
FX2_IO19	D20		24	24	GND	GND
FX2_IO20	D21		25	25	GND	GND
FX2_IO21	D22		26	26	GND	GND
FX2_IO22	E22		27	27	GND	GND

Figura 2.4. Primeros 27 pines del conector de expansión J17 [Xilinx07].

Por otro lado, conectaremos la PCB transmisora por el conector J18 para emitir la salida del convertidor serie-paralelo diseñado en VHDL a través del LED. Utilizaremos los primeros 5 pines, donde conectaremos el bus de 4 bits del convertidor, además del pin destinado a GND (figura 2.5).

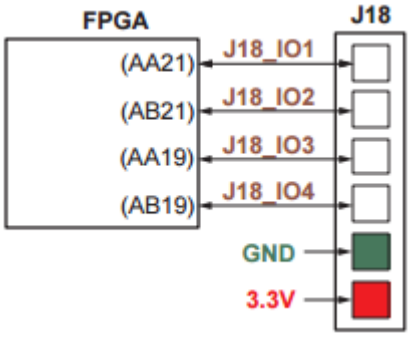


Figura 2.5. Pines del conector J18 [Xilinx07].

Por último, para enlazar la PCB que contiene el ADC, utilizamos los puertos J20 (figura 2.6) y J18. Donde conectaremos a los 8 pines de datos, los 7 bits más significativos del ADC y el reloj del mismo. Además de utilizar un pin para la tierra, con el objetivo de dar referencia a los valores.

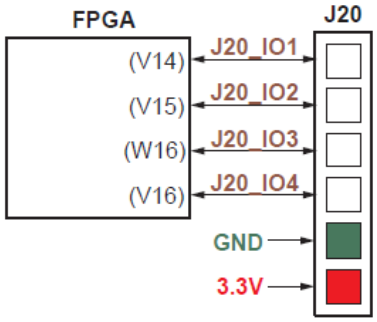


Figura 2.6. Pines del conector J20 [Xilinx07].

2.3. Herramientas *software*

El diseño de nuestro sistema se realiza por medio de diversas herramientas *software*. Nos servirán para la creación de las PCB y para la programación del comportamiento a nivel lógico de nuestro diseño.

2.3.1. ISE Design Suite 14.7

Este *software* de Xilinx permite programar la Spartan-3AN *Starter Kit Board* haciendo que reproduzca el código especificado en él. Dicha herramienta permite definir qué FPGA vamos a usar, qué simulador y qué lenguaje de descripción de *hardware*. El lenguaje elegido será el VHDL, lenguaje diseñado para describir circuitos digitales.

El **ISE Design Suite** es un programa que trabaja directamente con la jerarquía, pudiendo crear escalafones de bloques por medio de instanciaciones, donde los bloques de menor categoría se unen a los bloques superiores. Ese una herramienta de gran utilidad cuando se tiene multitud de códigos, permitiendo controlar el comportamiento de forma más sencilla y óptima (figura 2.7).

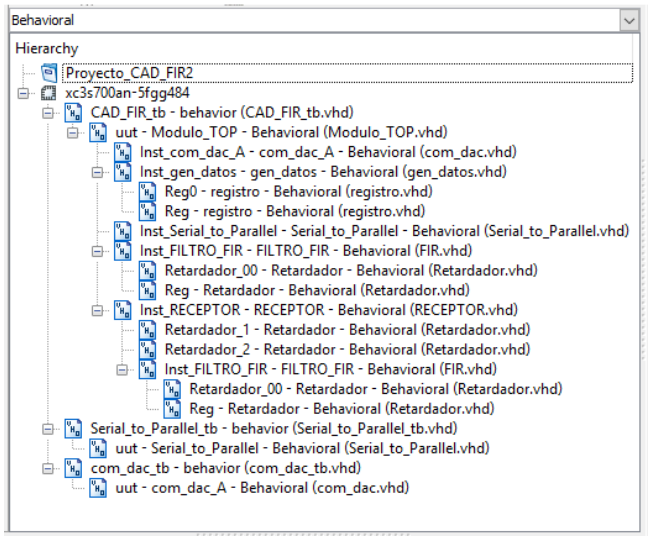


Figura 2.7. Ejemplo de jerarquía en el ISE Design Suite 14.7.

El software contará con dos partes diferenciadas (figura 2.8): la vista de implementación, donde escribiremos el código, y la vista de simulación, en la que simularemos el comportamiento definido en la implementación.

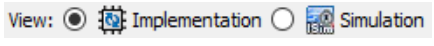


Figura 2.8. Partes del software ISE Design Suite 14.7.

En el *software* podemos elegir multitud de tipos de archivos (figura 2.9). En nuestro caso, la implementación de la programación se definirá en un archivo del tipo *VHDL Module*

para llevar a cabo la descripción de los distintos bloques del sistema a diseñar. El código se compondrá de las librerías, la entidad y la arquitectura, pudiendo así determinar el comportamiento que se reproducirá en el interior de la FPGA (figura 2.10).

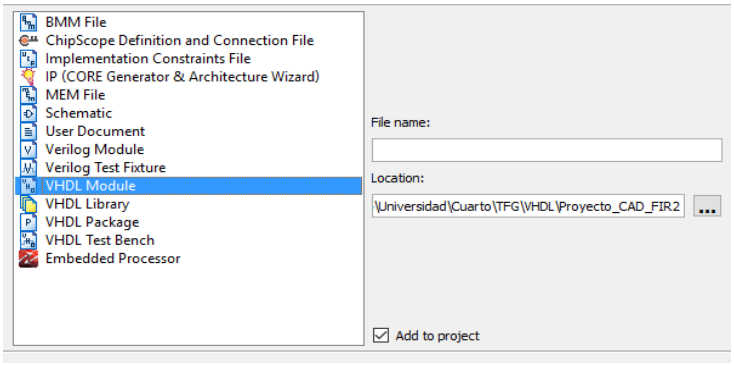


Figura 2.9. Distintos archivos contenidos en el paquete ISE Design Suite 14.7.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

Librerías

```
entity Serial_to_Parallel is
port( Parallel_Data : out std_logic_vector(2 downto 0);
      Clk, Serial_Data, Reset : in std_logic);
end Serial_to_Parallel;
```

Entidad

```
architecture Behavioral of Serial_to_Parallel is
   signal content: std_logic_vector(2 downto 0);

begin
   process(Clk, Reset)
    variable cuenta : natural range 0 to 64;
   begin
      if Reset = '1' then
            content <= "000";
            cuenta := 0;
         elsif Clk'event and Clk = '1' then
            cuenta := cuenta + 1;
         content <=  Serial_Data & content(2 downto 1);
               if (cuenta = 3) then
                  Parallel_Data <= content;
                  cuenta := 0;
               end if;
      end if;
   end process;
end Behavioral;
```

Arquitectura

Figura 2.10. Distintas partes del código VHDL.

Con el fin de obtener un buen control del trabajo realizado en los proyectos de VHDL, pudiendo corroborar lo elaborado, se dispone de una herramienta de simulación, la cual se puede definir en la configuración del proyecto (figura 2.11).

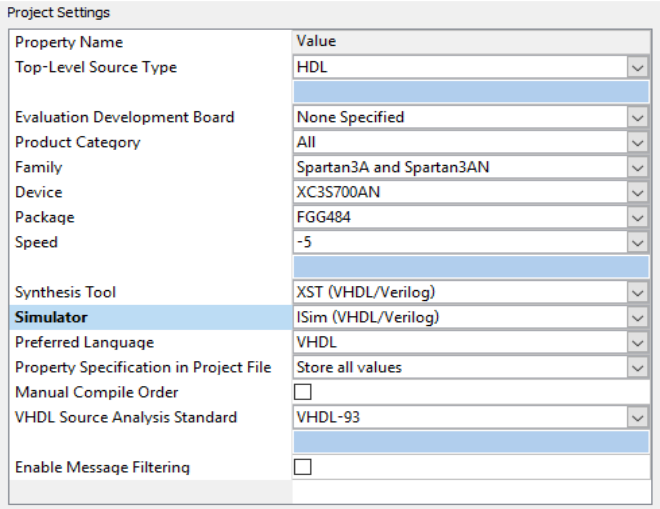


Figura 2.11. Configuración del proyecto en ISE Design Suite 14.7.

Para la simulación, elegimos el simulador ISim adaptado a códigos escritos en VHDL y Verilog. Tras designarlo, asociamos al proyecto un archivo del tipo *VHDL Test Bench* donde definiremos el banco de pruebas y realizaremos la simulación por medio de la herramienta *Simulate Behavioral Model* (figura 2.12) comprobando el comportamiento de las salidas del sistema.

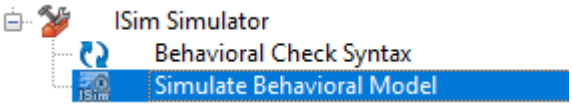


Figura 2.12. Herramienta de simulación del proyecto VHDL.

2.3.2. KiCAD

KiCAD es un *software* libre cuya función es realizar el diseño electrónico de circuitos impresos. Cuenta con una sencilla interfaz (figura 2.13) con las herramientas disponibles para la creación del circuito impreso, con las que podremos crear el esquemático y el *layout*, disponiendo de una amplia variedad de librerías y huellas de componentes.

Figura 2.13. Interfaz del software KiCAD.

Para ello, primero realizamos el esquemático de nuestro circuito en la herramienta *Eeschema* (editor de esquemas electrónicos), en la cual dibujaremos nuestro circuito (figura 2.14).

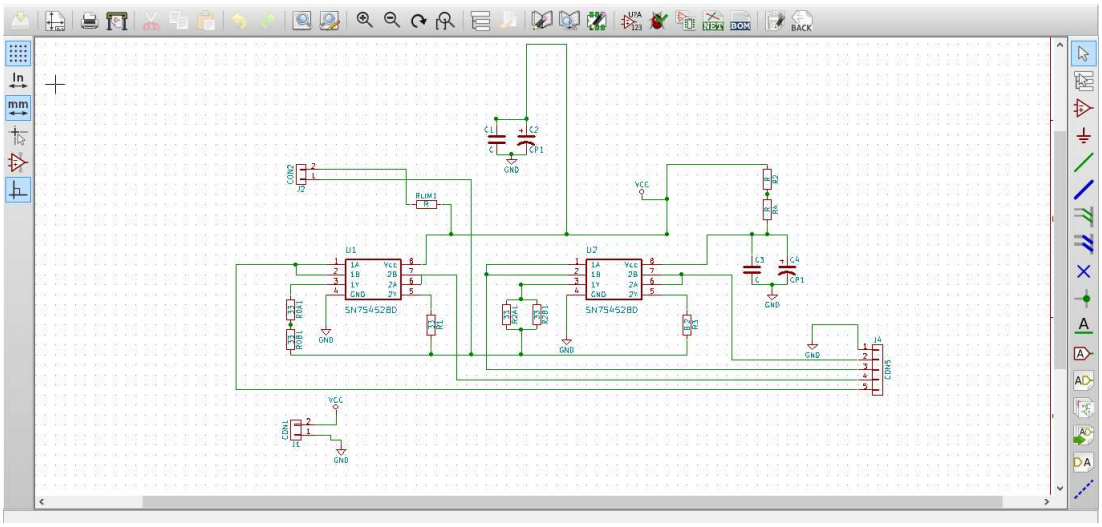


Figura 2.14. Vista de la herramienta Eeschema.

Tras definir el circuito, debemos elegir los componentes adecuados del mismo. Para ello, asignaremos las librerías y las huellas adecuadas a cada encapsulado perteneciente al esquema diseñado, por medio de la herramienta *Cvpcb* (figura 2.15).

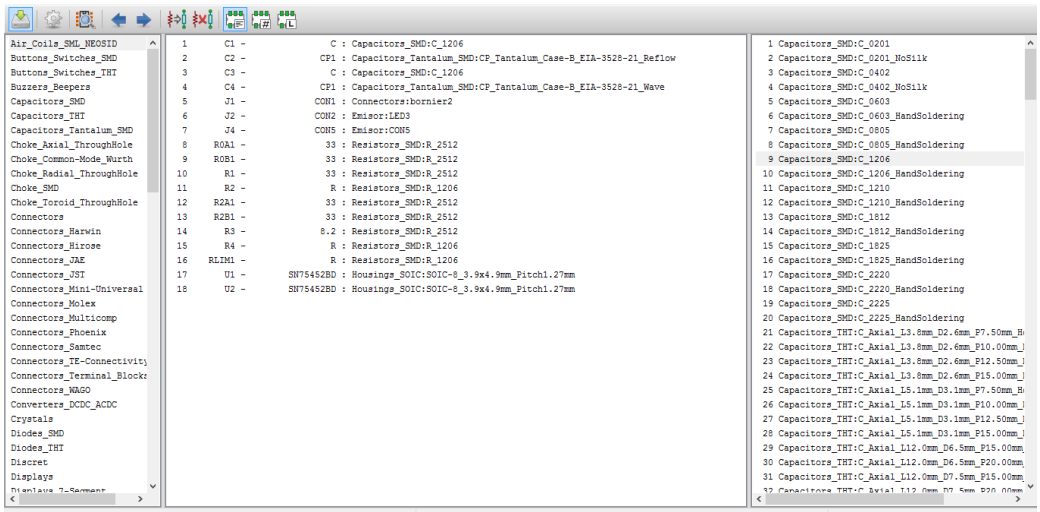


Figura 2.15. Vista de la herramienta Cvpcb.

Una vez asociadas, generamos el *netlist* del circuito, fichero que describe las conexiones de un circuito, utilizando los nudos, dispositivos y pines. Después de generar el *netlist* del circuito, leeremos el archivo en la herramienta *Pcbnew* (editor de placas de circuito impreso) en el que crearemos el *layout* del circuito (figura 2.16) trazando las conexiones entre componentes, las

cuales deben cumplir las reglas de diseño. Ya establecido el *layout*, se generan los fotolitos (ver anexo 3) deseados para la fabricación de la PCB.

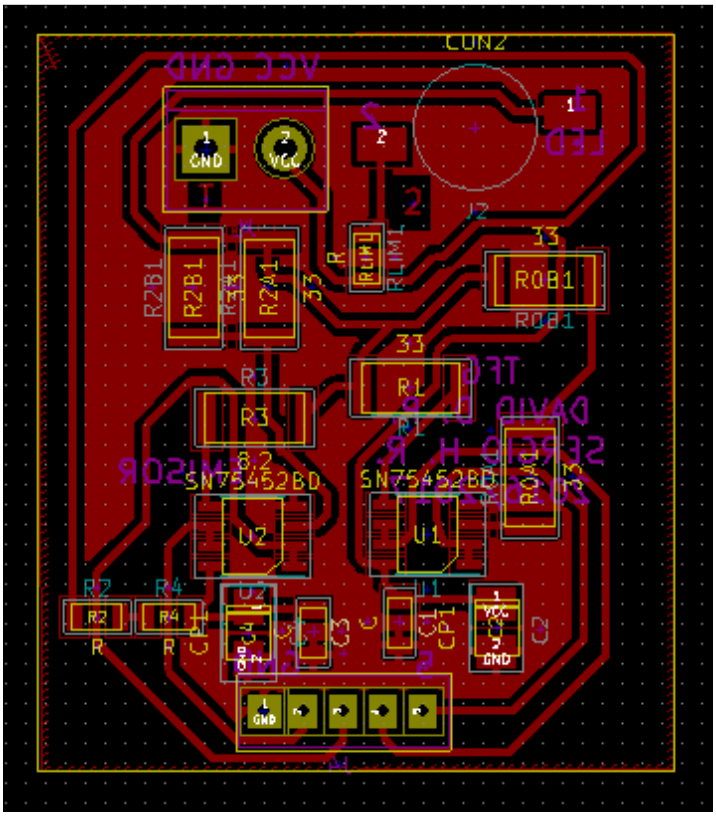


Figura 2.16. Layout final de la placa emisora.

2.4. Material de fabricación de los circuitos impresos

La fabricación de las PCB se puede resumir en los siguientes pasos [1]:

1. Creación del *layout* de cada circuito por medio del *software KiCAD.*

2. Impresión de los fotolitos deseados en transparencias, que actuarán como máscaras fotográficas con el objeto de grabar el mapa de pistas sobre la placa de cobre virgen, además de serigrafiarla.

3. Insolación de los fotolitos en la placa de cobre. Se realiza por medio de la insoladora (figura 2.17), que tras colocar correctamente los fotolitos a través del vacío debilita las zonas donde se requiere eliminar el cobre.

Figura 2.17. Insoladora [2].

4. Proceso de revelado con sosa cáustica, donde introduciremos la placa en una cubeta con la sosa que eliminará la película fotosensible que previamente ha sido debilitada en el insolado.

5. Proceso de atacado con solución acida, momento de eliminación del cobre no protegido por la película fotosensible.

6. Taladrado de los *drills*.

7. Soladura de los componentes SMD (*Surface-Mount Device*) y THD (*Through-Hole Device*).

2.5. Comprobación en un entorno real

Para verificar el comportamiento del sistema de enlace óptico en un caso real, utilizamos los conectores de expansión de la tarjeta de desarrollo de la FPGA anteriormente mencionados (ver apartado 2.2.3), que se guiarán por medio de sondas a un osciloscopio digital RIGOL DS1102D (figura 2.18) que cuenta con dos canales de visualización de señales analógicas y un analizador lógico.

Es de especial importancia comprobar la conducta de todos los bloques implementados, debido a que en los casos reales pueden surgir discrepancias con lo simulado anteriormente, pudiendo desestabilizar todo nuestro sistema.

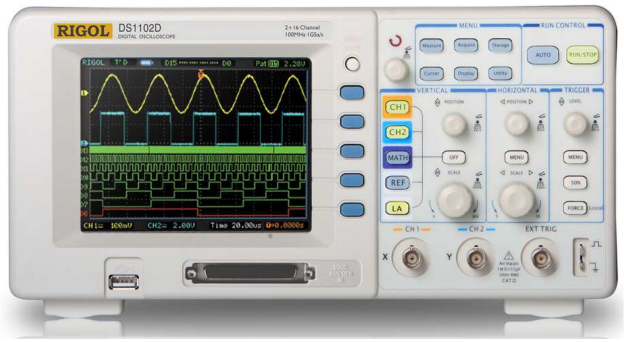


Figura 2.18. Osciloscopio digital RIGOL DS1102D [3].

Capítulo III:

Diseño en VHDL.

3. Capítulo III. Diseño en VHDL.

3.1. Introducción.

Este capítulo se ocupa de la explicación detallada de los distintos módulos VHDL, de las conexiones entre los mismos y de las simulaciones y pruebas realizadas para comprobar su correcto funcionamiento. Para facilitar la comprensión del diseño implementado se mostrarán de forma separada los distintos módulos del emisor y del receptor. Tras la explicación de todos los módulos de cada bloque se atenderá a las instanciaciones realizadas para generar el sistema completo.

Además, se aclarará el funcionamiento del módulo de comunicación con el convertidor digital-analógico de la tarjeta mediante el cual se han realizado gran parte de las pruebas de funcionamiento.

3.2. Bloque Transmisor.

Este bloque se centra en el tratamiento de los datos a que se desean transmitir, ya lleguen de forma externa o se generen internamente a partir del generador de datos pseudoaleatorios. En este proyecto sólo se ha trabajado con datos generados internamente en la FPGA, aunque es fácilmente adaptable a la transmisión de datos provenientes de algún dispositivo externo. La estructura interna del módulo transmisor se muestra en el diagrama de bloques de la figura 3.1. En él se encuentran los bloques siguientes:

- Generador de datos pseudoaleatorios (activo en caso de generación interna de datos).
- Convertidor Serie-Paralelo.
- Bloque generador de Relleno.
- Ralentizador.

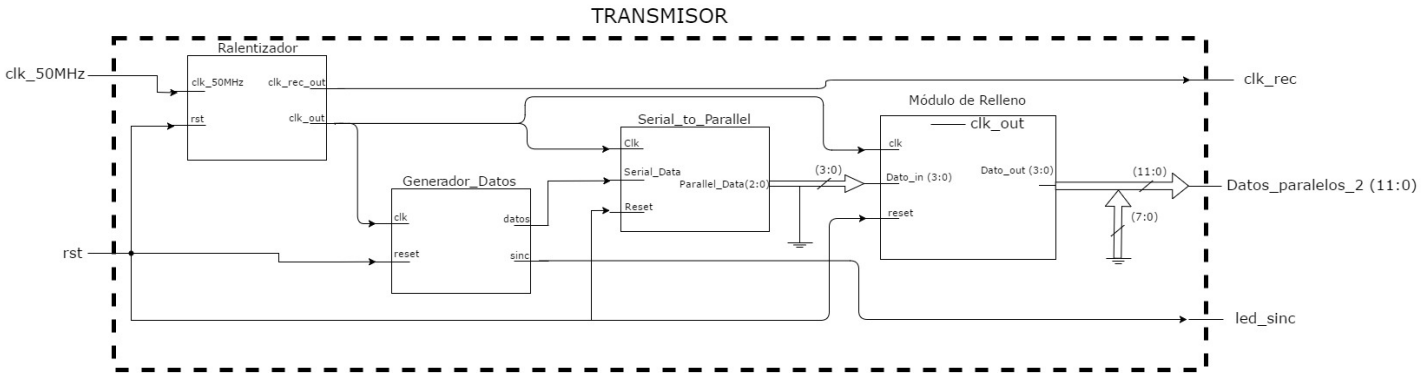


Figura 3.1. Diagrama de bloques del emisor.

3.2.1. Generador de datos aleatorios

3.2.1.1. Objetivo y funcionalidad

El generador de datos pseudoaleatorios es un bloque que permite la producción de una sucesión de bits con una buena aproximación a un conjunto aleatorio de datos. Esta sucesión no es completamente aleatoria ya que se encuentra determinada por un conjunto de valores iniciales. Éste se ha creado con el motivo de simular la creación de un conjunto de datos seriales que permitan comprobar el correcto funcionamiento de los equipos transmisor y receptor.

Para la creación del generador de datos pseudoaleatorios es necesario el uso de registros. El número de los mismos afecta directamente a la longitud de la secuencia pseudoaleatoria. En el caso presente se han utilizado cuatro registros, por lo que la secuencia tendrá una longitud de:

$$2^{n^{o}\,registros} - 1 = 2^{4} - 1 = 15$$

Uniendo estos cuatro registros y diversas puertas lógicas tenemos el generador con la siguiente estructura interna:

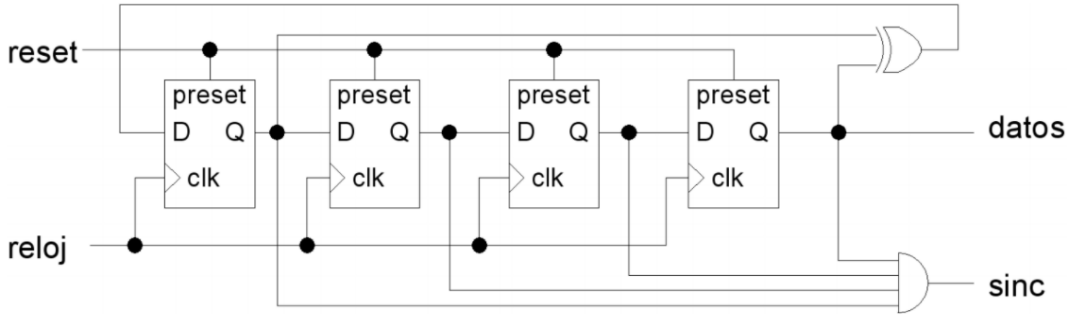


Figura 3.2. Estructura interna del generador de números pseudoaleatorios [4].

Los registros utilizados son los mismos que se encuentra en el bloque retardador, por lo tanto, se explicará en dicho bloque el funcionamiento interno de los mismos.

Por otro lado, el funcionamiento del bloque completo es el que se explica a continuación:

- *Preset* de los registros. Cuando se produce un reseteo del resto de bloques en este caso se ponen a '1' en lugar de a `0', dado que si se hiciera de la forma contraria el generador se quedaría "apagado" de forma indefinida con su salida a cero. Al ponerlo a nivel lógico '1' provoca la iniciación de una nueva secuencia.
- Puerta XOR. Las salidas del primer y el último registro sirven como entrada a esta puerta y su salida realimenta la entrada del primer registro. Esto se realiza para generar la señal pseudoaleatoria en la salida del último registro repitiéndose cada 15 bits.
- Puerta AND. Esta puerta controla la repetición de la secuencia, esto ocurrirá cuando la salida de todos los registros se ponga a uno, momento en el cual se encenderá un led informativo en la SPARTAN 3A.

3.2.1.2. Entradas y Salidas

El bloque tendrá las siguientes entradas y salidas (figura 3.3):

Entradas:

- *clk*: reloj compartido por los módulos del emisor, proveniente del bloque ralentizador.
- *reset*: se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso pondrá a cero los punteros, inicializará los estados de las

máquinas y el dato previo de entrada. Está implementado como un pulsador de la FPGA.

Salidas:

- *datos*: son los datos seriales generados en el bloque.
- *sinc*: señal de sincronización generada cuando todos los registros del generador se encuentran a nivel lógico '1'.

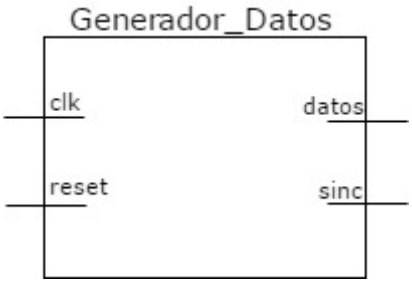


Figura 3.3. Entradas y salidas del generador.

3.2.1.3. Programación VHDL

La programación del bloque es sencilla pues se basa en dos partes diferenciadas: el bloque de registros y las operaciones lógicas. Para su aplicación son necesarias varias señales en el programa:

- **Señales:** se utilizan dos de ellas para la interconexión de las puertas lógicas y los registros, estas son:
  - La señal *sig_xor*, que realimentará la entrada del primer registro con la salida de la suma exclusiva.
  - La señal *Q_int*, la cual es un vector de datos que permitirá referirnos a las distintas salidas de los registros según el índice de la misma.

Tras definir los elementos de programación se explicará la arquitectura del programa.

En primer lugar, se encuentra la sentencia *generate* que instancia los registros del generador. Esta sentencia permite la creación de tanto registros como se deseen los cuales se encuentran interconectados unos con otros a través de sus entradas y salidas. Se realiza en dos partes para definir las conexiones de cada uno de ellos de forma correcta según su lugar en la

estructura. Tras la sentencia se observa cómo se vuelca la salida del último registro (*Nreg* - 1) en la salida del bloque.

```
Generador_datos : for I in 0 to Nreg-1 generate
   Reg00 : if ( I=0) generate
      Reg0 : Registro port map ( clk ,reset ,sig_xor,Q_int(0)) ;
   end generate ;

   Regs : if I>0 generate
      Reg : Registro port map ( clk ,reset,Q_int (I-1) ,Q_int (I)) ;
   end generate ;
end generate;
```

Figura 3.4. Sentencia Generate de instanciación de registros.

Por último, están las operaciones lógicas que se observan en la estructura de la figura 3.5. La primera de ellas es la operación *xor* realizada sobre las salidas del primer y último registro y que es llevada en la señal *sig_xor* anteriormente descrita a la entrada del primero. La segunda es la función "y lógica" que produce la señal de sincronismo cuando todas las salidas del registro se encuentran en el valor '1'.

```
datos <= Q_int (Nreg-1);
sig_xor <= Q_int(0) xor Q_int (Nreg-1);
sinc <= and_vector(Q_int);
```

Figura 3.5. Operaciones lógicas del módulo.

3.2.1.4. Observación en el osciloscopio (analizador lógico)

Para cerciorarnos del correcto funcionamiento del bloque, observamos su comportamiento por medio de la FPGA SPARTAN-3A. Para poder ver la salida de este bloque ha sido necesario utilizar el modo de analizador lógico del osciloscopio digital RIGOL, en el cual podemos ver cómo se generan la señal de bits seriales en cada flanco de subida del reloj *clk*, test que se puede comprobar en la imagen a continuación. Para la conexión de la tarjeta con el analizador se ha precisado el uso del conector J17 de la tarjeta el cual es explicado en el capítulo de recursos y materiales utilizados.

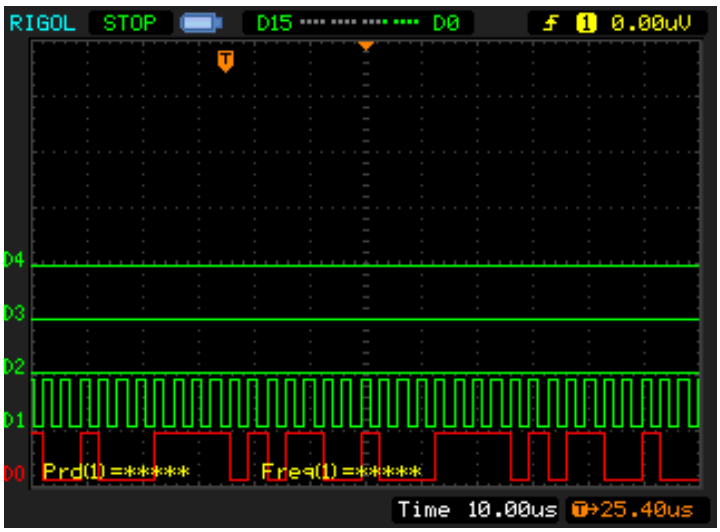


Figura 3.6. Pruebas en el osciloscopio del funcionamiento del generador de datos.

3.2.2. Convertidor Serie-Paralelo

3.2.2.1. Objetivo y función

Este dispositivo se encuentra destinado a la conversión de un conjunto de elementos consecutivos (datos en serie) de una señal digital en otro conjunto de datos que se presentan de forma simultánea (en paralelo), representando la misma información, pero mostrada de forma diferente.

La conversión de los datos seriales a los datos paralelos provoca que la visualización de los mismos sea en forma de escalera, mostrando los distintos niveles de decisión.

En el caso que se presenta, los datos consecutivos se convierten en grupos de tres bits a los que a su vez se les añade un bit adicional (reservado para futuras aplicaciones, aunque en este caso se deja a '0'), resultando un grupo de cuatro bits a la salida del convertidor.

3.2.2.2. Entradas y Salidas

El convertidor Serie-Paralelo tendrá varias entradas seriales y una salida digital de varios bits (Figura 3.7), las cuales son:

Entradas:

- *Clk*: señal de reloj generada en el ralentizador (apartado 3.2.4) a partir de la señal de reloj de 50 MHz de la tarjeta.

- *Reset*: se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso pondrá a cero las variables internas y las salidas digitales del bloque. Esta implementado como un pulsador de la FPGA.

- *Serial_Data*: en esta entrada son introducidos los datos consecutivos que se desean convertir en el grupo de datos que forma el vector de bits.

Salidas:

- *Parallel_Data*: en la salida del bloque se obtendrá un grupo de *N* bits mostrados simultáneamente, siendo *N* el número de bits que deseemos conocer de forma conjunta.
- Además de la salida del bloque se debe hablar de la señal que se obtiene al realizar la concatenación de los datos de salida del convertidor y un conjunto de ocho bits de tierra que permitirán rellenar un total de doce bits que necesitaremos para que el valor de salida pueda ser mostrado de forma correcta a través del conversor digital-analógico (DAC, *Digital-to-Analog Converter*) disponible en la tarjeta de desarrollo. Esta señal recibe el nombre de "*datos_paralelos_2*".

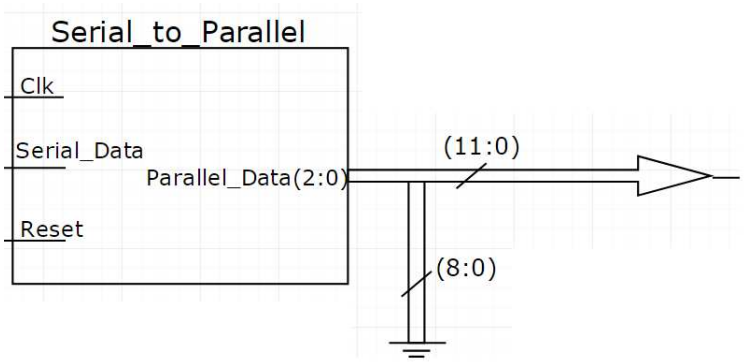


Figura 3.7. Entradas y salidas del bloque convertidor Serie-Paralelo.

3.2.2.3. Programación VHDL

La programación del bloque es muy sencilla, se trata de un único proceso (dependiente del reloj de entrada y del reset). Dicho proceso contiene las siguientes señales y variables:

- **Señales:** En el código VHDL se necesita definir una señal de tipo vector, la cual almacenará de forma temporal el conjunto de valores (bits) hasta que sea vertido en la salida del bloque.
- **Variables:** Hay una variable en el código cuyo fin es contar el número de bits que se han almacenado en la señal temporal. Cuando esta alcanza su valor máximo se volcará el valor de la señal en la salida del bloque y posteriormente se reseteará dicha cuenta.

El funcionamiento del bloque conversor se resume en el desplazamiento del contenido de un registro (señal temporal de almacenamiento) un bit a la derecha, de modo que se cargará el último bit como el más significativo del grupo de tres bits. Esto se realizará en cada flanco de subida de la señal de reloj, mientras la línea de *reset* tenga un valor de '0'. Además de esto, la cuenta aumentará en cada flanco de subida y cuando ésta alcance el valor del tamaño del grupo de datos paralelos se copiará el contenido de la señal temporal en la salida del bloque.

3.2.2.4. Simulación VHDL

Para la simulación previa se ha utilizado un archivo de pruebas (*test bench*) en el cual se ha simulado la llegada, en cada flanco de subida de un reloj cuyo periodo es de diez nanosegundos, de varios bits y se ha comprobado que la salida del grupo de bits se corresponde a lo esperado.

```
clk <= not clk after 5 ns;
Reset <= '1', '0' after 2 ns;
Serial_Data <= '1', '0' after 15 ns, '1' after 20 ns, '0' after 35 ns, '0' after 45 ns,
'0' after 55 ns, '1' after 65 ns;
```

Figura 3.8. Variables utilizadas en la simulación del convertidor Serie-Paralelo.

A continuación, en la figura 3.9, se comprueban los resultados de la simulación. Los vectores de bits se forman al completar la tercera cuenta con los datos de los últimos tres flancos seleccionados en orden inverso al de entrada.

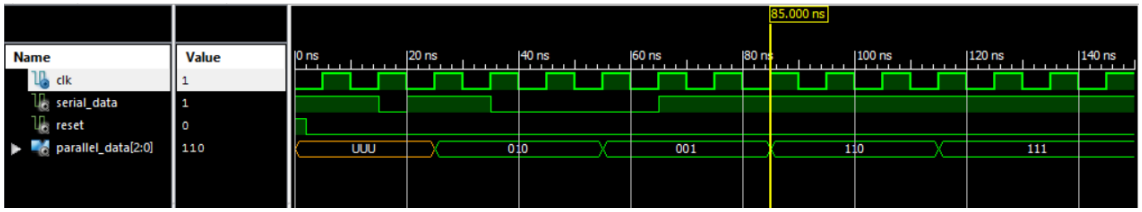


Figura 3.9. Simulación del test bench del convertidor Serie-Paralelo.

3.2.2.5. Simulación en el laboratorio.

Para la comprobación del correcto funcionamiento del bloque también se han realizado varias simulaciones con el osciloscopio digital RIGOL y la FPGA utilizada. Para la visualización de los resultados ha sido necesaria la concatenación a la derecha de los datos de salida con un bus de ceros de nueve bits ya que es necesario tener un bus de doce bits para que la señal pueda

ser convertida a una señal analógica y poder visualizarse en el DAC. Como podemos ver en la figura 3.10 a continuación, la señal obtenida es periódica y tiene distintos niveles de tensión algo que tiene sentido ya que la señal de salida del bloque es paralela. Recuérdese que el generador pseudoaletario generaba una secuencia de longitud 15 bits, y nosotros entregamos en paralelo los datos de 3 en 3 bits, de ahí que haya siempre cinco niveles distintos en la señal en escalera que se vuelven siempre a repetir.

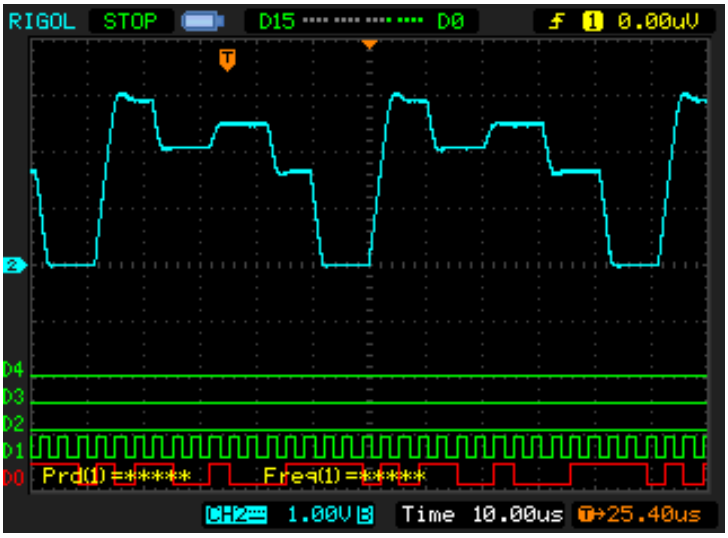


Figura 3.10. Simulación en el osciloscopio del convertidor Serie-Paralelo.

3.2.3. Módulo de Relleno

3.2.3.1. Objetivo y funcionalidad

El objetivo de este módulo es rellenar con una secuencia (de nivel de amplitud promedio) la salida del bloque transmisor, que actuará en el caso de que se requiera emitir datos y no existan dichos datos, por lo que se debe de “rellenar” para no tener pérdida de “información” a la entrada del led y no cerrar el canal de transmisión (evitar que el led deje de encenderse).

Este módulo se conecta a la salida del convertidor serie-paralelo, cuya salida es un bus de 4 bits. En el caso de que el convertidor no le proporcione ningún dato, el bloque de relleno emitirá una “F” (“1111”) y un “0” (“0000”), para dar un nivel medio constante de voltaje, y dar tiempo al convertidor a facilitar nuevos datos.

Para almacenar los datos de entrada y poder emitir los datos de entrada o los datos de relleno, dependiendo del caso, se debe de utilizar un buffer circular (figura 3.11). Este buffer contará con un único reloj, que marcará la recepción de datos y se encargará de transmitirlos. Además,

utilizará un puntero de escritura y otro de lectura, que se relacionarán con la entrada (escritura de datos) y la salida (lectura de datos) del bloque respectivamente. Por lo tanto, iremos almacenando los datos en la pila de memoria, y en el caso de que no existan datos en el momento de transmisión se escribirá en el buffer los datos de relleno.

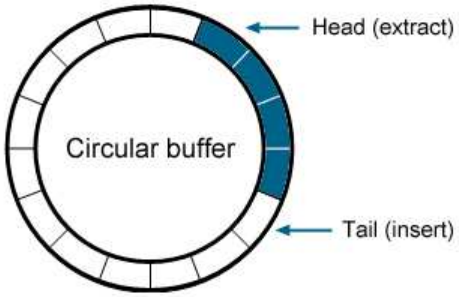


Figura 3.11. Buffer Circular [5].

Hay que tener en cuenta que en el receptor se implementará un módulo que quitará el relleno para recoger la información real suministrada. Por ello se deben crear dos máquinas en este módulo:

- **Máquina de entrada**: Esta máquina se encargará de evitar que el módulo que quita el relleno, borre datos reales de entrada. Es decir, que no confunda los datos "F" y "0" de manera consecutiva con relleno, cuando en realidad son datos reales.

  Para ello, esta máquina debe de escribir una secuencia de datos de relleno para engañar al módulo que elimina dicho relleno en el extremo receptor. Siempre que se tenga entre los datos los valores "F0" de manera consecutiva, se introduce otra secuencia "F0" entremedio de dichos datos. La secuencia resultante "FF00" hará que en el receptor se eliminen los datos consecutivos "F0" centrales (los introducidos como relleno), dejando el "F0" de los datos reales, de la siguiente manera:

  $$FF00 \rightarrow F0$$

- **Máquina de salida:** Este elemento se encargará de enviar el relleno, cuando en el momento de transmisión no existan datos. Sabrá que no hay datos cuando el puntero de lectura (puntero relacionado con la salida) alcance al puntero de escritura (puntero relacionado a la entrada). En este momento no habrá dato que enviar puesto que el

puntero de escritura siempre está en la posición donde se escribirá el siguiente dato que entre. Por tanto, si el puntero de lectura (salida de datos) está en la misma posición que el de escritura, en esa posición no puede haber información que enviar.

3.2.3.2. Entradas y Salidas

El bloque tendrá las siguientes entradas y salidas (figura 3.12):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador, que marcará la llegada de datos y su correspondiente transmisión.
- *reset*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso, pondrá a cero los punteros, inicializará los estados de las máquinas y el dato previo de entrada. Está implementado como un pulsador de la FPGA.
- *Dato_in*: Entrada de tipo bus de 4 bits, que recibirá los datos de entrada provenientes del convertidor serie-paralelo.

Salidas:

- *Dato_out*: Salida de tipo bus de 4 bits, que se corresponden con los datos que se emitirán al receptor.

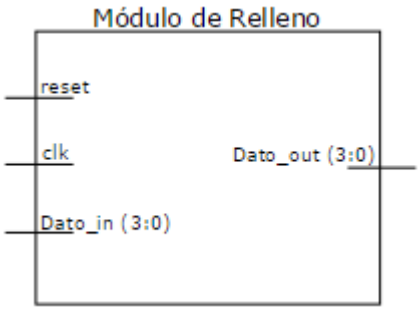


Figura 3.12. Módulo de relleno.

3.2.3.3. Programación VHDL

La programación se divide en dos grandes bloques, la máquina de entrada y la máquina de salida (explicadas anteriormente), ambas dependientes del reloj de entrada y del reset.

Además de ellas, existen variables y señales que se utilizan en ambos procesos, las cuales son (figura 3.13):

- **Type pila:** Definimos el tipo de nuestro buffer de memoria, el cual será una matriz 8x4. Lo necesitamos para poder asignarle el tipo a una señal.
- **Shared Variables:** Creamos los iteradores que recorrerán el buffer de memoria. Lo hacemos de tipo *shared*, para que se puedan declarar fuera de los procesos.
- **Señales:**
  - La señal *dato_previo* se conectará con el dato previo de entrada, ya que la máquina de entrada compara el dato previo con el dato que actualmente le llega, para saber si se trata de un "F" y un "0" consecutivo.
  - *Buffer_datos* es una señal de tipo pila, donde se almacenarán los datos para posteriormente transmitirlos.

```
architecture Behavioral of Relleno is

type pila is array(dim_pila-1 downto 0) of std_logic_vector (N-1 downto 0); -- Matriz 8x4
Shared Variable puntero_in : integer := 0; -- iterador puntero entrada
Shared Variable puntero_out : integer := 0; -- iterador puntero salida
signal dato_previo : std_logic_vector (N-1 downto 0);
signal buffer_datos : pila;

begin
```

Figura 3.13. Shared Variables y señales del módulo de relleno.

Tras definir aquellos elementos que estarán en ambos procesos, comienza la programación de los dos bloques en los que se define el bloque:

- **Máquina de entrada:** Se trata de un proceso dependiente del *clk* y del reset. Este va comparando el dato previo de entrada (que se va actualizando en cada flanco de bajada del reloj, tras concluir el proceso) con el dato de entrada actual.

  Imponiendo la condición de que si el dato previo tiene el valor de *detecta1* (constante definida en el paquete de constantes, de valor "1111") y el dato actual de entrada tiene el valor de *detecta2* (constante determinada en el paquete de constantes, de valor "0000"), se introducirá la secuencia de relleno en el buffer. Para introducir la secuencia, vamos apuntando con el iterador a la posición deseada del buffer y vamos escribiendo en él, una vez terminado actualizamos la posición del puntero para que el

siguiente dato no se sobrescriba. En el caso de que no se dé la condición anterior, directamente se escribe el valor de la entrada. Tras terminar las secuencias se actualiza el dato previo. Su código es el siguiente:

```
ENTRADA: process (clk, reset)

begin

if (reset = '1') then
   puntero_in := 0;
   dato_previo <= previo;

--Si hay un flanco de bajada:
elsif ( clk'event and clk = '0') then
   --Y si el dato previo es "1111" y el dato que entra es "0000":
   if ( dato_previo = detecta1 and Dato_in = detecta2 ) then
         --Si se cumple la condición escribimos la secuencia de FF00, como el dato previo ya
         --es una F, solo bastaría con introducir FF0 al buffer, escribiendo cada dato en la posición
         --del puntero.
         buffer_datos(puntero_in) <= relleno1; --Escribimos "F"
         buffer_datos((puntero_in + 1) mod dim_pila) <= relleno2; --Escribimos "0"
         buffer_datos((puntero_in + 2) mod dim_pila)<= relleno2; --Escribimos "0"

         puntero_in := (puntero_in + 3) mod dim_pila; --Movemos la posición del puntero a la posicion posterior

   else

      buffer_datos(puntero_in) <= Dato_in; --Si no, escribes el dato de entrada
      puntero_in := (puntero_in + 1) mod dim_pila; --Ruedas una posición el puntero

   end if;

   dato_previo <= Dato_in; --Actualizas dato previo

end if;

end process;
```

Figura 3.14. Máquina de entrada del relleno.

- **Máquina de salida:** Es una máquina de estado del tipo Moore, es decir, la salida sólo depende del estado de la máquina. Por ello, para definirla debemos de saber la salida dependiendo del estado y cómo se relacionan los estados entre sí. El objetivo de este proceso es escribir el relleno cuando no exista dato que emitir.

  Creamos un proceso que dependa del reset y del reloj. En un primer paso definimos los estados y las señales que nos permitirán pasar de un estado a otro (figura 3.15). Posteriormente, con el reset asignamos el estado inicial de la máquina (figura 3.16).

```
WORK: process (clk, reset)
--Definimos los tres estados de nuestra máquina
type state_type is (st_normal, st_relleno1, st_relleno2);

--Determinamos las variable que determinará el estado actual y el siguiente estado
Variable state, next_state : state_type;

begin
```

Figura 3.15. Tipos de estado y variables del módulo de relleno.

```
if (reset = '1') then

      state := st_normal ;  -- Asignación del estado inicial
      next_state := st_normal;  -- Asignación del estado inicial
      puntero_out := 0; --Reseteamos el puntero de salida
```

Figura 3.16. Asignación del estado inicial en el módulo de relleno.

Si se cumple que el reset está desactivado y existe un flanco de subida del reloj comenzamos con el comportamiento del programa. Tras definir los estados, determinamos sus salidas. Dichos estados serán:

- *Estado normal:* La máquina se encuentra en este estado cuando no hay que aplicar relleno, es decir, hay dato que transmitir. Por lo tanto, la salida del estado se corresponderá a la dirección del puntero de salida dentro del buffer de datos (figura 3.17).

```
 --Si estas en el estado normal, escribes en la salida, lo que se haya almacenado en el buffer
if (state = st_normal) then

   Dato_out <= buffer_datos(puntero_out);
   estado <= "00";
end if;
```

Figura 3.17. Salida del estado normal en el módulo de relleno.

- *Estado Relleno1:* Se entraría en este estado cuando la máquina no tiene dato que emitir, debiéndose de escribir el relleno. Este estado se encarga de escribir la primera palabra del relleno (figura 3.18).

```
--Primer estado de relleno, sacas la "F"
if (state = st_relleno1) then
      Dato_out <= relleno1;
      estado <= "01";
end if;
```

Figura 3.18. Salida del estado relleno1 en el módulo de relleno.

- *Estado Relleno2:* Este estado se encarga de escribir la segunda palabra del relleno (figura 3.19).

```
--Segundo estado de relleno sacas el "0"
if (state = st_relleno2) then
      Dato_out <= relleno2;
      estado <= "10";
end if;
```

Figura 3.19. Salida del estado relleno2 en el módulo de relleno.

Una vez definidas las salidas, establecemos cómo pasamos de un estado a otro, haciendo uso de las sentencias case y analizando cada caso dependiendo del estado en el que estemos.

Si se encuentra en el estado *normal,* debemos comprobar la relación entre punteros: si el puntero de escritura (*puntero_in* en el código) se encuentra por encima del puntero de escritura (*puntero_out*), significa que hay un dato real para emitir, por lo que se sigue en el estado normal. Además, se avanza una posición del puntero de lectura para escribir el siguiente dato. Si la relación no se da, se pasa a los estados de relleno (figura 3.20).

```
case (state) is

--Si estás en el estado normal, y el puntero de entrada es distinto al de la salida, es decir hay dato real,
--sigues en el estado normal y aumentas una posición al puntero de salida. Si no, pasas al estado relleno 1.
when st_normal =>
         if ( puntero_in /= puntero_out) then
            next_state := st_normal;
            puntero_out := (puntero_out + 1) mod dim_pila;
         else
            next_state := st_relleno1;
         end if;
```

Figura 3.20. Paso de estados, para el estado normal en el módulo de relleno.

Entrado en el estado de *relleno1,* pasamos al estado de *relleno2* sin condiciones (figura 3.21) para escribir el relleno correctamente, además no se incrementa el puntero de salida.

```
--Cuando estás en el relleno1 pasas al relleno2
when st_relleno1 =>
            next_state := st_relleno2;
```

Figura 3.21. Paso de estados, para el estado relleno1 en el módulo de relleno.

Tras pasar por el *relleno2*, se consigue escribir el relleno correctamente, por lo que se vuelve al estado normal, donde se comparan los iteradores de nuevo (figura 3.22).

```
when st_relleno2 =>
            next_state := st_normal;

end case;
```

Figura 3.22. Paso de estados, para el estado relleno2 en el módulo de relleno.

Por último, definimos la señal *state* con el valor de *next_state*, para actualizar el estado de nuestra máquina (figura 3.23).

```
state := next_state;
```

Figura 3.23. Actualización del estado en el módulo de relleno.

El módulo de relleno se encontrará programado, definiendo las máquinas anteriormente citadas.

3.2.3.4. Simulación VHDL

Hacemos uso de la simulación gracias a la herramienta *Simulate Behavioral Model*, utilizando el *test bench* de nuestro código. En dicho *test bench* (figura 3.24), introducimos valores aleatorios de entrada, entre ellos un “F” y un “0” consecutivo (para ver cómo se comporta el módulo), simulamos el reloj y definimos el reset.

```
clk <= not clk after 5 ns;
reset <= '1' after 0 ns, '0' after 2 ns;
Dato_in <= "1111" after 0 ns, "1111" after 25 ns, "1010" after 35 ns, "1111" after 45 ns, "0000" after 55 ns, "1010" after 65 ns;
```

Figura 3.24. Test bench del módulo de relleno.

Comprobando que efectivamente realiza el relleno (podemos observarlo en los cambios de estado) y que introduce la secuencia “FF00” cuando corresponde (figura 3.25).

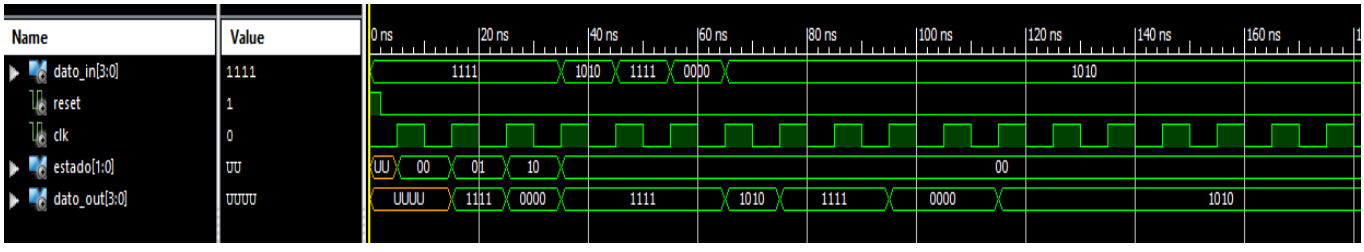


Figura 3.25. Simulación del módulo de relleno.

Se puede observar que realiza correctamente el relleno a los 15 ns hasta los 25 ns por no tener datos que transmitir, y que introduce la secuencia de relleno (al encontrar una secuencia de datos "F0") desde los 75 ns hasta los 115 ns, además de emitir los datos correctamente en el resto de los casos.

3.2.4. Ralentizador

3.2.4.1. Objetivo y función

El objetivo de este bloque es crear relojes de distintas velocidades partiendo del reloj principal de la FPGA SPARTAN-3A, el cual trabaja a una frecuencia de 50 MHz. La función de crear estos relojes es proporcionar un mejor sincronismo entre bloques, permitiendo al CAD de la FPGA (ya que no es lo suficientemente rápido) y al filtro FIR implementado en el receptor, poder trabajar correctamente. Dichos relojes serán:

- *Clk_out*: Reloj destinado a los componentes del transmisor, su periodo será 168 veces más lento que el reloj original, debido a que el DAC es demasiado lento, necesitando un mayor tiempo para un uso óptimo del mismo.
- *Clk_rec_out*: Reloj dirigido a los elementos del receptor, que es 63 veces más lento que el original. Esto es así para permitir que al filtro FIR (*Finite Impulse Response*) implementado en el receptor (ver apartado 3.3.1) le dé tiempo a procesar cada palabra digital de datos adecuadamente.

Observe que existe una relación entre relojes de 168/63 = 8/3, es decir, cuando *Clk_rec_out* complete ocho pulsos, *Clk_out* debe haber realizado tres. Esto es debido a que el filtro FIR implementado en el receptor requiere ocho ciclos de reloj para procesar cada palabra digital de entrada, que en el transmisor fue generada mediante la concatenación de tres bits de datos. Además, se requiere un mínimo de 63 ciclos del reloj de la tarjeta (50 MHz) para el funcionamiento del DAC, de ahí que se haya establecido este valor para generar cada ciclo del reloj que controla el filtro FIR.

Por todo lo anteriormente expuesto, este bloque es indispensable para el sincronismo de todo el sistema, pudiendo tratar las señales de la manera deseada.

3.2.4.2. Entradas y Salidas

El ralentizador tendrá entradas y salidas digitales de un bit (Figura 3.26), las cuales son:

Entradas:

- *clk_50MHz*: Reloj de la FPGA SPARTAN-3A.
- *rst*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso pondrá a cero las variables internas y las salidas digitales del bloque. Está implementado como un pulsador de la FPGA.

Salidas:

- *clk_rec_out*: Reloj destinado al receptor.
- *clk_out*: Reloj utilizado para el transmisor.

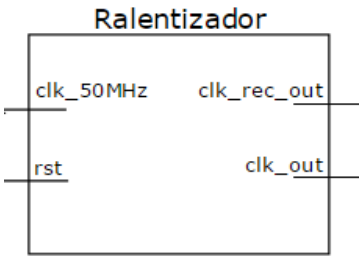


Figura 3.26. Entradas y salidas del bloque ralentizador.

3.2.4.3. Programación VHDL

La programación del bloque es muy sencilla. Se trata de un único proceso (dependiente del reloj de entrada y del reset). Dicho proceso contiene las siguientes señales (Figura 3.27), constantes y variables (Figura 3.28):

- **Señales:** El programa necesita dos señales, las cuales se conectarán a los relojes de salida.
- **Constantes:** Existe una constante llamada *cuenta_ralentizador*, del tipo natural y cuyo valor es de 504. Tiene dicho valor porque es múltiplo de 63 y 168, por un factor de 8 y 3, respectivamente, lo que es necesario para la creación de los relojes de salida.
- **Variables:** Hay dos variables, dos cuentas independientes, una para cada reloj de salida. Estas cuentas se incrementarán en cada flanco de subida del reloj de entrada. Son cuentas naturales, cuyo valor inicial es cero.

```
--Señales utilizadas:
   signal clk, clk_rec: std_logic := '0';
```

Figura 3.27. Señales implementadas en el proceso del bloque ralentizador.

```
--Constante definida, que servirá a la hora de la comparación, debe ser multiplo de 3 y de 8.
constant cuenta_ralentizador : natural := 504;
--Variables que se incrementarán en cada flanco del reloj de entrada
variable cuenta,cuenta_rec : natural range 0 to cuenta_ralentizador :=0;
```

Figura 3.28. Variables y constantes del bloque ralentizador.

Tras declarar las constantes y variables, entramos en las sentencias IF. La primera sentencia da la condición de si el reset está activado o no (Figura 3.29). Si lo está, las variables y las salidas del bloque se ponen a cero.

```
--Si el reset es 1, las variables, y las salidas se ponen a 0
if rst = '1' then
   cuenta := 0;
   cuenta_rec := 0;
   clk <= '0';
   clk_rec <= '0';
```

Figura 3.29. Condición del reset del bloque ralentizador.

En caso de que el reset no esté activado y haya un flanco de reloj, se comenzará con el programa. Cuando se cumpla la condición anterior, las variables se incrementarán en uno a cada flanco del reloj de entrada, e iremos comparando el valor de cada cuenta con las siguientes condiciones para crear los periodos de los relojes:

- **Periodo del reloj de salida clk_out:** En concordancia al reloj destinado al receptor, debe dar tres pulsos cuando el reloj del receptor dé ocho. Para ello hacemos que se mantenga en cada semiperiodo 84 cuentas del reloj original (es decir que *cuenta* valga 84), siendo el periodo completo de 168 cuentas, obteniendo tres pulsos según la relación a la constante ($\frac{\text{cuenta\_ralentizador}}{168} = 3$).

  Para lograr esto, se va comparando la variable *cuenta* con la constante definida (Figura 3.30). Para la comparación se utiliza la constante especificada en el bloque de constantes *LogL*, cuyo valor es 3.

```
--Si la cuenta para el reloj clk_out, es menor que cuenta_ralentizador/(LogL*2), es decir menor
--que 84, es 1, en otro caso sería 0. Pero como la cuenta se resetea cuando
--cuenta = (cuenta_ralentizador/LogL). Tenemos que el reloj se mantiene a 1 durante 84 cuentas y a 0 durante
--otras 84 cuentas, dando un periodo total de 168 cuentas, haciendo que este reloj tenga una proporción
-- de 3 pulsos en relación a la constante(504/168 = 3).

if cuenta < cuenta_ralentizador/(LogL*2) then
   clk <= '1';
else
   clk <= '0';
end if;
```

Figura 3.30. Condición del reloj clk_out.

Para que el periodo sea el deseado, se resetea la cuenta (Figura 3.31).

```
--Se resetea la cuenta, cuando:
if cuenta = (cuenta_ralentizador/LogL) then
   cuenta := 0;
end if;
```

Figura 3.31. Reset de la variable cuenta en el bloque ralentizador.

- **Periodo del reloj de salida *clk_rec_out*:** Debe de tener ocho pulsos en dependencia a la constante definida (consiguiendo la relación de 8/3 anteriormente citada), por lo que hacemos que se mantenga en cada semiperiodo 31,5 cuentas del reloj original (*cuenta_rec* igual a 31,5), siendo el periodo completo de 63 cuentas, teniendo una proporción de 8 pulsos respecto a *cuenta_ralentizador* $(\frac{\text{cuenta\_ralentizador}}{63} = 8)$.

  Para ello, al igual que el anterior reloj, se compara la variable *cuenta_rec* con la constante definida (Figura 3.32). Además, se utiliza la constante *L* determinada en el bloque de constantes, cuyo valor es $2^{LogL} = 8$.

```
--Si la cuenta para el reloj clk_rec_out, es menor que cuenta_ralentizador/(2*L), es decir menor
--que 31.5, es 1, en otro caso sería 0. Pero como la cuenta se resetea cuando
--cuenta = (cuenta_ralentizador/L). Tenemos que el reloj se mantiene a 1 durante 31.5 cuentas y a 0 durante
--otras 31.5 cuentas, dando un periodo total de 63 cuentas, consiguiendo que el reloj dé ocho pulsos (504/63 = 8).

if cuenta_rec < cuenta_ralentizador/(2*L) then
   clk_rec <= '1';
else
   clk_rec <= '0';
end if;
```

Figura 3.32. Condición del reloj clk_rec_out.

Se reseteará *cuenta_rec* para que el periodo sea el adecuado (Figura 3.33).

```
--Se resetea cuenta_ec, cuando:
if cuenta_rec = (cuenta_ralentizador/L) then
   cuenta_rec := 0;
end if;
```

Figura 3.33. Reset de la variable cuenta_rec.

Una vez creados los relojes solo queda conectar las señales *clk* y *clk_rec* a los relojes de salida (Figura 3.34), obteniendo los relojes deseados.

```
clk_out <= clk;
clk_rec_out <= clk_rec;
```

Figura 3.34. Conexión de las señales en el bloque ralentizador.

3.2.4.4. Observación en el osciloscopio

Además de realizar la simulación con el simulador del paquete ISE se ha realizado una pequeña prueba en la que, gracias al analizador lógico del osciloscopio RIGOL, podemos ver las diferencias temporales entre los distintos relojes que se utilizarán para sincronizar los módulos tanto de los módulos receptor como emisor. Estas diferencias son notorias en la figura 3.35 que se presenta tras el final del párrafo.

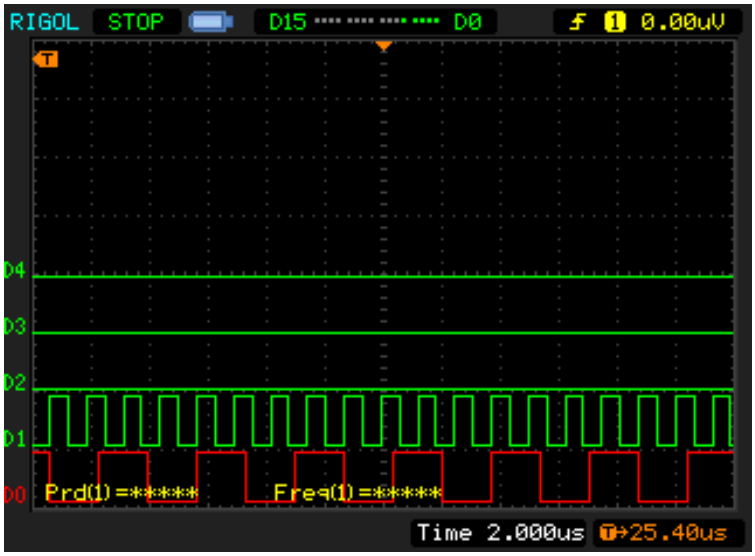


Figura 3.35. Pruebas del ralentizador.

3.2.5. Instanciación de los módulos en el bloque emisor.

La instanciación de los módulos se realiza de acuerdo al diagrama de bloques que se muestra en la figura 3.1, presentada antes de comenzar la explicación de cada uno de los módulos del bloque transmisor. Dicha instanciación se muestra en los anexos del código mostrados tras el último capítulo.

3.3. Bloque Receptor.

Este bloque se centra en el procesamiento de la señal convertida por el ADC que ha sido recibida por el fotodiodo. La estructura interna de dicho módulo se muestra en el diagrama de bloques de la figura 3.36. En él se encuentran los bloques siguientes:

- Filtro FIR.
- Derivadores.
- Detector de umbral.
- Generador de reloj.
- Bloque retardador.
- *Sample & Hold*.
- Decodificador.
- Bloque eliminador de relleno.

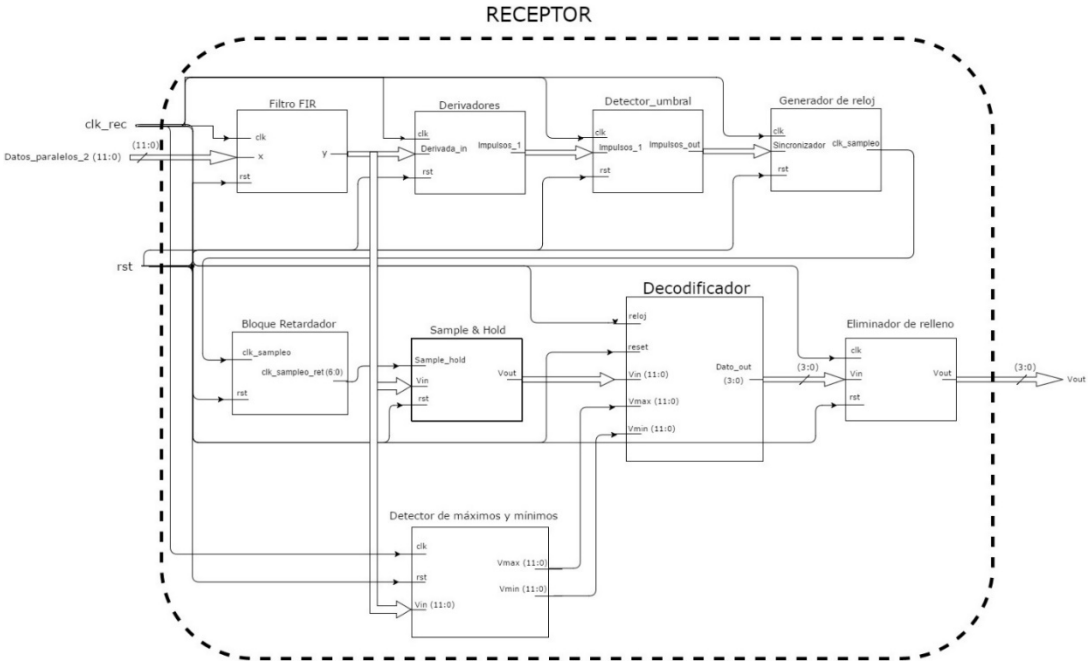


Figura 3.36. Diagrama de bloques del receptor.

3.3.1. Filtro FIR.

3.3.1.1. Objetivo y funcionalidad

El filtro FIR (respuesta finita al impulso) es un filtro digital cuya respuesta a una señal impulso como entrada tendrá un número finito de términos no nulos. Los filtros digitales son un tipo de filtros que trabajan sobre señales discretas y cuantizadas. Estos filtros se encuentran basados en una operación matemática que toma una secuencia de números (la señal de entrada) y la modifica produciendo otra secuencia de números (la señal de salida) con el objetivo de resaltar o atenuar ciertas características.

Los filtros digitales tienen una gran cantidad de aplicaciones entre las que se encuentra la recuperación de señales distorsionadas de alguna forma, como por ejemplo al ser transmitidas.

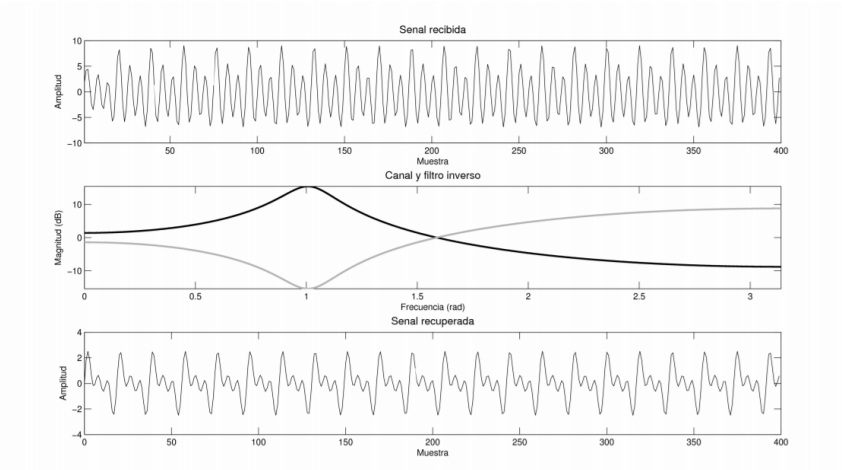


Figura 3.37. Recuperación de la señal tras ser transmitida [6].

Un filtro se puede caracterizar de tres formas equivalentes: según su respuesta al impulso, su respuesta a la frecuencia o su respuesta al escalón. El filtro FIR está especialmente caracterizado por la respuesta al impulso. Esta respuesta al impulso de un filtro es la respuesta a una entrada impulsiva, que en el caso del FIR cae a cero tras un tiempo finito. En el caso de que fuera en un tiempo infinito se obtendría un filtro IIR. Para diferenciar entre uno y otro se debe comprobar si la salida del filtro depende solamente de la entrada actual, así como de los valores de entrada pasados, caso en el que será un filtro FIR, o si también depende de los valores de salida pasados, caso en el que será un filtro IIR. Gracias a estas características los filtros FIR son siempre estables. Además, presentan ciertas propiedades en los coeficientes y suelen ser utilizados en aplicaciones de audio. Pese a esto, necesitan siempre un orden mayor respecto a los filtros IIR para cumplir las mismas características, lo que significa un mayor gasto computacional.

Debido a lo nombrado anteriormente, la expresión matemática de un filtro FIR se basa solamente en las entradas, tanto las actuales como las anteriores. Esta expresión es:

$$y_n = \sum_{k=0}^{N-1} b_k x(n-k)$$

En esta expresión:

- $N - 1$ es el orden del filtro
- N es el número de términos no nulos y el número de coeficientes del filtro.
- B_k son los coeficientes.

Esta expresión también se puede expresar como la convolución de la señal de entrada con la respuesta impulsiva. Al aplicársele la transformada Z a dicha convolución obtendremos la ecuación final:

$$H(z) = \sum_{k=0}^{N-1} h_k z^{-k} = h_0 + {h_1}^{z-1} + \cdots + {h_{N-1}}^{-(N-1)}$$

3.3.1.2. Entradas y Salidas

El bloque tendrá las siguientes entradas y salidas (figura 3.38):

Entradas:

- *clk*: reloj proveniente del bloque ralentizador.
- *reset*: se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso pondrá a cero los punteros, inicializará los estados de las máquinas y el dato previo de entrada. Está implementado como un pulsador de la FPGA.
- *x*: entrada de tipo palabra, dicha entrada se corresponde con la entrada del módulo receptor.

Salidas:

- *y*: la salida será la señal filtrada de tipo palabra.

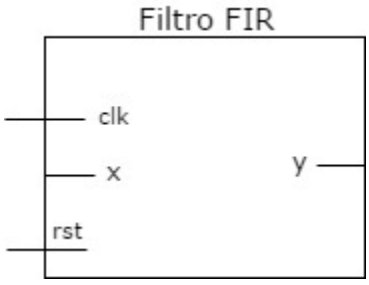


Figura 3.38. Entradas y salidas del filtro FIR.

3.3.1.3. Programación VHDL

La programación del bloque se basa en imitar la estructura básica de un filtro FIR, la cual se presenta en la figura 3.39.

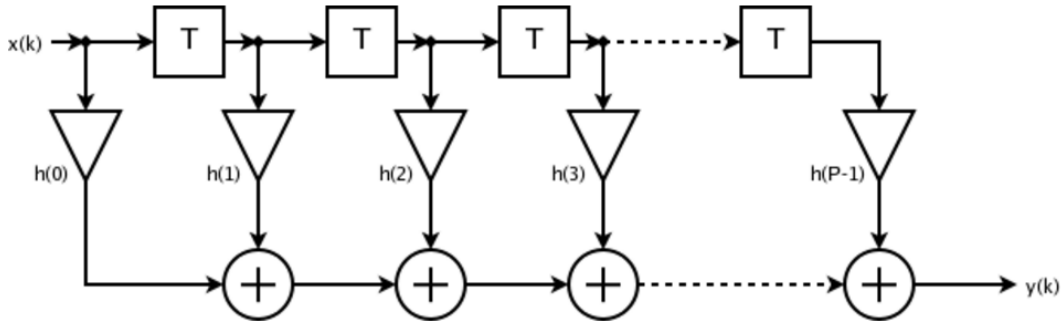


Figura 3.39. Estructura básica del filtro FIR [7].

Para la aplicación de dicha estructura han sido necesarias dos sentencias "*generate*" además de varias señales:

- **Type memoria:** definimos el tipo de una memoria, la cual será una matriz 8x12 (8 palabras digitales de 12 bits cada una). Esta memoria será necesaria para poder retrasar la señal (palabras digitales) varias veces y así realizar los cálculos con los valores de entrada previos. La definición del tipo se precisará para crear varias señales de dicho tipo.

- **Señales:** Las señales de tipo memoria, "*mem_inter*" y "*mem_entrada*" son señales que permiten el retardo de la entrada para realizar los cálculos de la expresión. En el caso de "*mem_entrada*" se retrasará, tal y como dice el nombre de la señal, la señal de entrada del filtro. La señal "*mem_inter*" almacenará los resultados de las diferentes sumas que se pueden ver en la figura 3.39 de la estructura del filtro.

```
ARCHITECTURE Behavioral OF FILTRO_FIR IS

   component Retardador is
      port (clk, rst: in std_logic;
            d: in palabra;
            q: out palabra);
   end component;

   type memoria is array (L - 1 downto 0) of palabra; -- Array (palabra, )
   signal mem_inter, mem_entrada : memoria; -- Retardos
   signal mem_salida : palabra :=(others=>'0');
```

Figura 3.40. Variables y señales del filtro FIR.

Tras definir aquellos elementos que estarán en ambos procesos, comienza la programación de las dos sentencias "*generate*" que componen el bloque:

- **Bloque Retardador:** se trata de una sentencia "*generate*" que permite la obtención de las señales de entrada retrasadas que serán necesarias para realizar las sumas de la expresión en el segundo "*generate*"

```
Bloque_Retardador: for I in 0 to L - 1 generate
   Retardador0 : if (I=0) generate
      Retardador_00 : Retardador port map (clk, rst, x, mem_entrada(0));
   end generate;
   Retardador_1_to_L-1 : if I>0 generate
      Reg : Retardador port map (clk, rst, mem_entrada(I-1), mem_entrada(I));
   end generate;
end generate;
```

Figura 3.41. Sentencia generate para la instanciación del bloque retardador.

- **Suma FIR:** en esta sentencia "*generate*" se realizarán las sumas y la división entre el coeficiente de la expresión. La suma se realiza con los términos que se extraen del bloque anterior y con otros términos previos generados por este bucle. El coeficiente de la expresión en este caso es de 1/L, siendo L = 8. Este coeficiente se ha escogido para evitar el *overflow* sin tener que aumentar los bits de entrada. La división, que se realiza sólo sobre cada nueva palabra de entrada, se ha implementado mediante el desplazamiento a la derecha de la misma una cantidad de bits *LogL*, cuyo valor es 3. Desplazar esos tres bits a la derecha ofrece el mismo resultado que al dividir entre ocho, por lo tanto, estaremos multiplicando por el coeficiente de 1/L. Tras realizar las diversas operaciones que se muestran en la figura 3.42, el último elemento de esta secuencia se vuelca sobre la salida del bloque tal y como se muestra en la última línea de código.

```
Suma_FIR: for k in 0 to L - 1 generate
   Suma_0: if (k=0) generate
      Suma_00: mem_inter(0) <= palabra(shift_right(signed(mem_entrada(0)),LogL));
   end generate;
   Suma_otra: if (k>0) generate
      sum_otra: mem_inter(k) <= palabra(signed(shift_right(signed(mem_entrada(k)),LogL))
            + signed(mem_inter(k-1)));
   end generate;
end generate Suma_FIR;
y <= mem_inter(L-1);
```

Figura 3.42. Sentencia generate para la realización de las sumas sucesivas en el filtro FIR.

3.3.1.4. Observación en el osciloscopio

Para cerciorarnos del correcto funcionamiento del bloque, observamos su comportamiento por medio de la FPGA SPARTAN-3A. Para realizar dicha comprobación se debe comparar la señal de salida del bloque, la señal filtrada, con la señal que es enviada por el transmisor. Un correcto funcionamiento del bloque significaría que la señal de salida tendrá diferentes rampas que deberán mostrar los cambios en la señal de entrada. Las señales obtenidas se pueden ver en la figura 3.43.

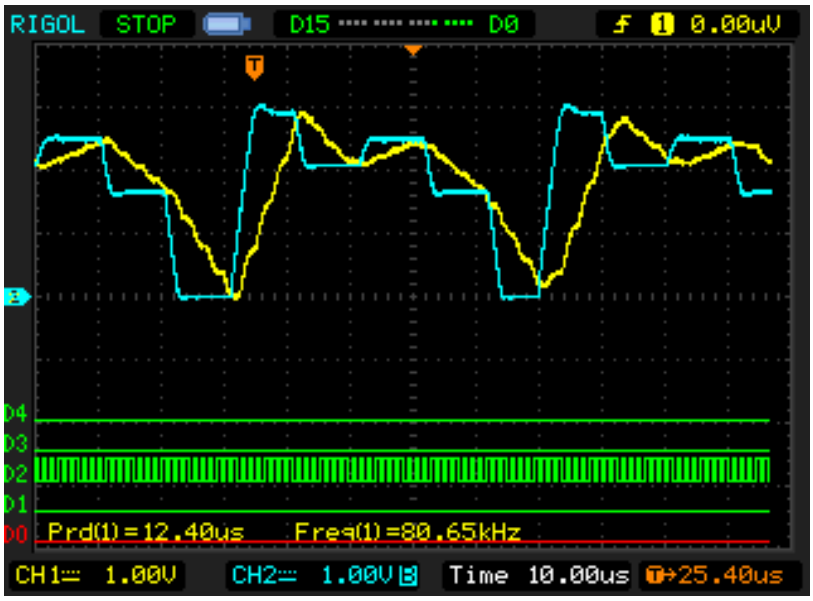


Figura 3.43. Señales obtenidas en las simulaciones del filtro FIR.

3.3.2. Bloque Derivador

3.3.2.1. Objetivo y funcionalidad

Componente cuya función es la de generar impulsos cuando existan máximos y mínimos en la señal de salida del filtro FIR, los cuales se compararán con el umbral, para crear el reloj que marcará el muestreo.

Para generar estos impulsos, se debe cumplir correctamente con la aproximación digital de la derivada. Esta aproximación se basa en la resta de la señal a derivar con la señal a derivar retardada. Además, como necesitamos los máximos y mínimos, se derivará dos (ver figura 3.44).

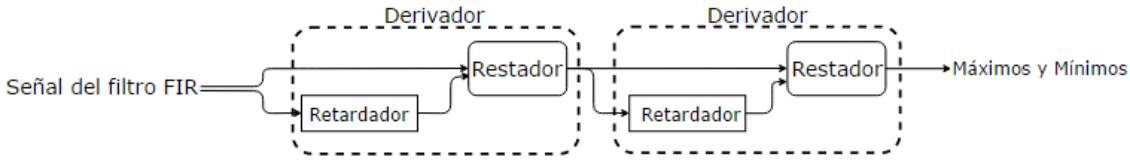


Figura 3.44. Esquema del bloque derivador.

El retardador utilizado se basará en el *flip-flop* tipo D, con la variante de que no se trata de un elemento de memoria biestable, sino que, en vez de ello, es de tipo palabra (bus de 12 bits). Estos elementos tendrán como salida del bloque la entrada, la cual se actualizará a cada flanco de reloj.

El bloque derivador se compondrá por dos derivadores instanciados que, a su vez, cada uno de ellos, estarán formados por un restador y un retardador.

3.3.2.2. Entradas y Salidas

El bloque derivador tiene las siguientes entradas y salidas (figura 3.45):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador (*clk_rec*), que actualizará la salida de los elementos de memoria.
- *rst*: Se trata de un reset cuyo objetivo es sincronizar los bloques dándoles valores iniciales, poniendo a cero la salida del registro implementado. Está implementado como un pulsador de la FPGA.
- *Derivada_in*: Se trata de la señal de entrada que se quiere derivar. En este caso queremos detectar los máximos y mínimos provenientes del filtro FIR. Esta entrada es del tipo palabra (bus de 12 bits).

Salidas:

- *Impulsos_1*: Salida que contendrá los máximos y mínimos de la señal de entrada, la cual se conectará al detector de umbral que filtrará los máximos más grandes al umbral impuesto. Esta salida es del tipo palabra (bus de 12 bits).

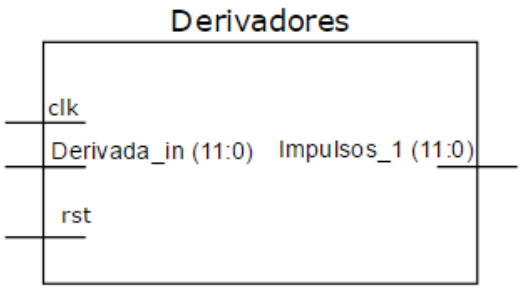


Figura 3.45. Entradas y salidas del bloque derivador.

3.3.2.3. Programación VHDL

La programación del bloque de derivadores, se basa en la instanciación de sus componentes. Primeramente, creamos el elemento retardador, tratándose de un proceso que actualiza la salida del registro en cada flanco de reloj, en el caso de que el reset no esté a '1' (figura 3.46).

```
entity Retardador is
    Port ( clk : in std_logic;
           rst : in std_logic;
           d : in palabra;
           q : out palabra);
end Retardador;


architecture Behavioral of Retardador is
   begin
      process (clk) is
      begin
         if rising_edge(clk) then
            if (rst='1') then
               q <= "000000000000";
            else
               q <= d;
            end if;
         end if;
      end process;
end Behavioral;
```

Figura 3.46. Código del retardador.

Tras crear el registro de tipo palabra, creamos el segundo elemento de la jerarquía, el derivador. En él se instanciará el retardador antes definido (figura 3.47) y se restará la señal que se conecta a la entrada del retardador con la salida del mismo (figura 3.48), por último, se enlazará la resta con la salida del derivador (figura 3.49).

```
architecture Behavioral of Derivador is

   COMPONENT Retardador
   PORT(
      clk : IN std_logic;
      rst : IN std_logic;
      d : IN palabra;
      q : OUT palabra
      );
   END COMPONENT;

   signal resta, retardo : palabra;

begin

   Inst_Retardador: Retardador PORT MAP(
      clk => clk,
      rst => rst,
      d => Entrada_derivada,
      q => retardo
   );
```

Figura 3.47. Instanciación del retardador en el derivador.

```
resta <= palabra(signed(Entrada_derivada) - signed(retardo));
```

Figura 3.48. Resta del derivador.

```
Derivada <= resta;
```

Figura 3.49. Salida del derivador.

Para finalizar creamos el bloque derivador, que se compondrá de la instanciación de los derivadores (figura 3.50), finalizando el bloque que dará la señal con los máximos y mínimos del filtro FIR.

```
architecture Behavioral of Bloque_derivador is

   COMPONENT Derivador
   PORT(
      clk : IN std_logic;
      rst : IN std_logic;
      Entrada_derivada : IN palabra;
      Derivada : OUT palabra
      );
   END COMPONENT;

   signal derivada_1 : palabra := (others=>'0');

begin

   Derivador_1: Derivador PORT MAP(
      clk => clk,
      rst => rst,
      Entrada_derivada => Derivar,
      Derivada => derivada_1
   );

   Derivador_2: Derivador PORT MAP(
      clk => clk,
      rst => rst,
      Entrada_derivada => derivada_1,
      Derivada => Derivada_2
   );

end Behavioral;
```

Figura 3.50. Código del bloque de derivadores.

3.3.2.4. Observación en el osciloscopio

El bloque derivador permite la generación de los impulsos a partir del filtro FIR, para comprobar el correcto funcionamiento se ha simulado y se ha mostrado el resultado en el osciloscopio digital RIGOL. Como podemos ver en la imagen siguiente los impulsos se corresponden con los diversos cambios (picos) que se producen en la señal del filtro.

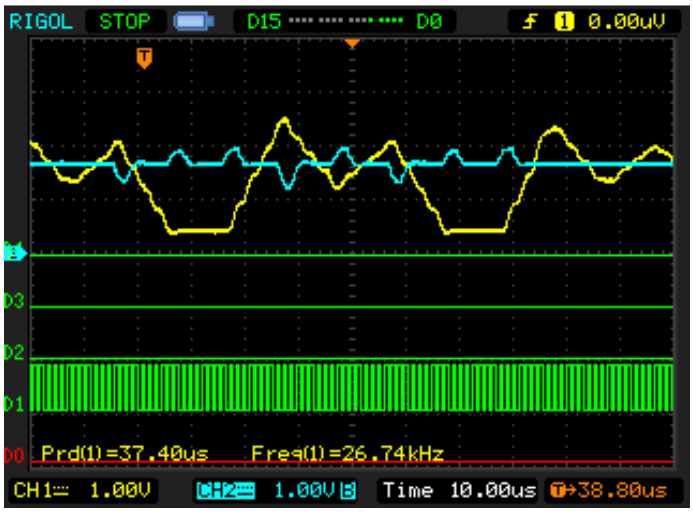


Figura 3.51. Simulaciones realizadas para la comprobación del funcionamiento de los derivadores.

3.3.3. Detector de Umbral

3.3.3.1. Objetivo y funcionalidad

El objetivo del detector de umbral es filtrar los máximos y mínimos obtenidos del derivador. Para llevar a cabo esto, primeramente, se hace el valor absoluto de la señal de entrada proveniente del derivador (para que todos los valores sean positivos). Tras ello se compara esta señal con un voltaje constante definido *(umbral)* en el paquete de constantes, quedándonos solo con aquellos máximos que superan este umbral.

Los máximos que superan este umbral son aquellos puntos de interés donde es necesario muestrear, por ello se generará un reloj de muestreo en base a ellos.

Este umbral se define con la ayuda del osciloscopio, visualizando la señal de entrada y decidiendo que máximos son los más representativos para representar óptimamente la señal recibida del transmisor (figura 3.52).

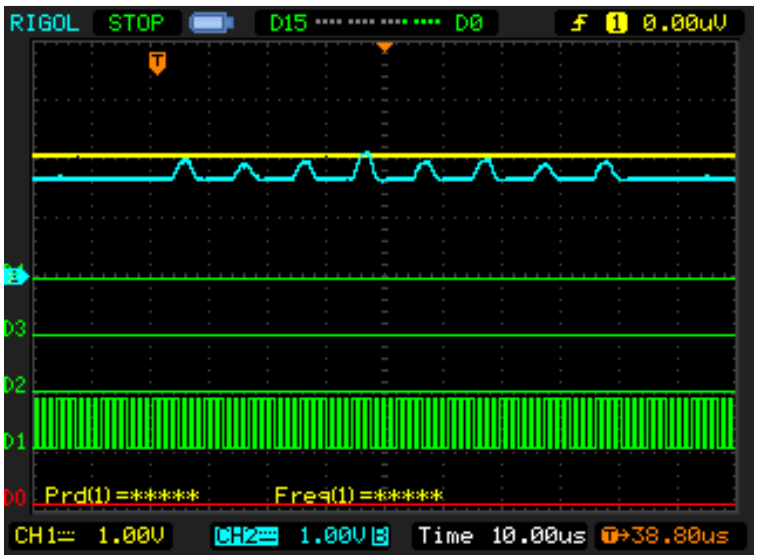


Figura 3.52. Visualización en el osciloscopio de la constante umbral y la salida de los derivadores.

Una vez precisado correctamente la constante deseada, el bloque detector umbral está correctamente configurado para detectar los máximos óptimos.

3.3.3.2. Entradas y Salidas

El módulo detector umbral tiene las siguientes entradas y salidas (figura 3.53):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador (*clk_rec*), que actualizará la salida del bloque en cada flanco.
- *rst*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, poniendo a cero la salida del bloque. Está implementado como un pulsador en la FPGA.
- *Impulso_in*: Señal de entrada proveniente del bloque derivador, la cual contiene los máximos y mínimos de la señal del filtro FIR. Esta entrada es del tipo palabra (bus de 12 bits).

Salidas:

- *Impulsos_out*: Salida que contendrá los máximos más representativos de la señal de entrada, los cuales superan el umbral determinado. Esta salida es del tipo palabra (bus de 12 bits).

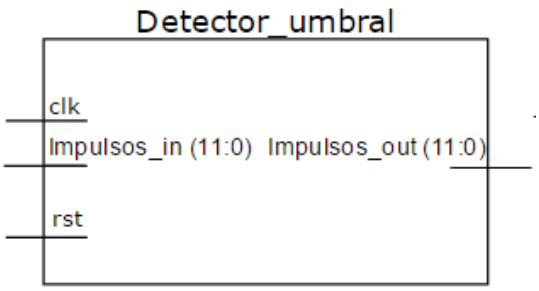


Figura 3.53. Entradas y salidas del detector umbral.

3.3.2.1. Programación VHDL

La programación del detector se basa, en un primer momento, en poner en valor absoluto la entrada proveniente del derivador (figura 3.54), para que todos los máximos recibidos sean positivos, para lo que hacemos uso de la función *abs*.

```
architecture Behavioral of Detector_umbral is

   signal Impulso_abs : palabra := (others=>'0');

begin

   Impulso_abs <= palabra(abs(signed(Impulso_in)));
```

Figura 3.3.54. Valor absoluto de la señal derivadora.

Una vez creada la señal en valor absoluto, vamos calibrando la constante umbral deseada (ver figura 3.52), la cual está definida en el paquete de constantes y cuyo valor es de 0,8 V (el rango máximo de voltaje es de 3,3 V).

Tras convertir la señal de entrada a valor absoluto y definir el umbral, creamos un proceso dependiente del reset y del reloj (figura 3.55), que compara la señal de entrada con el umbral. Si la señal de entrada es mayor que el umbral (son los máximos más representativos), la salida del bloque será igual a la entrada. En cambio, si la entrada es menor que el umbral, la salida será cero (se desechan los máximos innecesarios). Programando así el correcto funcionamiento del bloque.

```
process (clk, rst)

begin

   if rst = '1' then
      Impulso_out <= "000000000000";

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

      if (Impulso_abs > umbral) then
         Impulso_out <= Impulso_abs;
      else
         Impulso_out <= "000000000000";
      end if;

   end if;

end process;
```

Figura 3.55. Proceso de comparación del detector umbral.

3.3.3.3. Observación en el osciloscopio

Para cerciorarnos del correcto funcionamiento del bloque, observamos su comportamiento por medio de la FPGA SPARTAN-3A. Ya que gracias al ADC (conversor analógico digital), podremos visualizar las señales del detector umbral en el osciloscopio digital RIGOL. Comparando el umbral impuesto con la salida del bloque, verificando que trabaja

correctamente y sólo se queda con los máximos de mayor interés para representar la señal del transmisor (figura 3.56).

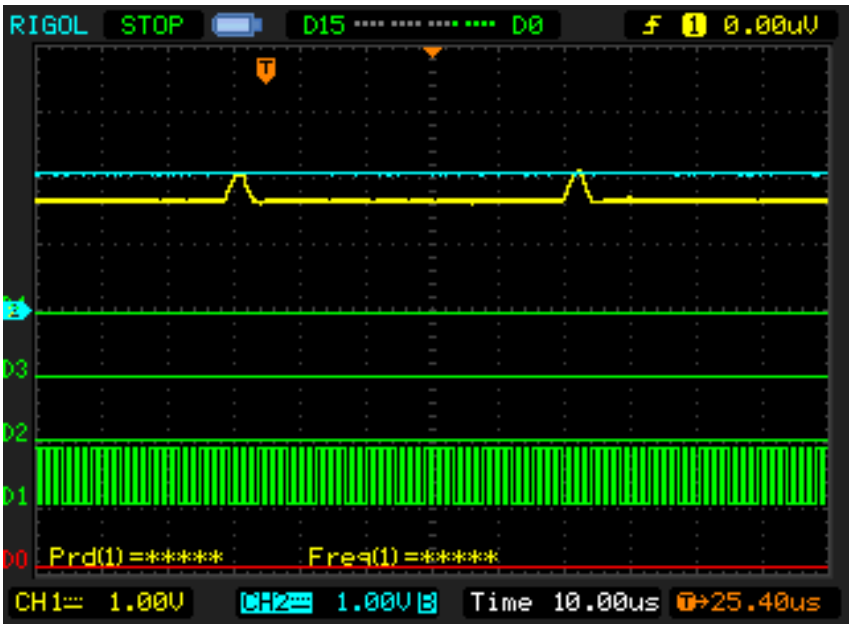


Figura 3.56. Salida del detector umbral con el constante umbral.

3.3.4. Generador de reloj

3.3.4.1. Objetivo y función

Este bloque se ocupa de la generación de la señal de reloj que servirá para el control del muestreo y la retención de la señal filtrada en el filtro FIR en el bloque del *Sample & Hold* que se describirá posteriormente. La señal de reloj generada por este bloque debe tener un periodo que le permita el muestreo de todos los cambios en la señal. Para calcular dicho período se debe tener en cuenta el número de muestras que componen cada escalón en la señal de salida del filtro FIR, que en este caso son ocho, y ralentizar ese mismo número de veces la señal de reloj del receptor. Hay que tener en cuenta que este reloj debe estar supeditado a los impulsos salientes del detector de umbral. Para ello se realiza un ajuste en la cuenta generadora del reloj cuando se detecta la llegada de un impulso. Este ajuste se basa en el reseteo de la misma al valor de tres. Este valor se ha calculado de forma experimental, ya que ofrece un perfecto muestreo según los resultados obtenidos tras varias simulaciones en el laboratorio con diferentes valores de reseteo.

3.3.4.2. Entradas y Salidas

El bloque generador de reloj tendrá varias entradas y salidas digitales, ya sean seriales o de varios bits (Figura 3.57), las cuales son:

Entradas:

- *clk*: señal de reloj del receptor generada en el ralentizador.
- *Reset*: se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. En este caso pondrá a cero las variables internas y las salidas digitales del bloque. Está implementado como un pulsador de la FPGA.
- *Sincronizador*: esta señal de entrada proviene del bloque detector de umbral, por lo que contendrá los impulsos mayores que el valor de umbral que ajustarán la cuenta generadora del reloj.

Salidas:

- *Clk_sampleo*: Como salida del bloque se obtendrá la señal de reloj que se utilizará como control del muestreo en el bloque del *Sample & Hold*. Definirá tanto el periodo de muestreo como el tiempo de retención de la señal.

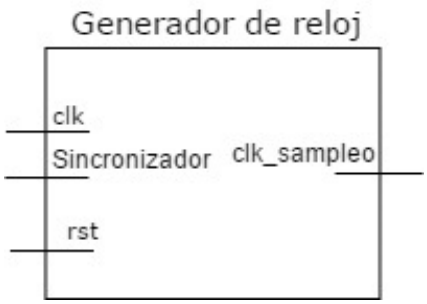


Figura 3.57. Entradas y salidas del generador de datos.

3.3.4.3. Programación VHDL

La programación del bloque generador se centra en un único proceso (dependiente del reloj generado y del reset). Dicho proceso contiene las siguientes variables:

- **Variables:** se presenta una variable en el programa, la cual es una cuenta que permite, a partir del reloj del receptor, crear un nuevo reloj de la misma forma que se realizó en el módulo ralentizador. Además, esta variable interactuará con la señal de impulsos mediante el reseteo de la misma.

El programa utilizado en este bloque es similar al utilizado en el ralentizador con la diferencia del ajuste de la cuenta en el caso de llegada de un nuevo impulso por la entrada de sincronización. En dicho momento la cuenta tomará el valor de tres, lo que permitirá un perfecto acoplamiento del muestreo a la señal emitida por el transmisor. Este ajuste se realiza a partir de una sentencia "*if*" en la cual se compara la señal de impulsos con el valor cero, cuando se presente un valor distinto de este número se reseteará la cuenta a tres.

3.3.4.4. Observación en el osciloscopio

Para la comprobación de este módulo se ha decidido visualizar al mismo tiempo los impulsos generados anteriormente y el reloj generado en el analizador de espectro, de modo que se puede ver un cambio en las cuentas realizadas cuando se produce un impulso, adelantando el siguiente pulso una serie de posiciones. Como ejemplo de este funcionamiento se muestra la siguiente imagen.

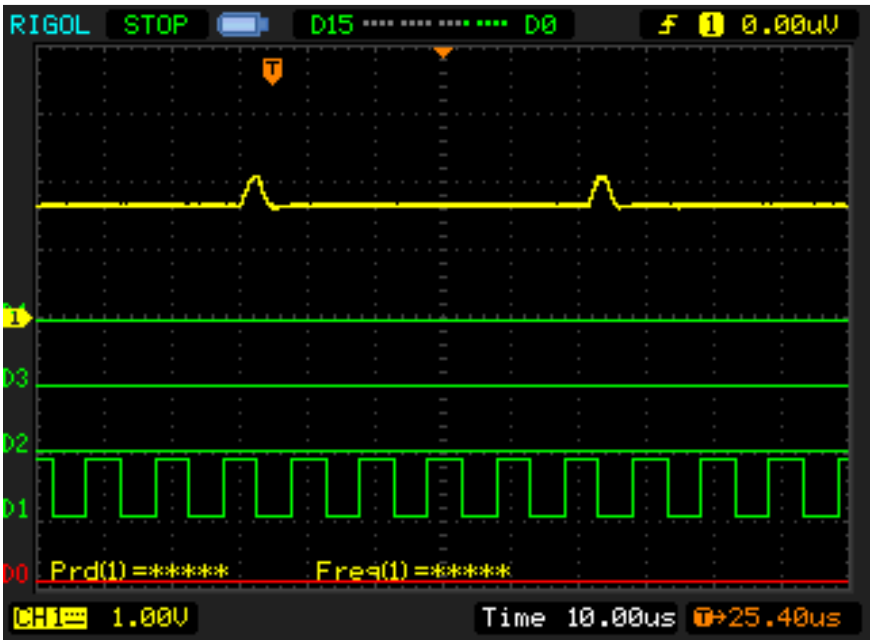


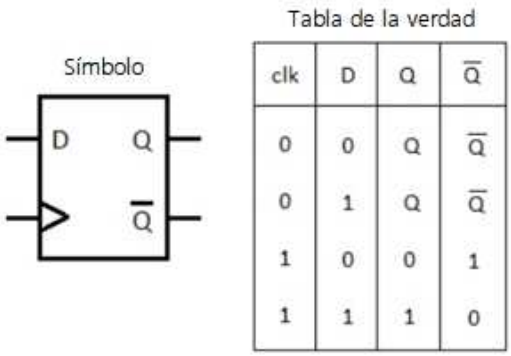
Figura 3.58. Simulación realizada sobre el reloj generado.

3.3.5. Bloque Retardador

3.3.5.1. Objetivo y funcionalidad

El objetivo de implementar un bloque retardador es obtener señales atrasadas en el tiempo. Estas son necesarias para conseguir la señal de muestreo del *Sample & Hold*, ya que debe retrasar el reloj que marca los puntos de interés a muestrear para a continuación enviarlo al *Sample & Hold*.

Estos retardadores tienen como base un flip-flop tipo D, a los que denominamos *registros* (figura 3.59), los cuales son elementos de memoria biestables, que tienen como particularidad que la salida del bloque es igual a la entrada. El disparo de la señal de salida se verá guiado por los flancos del reloj de entrada.



clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

Figura 3.59. Símbolo y tabla de la verdad del flip-flop D [8].

El bloque retardador se compone de *L* (constante definida en el paquete de constantes cuyo valor es 8) registros biestables conectados en cascada (figura 3.60), de tal manera que se retarda la señal de entrada tantas veces como número de registros estén conectados. En nuestro caso, creamos un bloque con 8 biestables conectados, pudiendo escoger como salida cualquiera de las salidas de los biestables, por lo que podemos ir variando el retardo de la señal de entrada dentro de nuestro rango.

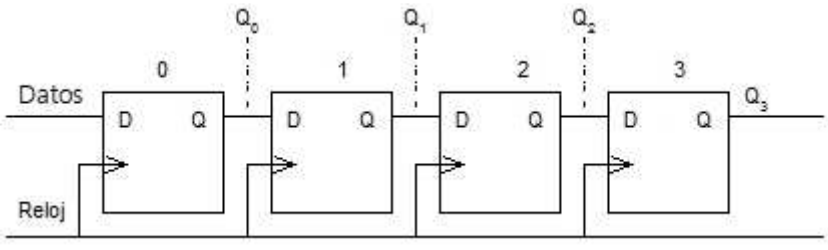


Figura 3.60. Registros conectados en cascada [9].

Además del registro biestable, hemos creado un registro cuya salida es de tipo palabra (bus de 12 bits), destinado a retrasar la salida del filtro FIR, tal como se definió en el *bloque derivador,* necesario para cumplir correctamente con la aproximación digital de la derivada de la señal de salida del filtro FIR.

3.3.5.2. Entradas y Salidas

Las entradas y salidas del bloque retardador (figura 3.61), son las siguientes:

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador (*clk_rec*), que marcará el disparo de la salida *Q*.
- *reset*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, a la salida del bloque. Está implementado como un pulsador en la FPGA.
- *D*: Entrada a retrasar, la cual es la salida del generador de reloj (*clk_sampleo*). Este reloj marca los puntos de interés a muestrear.

Salidas:

- *Q*: Salida retardada en el tiempo *L* veces. Esta se conectará a la entrada del *Sample & Hold*, y será la que marque el muestreo del mismo.

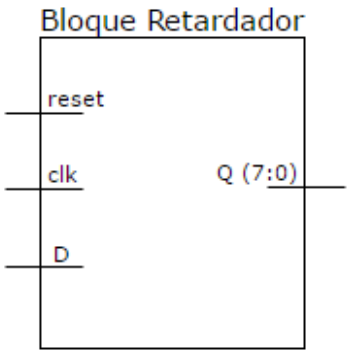


Figura 3.61. Entradas y salidas del bloque retardador.

3.3.5.3. Programación VHDL

En un primer paso, debemos crear el componente de menor jerarquía, los *registros.* Para ello hacemos uso de un proceso muy simple, ya que simplemente si hay un flanco de subida del reloj de entrada, la salida se actualizará al valor de la entrada (figura 3.62).

```
entity registro is
   Port ( clk ,preset ,D : in STD_LOGIC;
          Q : out STD_LOGIC) ;
end registro;

architecture Behavioral of registro is

begin
   process ( clk ,preset )
       begin
       if preset = '1' then
          Q <= '1';
       elsif clk'event and clk ='1' then
          Q <= D;
       end if;
   end process;
end Behavioral;
```

Figura 3.62. Código del registro.

Tras definir el registro, bastaría con conectarlos en cascada para crear el bloque retardador. Para ello utilizamos la función *loop generate*, la cual te permite instanciar *arrays* de componentes (figura 3.63), generando un bloque retardador desde *I = 0* hasta *I = L-1*, componiéndose de 8 registros instanciados entre sí.

```
--Bloque Retardador: La señal de reloj obtenida del generador de reloj se retarda tantas
--veces como sea necesario obteniendo la señal de muestreo del Sample & Hold.
   Bloque_Retardador: for I in 0 to L-1 generate
      Retardador0 : if (I=0) generate
         Retardador_00 : registro port map (clk_rec, rst, clk_sampleo,clk_sample_ret(0));
      end generate;
      Retardador_otro : if I>0 generate
         Reg : registro port map (clk_rec, rst, clk_sample_ret(I-1),clk_sample_ret(I));
      end generate;
   end generate;
```

Figura 3.63. Código del bloque retardador.

3.3.5.4. Observación en el osciloscopio

El funcionamiento de este módulo ha sido comprobado a través de la visualización de los distintos retardos que se han creado en el bloque en el analizador lógico del osciloscopio digital RIGOL. Una de las pruebas se mostrará al finalizar el párrafo, en concreto la

comparación del reloj sin retardar (señal D2, verde) con la del mismo retrasado cinco veces (señal D1, rojo).

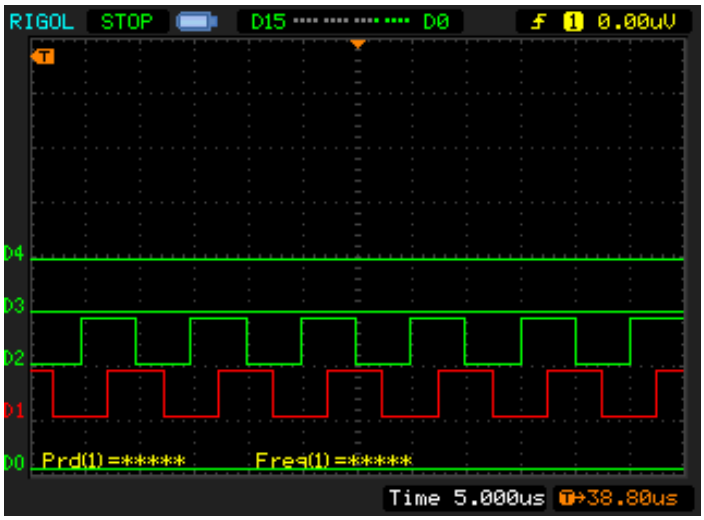


Figura 3.64. Simulación comprobadora del funcionamiento de los retardos.

3.3.6. Sample & Hold

3.3.6.1. Objetivo y función

Los circuitos de muestreo y retención, también conocidos como *Sample & Hold*, se utilizan para muestrear una señal analógica, aunque en este caso sea una señal digital de varios bits, en un instante dado y mantener el valor de muestra durante tanto tiempo como sea necesario. El muestreo de la señal y, por lo tanto, el tiempo de retención estarán determinados por una señal de control. Estos circuitos analógicos normalmente se aplican gracias al uso de condensadores formando circuitos semejantes al mostrado en la ilustración siguiente (figura 3.65).

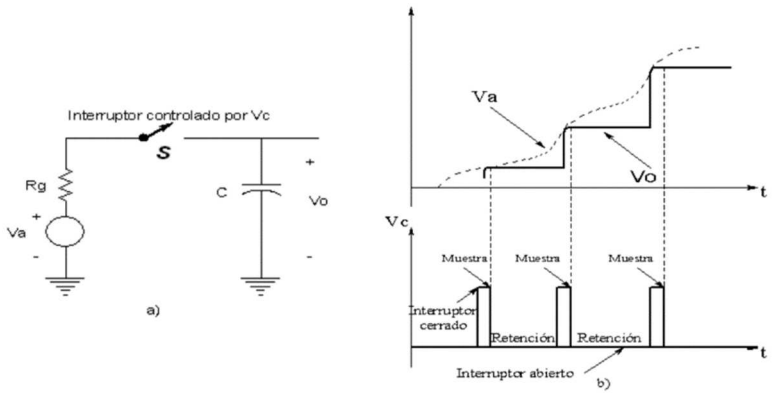


Figura 3.65. Circuitos analógicos de muestreo y retención [10].

3.3.6.2. Entradas y Salidas

El *Sample & Hold* tendrá varias entradas y salidas digitales, ya sean seriales o de varios bits (Figura 3.66), las cuales son:

Entradas:

- *Sample_Hold*: señal de reloj generada a partir de los impulsos en el generador de reloj.
- *Reset*: se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, en este caso poner a cero las variables internas y las salidas digitales del bloque. Esta implementado como un pulsador de la FPGA.
- *Vin*: esta entrada se corresponde con la palabra que proviene de la salida del filtro FIR que será muestreada por el dispositivo muestreador

Salidas:

- *Vout*: Como salida del bloque se obtendrá la señal muestreada en aquellos puntos en los que haya un flanco de subida en la señal de reloj generada.

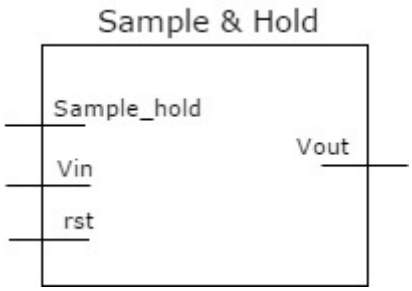


Figura 3.66. Entradas y salidas del Sample & Hold.

3.3.6.3. Programación VHDL

La programación del bloque muestreador se centra en un único proceso (dependiente del reloj generado y del reset). Dicho proceso contiene las siguientes señales y variables:

- **Variables:** se presenta una variable en el programa, la cual se ocupa de ir almacenando los distintos valores que toma la señal de entrada del muestreador en los momentos de muestreo, esto implica que este valor solamente cambiará de valor cuando se produzca un flanco de subida en la señal de reloj que controla este proceso. El contenido de esta variable se volcará al final del proceso se produzca o no el flanco de subida del reloj.

Para la aplicación de este dispositivo en VHDL se ha decidido realizar el muestreo de la señal en el momento en el que haya un flanco de subida del reloj de muestreo generado por los impulsos que se ha explicado previamente, de modo que el periodo de retención de la señal durará el tiempo que haya entre dos flancos de subida o, lo que es lo mismo, el periodo de la señal de reloj. El muestreo se almacenará en una variable temporal cuyo contenido se volcará en la salida del muestreador al final del proceso.

3.3.6.4. Simulación VHDL

Para la simulación previa se ha utilizado un archivo de pruebas (*test bench*) en el cual se ha simulado la señal de reloj generada con un reloj de diez nanosegundos de periodo, de modo que el muestreo se realizará cada dicho tiempo. Como entrada se han utilizado varios números aleatorios los cuales se muestrearán solamente en el flanco de subida del reloj y permanecerán en dicho valor hasta el siguiente flanco, independientemente de si se producen cambios en la entrada o no entre dichos instantes.

Como se puede apreciar en la figura siguiente (figura 3.67), el muestreo de la señal se realiza en los flancos de subida de la entrada "*sample_hold*", momentos en los que se puede ver cómo la salida toma el valor de la entrada del dispositivo. También se puede apreciar que, aunque se produzca un cambio en la entrada, no se producirá un cambio en la salida mientras no se produzca un flanco de subida.

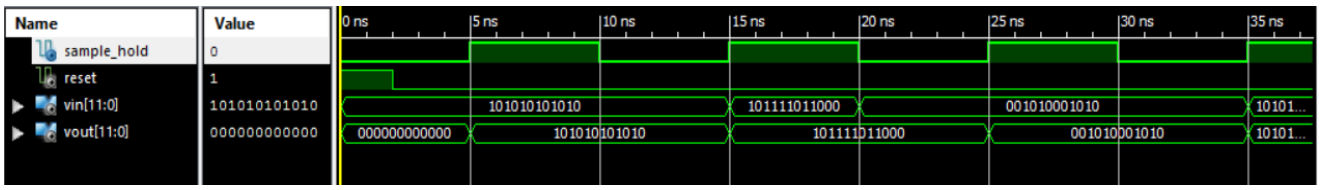


Figura 3.67. Simulación VHDL del Sample & Hold.

3.3.6.5. Observación en el osciloscopio

Por último, se realiza la observación en el laboratorio para comprobar su funcionamiento en un entorno real. Para ello comparamos la señal de salida del *Sample & Hold* con la salida del convertidor serie-paralelo, ya que la salida muestreada por este bloque debe ser semejante al mensaje transmitido por el convertidor (figura 3.68).

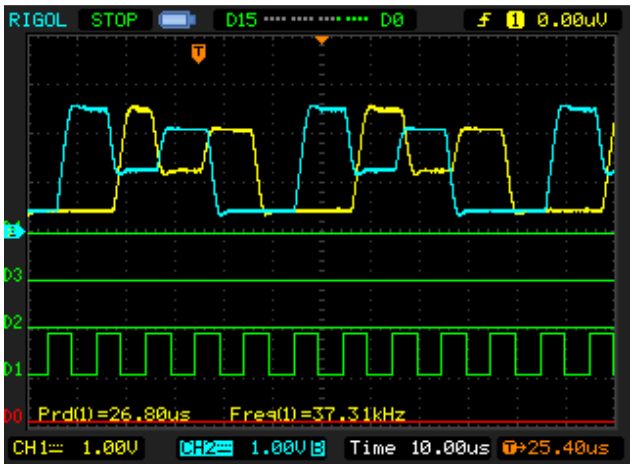


Figura 3.68. Visualización en el osciloscopio de la salida del convertidor serie-paralelo (señal amarilla) y el Sample & Hold (señal azul).

Se comprueba que la señal muestreada (señal azul), muestrea correctamente la señal transmitida por el transmisor. Esta señal se encuentra con un cierto retraso debido al procesado del filtro FIR previo al muestreo de la señal. Además, verificamos que la señal de reloj creada (*clk_sample*), escoge los puntos de mayor interés para una reproducción óptima.

3.3.7. Detector de máximos y mínimos

3.3.7.1. Objetivo y funcionalidad

Este bloque está definido para detectar los valores máximos y mínimos de voltaje de la señal de salida del filtro FIR. Una vez definidos los voltajes de fondo de escala, serán aportados al decodificador para que lleve a cabo una correcta cuantificación y decodificación de la señal.

3.3.7.2. Entradas y Salidas

El bloque detector de máximos y mínimos tendrá las siguientes entradas y salidas (figura 3.69):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador (*clk_rec*), del cual dependerán los procesos implementados en el bloque.
- *rst*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, en este caso inicializar los valores máximos y mínimos. Está implementado como un pulsador de la FPGA.
- *Vin*: Señal de entrada proveniente del filtro FIR, de la cual queremos calcular el valor máximo y mínimo. Se trata de una señal del tipo *palabra* (bus de 12 bits).

Salidas:

- *Vmax*: Valor máximo de la señal de entrada, que se irá actualizando a cada flanco de subida del *clk*, ya que puede variar. Es una salida del tipo *palabra*.
- *Vmin*: Valor mínimo de la señal de entrada, que se irá actualizando a cada flanco de subida del *clk*, ya que puede variar. Es una salida del tipo *palabra*.

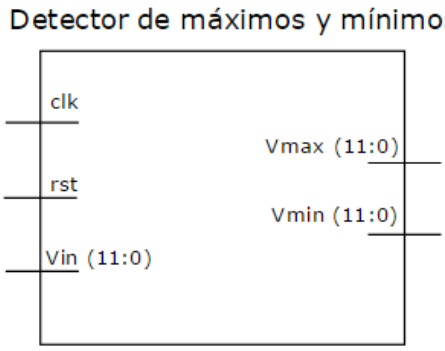


Figura 3.69. Entradas y salidas del detector de máximos y mínimos.

3.3.7.3. Programación VHDL

La programación del bloque consta de un sencillo proceso (dependiente del reloj y del reset), en el cual vamos comparando a cada flanco la seña de entrada *Vin*, con las variables que contendrán el valor máximo y mínimo.

En un principio crearemos las variables *Valor_max* y *Valor_min*, que ayudarán a crear el proceso y que posteriormente darán el valor a las señales de salida *Vmax* y *Vmin*. Estas variables las inicializaremos de manera que la variable *Valor_max* tenga el menor valor posible, es decir valga cero, y que la variable *Valor_min* tenga un valor máximo (figura 3.70). Los valores se eligen para que, una vez inicializados, tengamos la certeza de que el siguiente valor que entre en el bloque actualizará ambas variables con seguridad, ya que al poner el *Valor_max* a cero sabemos que el siguiente dato (a no ser que sea cero de nuevo), será más grande. Lo mismo pasaría con *Valor_min*, pero de forma contraria, donde el nuevo dato sería generalmente menor.

```
process(clk, rst)

   variable Valor_max: palabra := (others=>'0');
   variable Valor_min: palabra := "011111111111";
```

Figura 3.70. Variables del detector de máximos y mínimos.

Tras definir las variables entramos en el proceso, el cual compara las variables con la señal de entrada y le asigna nuevos valores a las variables (figura 3.71). Por último, las señales de salida toman el valor de las variables actualizadas.

```
   elsif (clk'event and clk = '1') then

      if (Valor_max < Vin) then
         Valor_max := Vin;
      end if;

      if (Valor_min > Vin) then
         Valor_min := Vin;
      end if;

      Vmin <= Valor_min;
      Vmax <= Valor_max;

   end if;
end process;
```

Figura 3.71. Proceso de salida del detector de máximos y mínimos.

3.3.7.4. Simulación VHDL

Hacemos uso de la simulación gracias a la herramienta *Simulate Behavioral Model*, utilizando el *test bench* de nuestro código. En dicho *test bench*, introducimos valores aleatorios de entrada (figuras 16 y 17) y comprobamos cómo se van actualizando los valores máximos y mínimos según entran nuevos valores más pequeños o más grandes que los que se encuentran guardados en las variables internas.

```
clk <= not clk after 5 ns;
rst <= '1', '0' after 2 ns;
Vin <= "000000000101", "000011111110" after 15 ns, "000000000001" after 25 ns, "001111100000" after 35 ns;
```

Figura 3.72.Valores aleatorios de entrada a la simulación. del detector de máximos y mínimos.

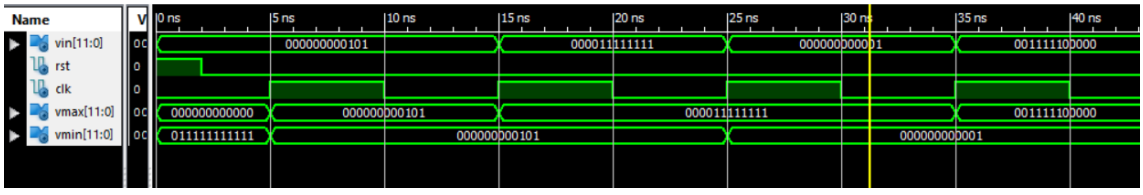


Figura 3.73. Simulación VHDL de bloque detector de máximos y mínimos.

3.3.8. Decodificador

3.3.8.1. Objetivo y funcionalidad

El objetivo principal del decodificador es realizar la conversión de la señal muestreada recibida del *Sample & Hold* (tipo *palabra*) a una señal del tipo *Nbits* (bus de 4 bits) enviada por el emisor. Para ello, se realiza en un primer lugar una cuantificación de la señal (una representación de la señal con un número finito de niveles de amplitud).

Para lograr la cuantificación, se genera una escala dependiente de los valores máximos y mínimos de la señal de entrada, que tendrá una resolución de 2^{Nbits} (16 niveles de amplitud) y que se define a través de los parámetros a continuación:

- Intervalo de cuantificación (*q*): representa la diferencia entre el mayor y menor nivel de entrada a los que se le asigna el mismo nivel de salida. Se calcula dividiendo el margen de entrada *(M)* entre los niveles de decisión:

$$q = \frac{M}{Nº\ de\ niveles} = \frac{Vmax - Vmin}{2^{Nbits} - 1}$$

  El voltaje dado en cada intervalo *i*-ésimo, sigue la siguiente expresión que irá desde $i = 0$ hasta $i = 2^{Nbits} - 2$ (número de intervalos de decisión de tamaño *q*):

$$Vq(i) = Vmin + \frac{q}{2} + q\ (i - 1)$$

- Niveles de decisión: niveles en los que se divide la escala, encontrándose dividida en $2^{Nbits} - 1$ niveles.
- Intervalos de decisión de tamaño *q*, los cuales definen la distancia entre los sucesivos niveles de decisión. Se dividen en $2^{Nbits} - 2$ intervalos.

Una vez cuantificada la señal, se decodifica. Para lograrlo, se compara el voltaje de entrada con el voltaje en cada intervalo de cuantificación, escogiendo como salida el número del

intervalo de decisión superior al último voltaje de intervalo cuyo valor es menor que el voltaje de entrada.

3.3.8.2. Entradas y Salidas

El decodificador tiene las siguientes entradas y salidas (figura 3.74):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador (*clk_rec*), del cual dependerán los procesos implementados en el bloque.
- *rst*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, poniendo a cero la salida del bloque y las variables cuantificadas. Está implementado como un pulsador de la FPGA.
- *Vin*: Se trata de la señal de entrada que se quiere convertir de tipo *palabra* a tipo *Nbits*. Esta entrada es del tipo *palabra* (bus de 12 bits).
- *Vmax*: Valor máximo de *Vin*. Este valor provendrá del detector de máximos y mínimos y se actualizará a cada flanco de reloj. Servirá para calcular el intervalo de cuantificación y el voltaje en cada intervalo, definiendo así el cuantificador. Es de tipo *palabra* (bus de 12 bits).
- *Vmax*: Valor mínimo de *Vin*. Este valor provendrá del detector de máximos y mínimos y se actualizará a cada flanco de reloj. Servirá para calcular el intervalo de cuantificación y el voltaje en cada intervalo, definiendo así el cuantificador. Es de tipo *palabra* (bus de 12 bits).

Salidas:

- Vout: Salida que contendrá la señal que fue transmitida por el emisor, la cual será de *Nbits* (bus de 4 bits).

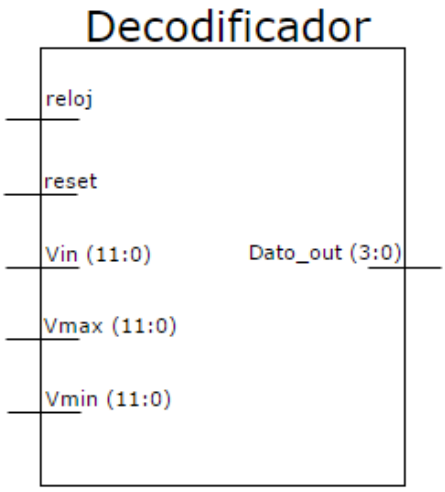


Figura 3.74. Entradas y salidas del decodificador.

3.3.8.3. Programación VHDL

Para programar el decodificador, primeramente, hay que definir el valor del intervalo de cuantificación y los voltajes en cada intervalo. Por ello, creamos un *array* que contendrá todos los valores de voltaje en los intervalos, además de crear e inicializar la señal *q* (figura 3.75).

```
architecture Behavioral of DECODIFICADOR is


   type Vector_q is array (2**N -2 downto 0) of palabra;
   signal Vqi: Vector_q;
   signal q: std_logic_vector (nbits-1 downto 0):= "000000000000";

begin
```

Figura 3.75. Señales del decodificador.

Una vez definidas las señales anteriores, creamos una máquina dependiente de los valores máximo y mínimo de entrada, además del reset. En ella definimos el cuantificador, donde calculamos los voltajes de cada intervalo y el intervalo de cuantificación según las fórmulas expresadas en el anterior apartado. Estas irán cambiando de valor, según vayan cambiando los valores provenientes del detector de máximos y mínimos (figura 3.76).

```
process (Vmin,Vmax,reset)
variable vq : Vector_q;
begin
      q <= std_logic_vector(signed(signed(Vmax) - signed(Vmin))/(to_signed((2**nbits) - 1, nbits)));

      vq(0) := palabra(signed(Vmin) + signed(shift_right(signed(q),1)));
      for cuenta in 1 to ((2**N)-2) LOOP
         vq(cuenta) := palabra(signed(vq(cuenta-1))+signed(q));
      end LOOP;
      Vqi <= vq;
end process;
```

Figura 3.76. Proceso del cuantificador del decodificador.

Tras determinar los voltajes de decisión, hacemos el proceso de decodificación, que compara estos voltajes con el voltaje de entrada para determinar el nivel de decisión al cual pertenece (dependiente del reloj y del reset). Nos ayudamos de la variable "*valor*" que dará el valor a la salida, y que será el nivel superior al marcado por el iterador *cuenta*. Si el reset está activo la salida y la variable se pondrán a cero, si no, y hay un flanco de subida en el reloj, se procederá a realizar la comparación entre voltajes. Por último, convertimos la salida en un bus de 4 bits que valdrá *valor* (figura 3.77).

```
process (clk, reset)

   variable cuenta, valor : natural range 0 to (2**N)-1 := 0;
begin
   if (reset = '1') then
         Dato_out <= "0000";
         valor := 0;

   elsif (clk'event and clk = '1') then
      valor := 0;
      for cuenta in 0 to ((2**N)-2) LOOP
         if signed(Vin) > signed(Vqi(cuenta)) then
            valor := cuenta + 1;
         end if;
      end LOOP;

      Dato_out <= std_logic_vector (to_signed(valor,4));
   end if;
end process;
```

Figura 3.77. Proceso del cuantificador del decodificador.

3.3.8.4. Simulación VHDL

El funcionamiento del decodificador se ha simulado gracias a la herramienta *ISim* del paquete ISE. En la simulación se han tomado diferentes valores aleatorios de entrada "*Vin*" además de un reset y el reloj de entrada. Los valores que se obtienen a la salida son el dato que saldría del decodificador y la variable *q*, la cual se muestra para asegurar que se realiza de forma

correcta la decodificación por niveles de tensión. En las figuras 3.78 y 3.79 se muestran las entradas introducidas y los resultados de la simulación.

```
clk <= not clk after 5 ns;
reset <= '1' after 0 ns, '0' after 2 ns;
Vin <= "000000000001", "0000111111110" after 15 ns, "001111100000" after 25 ns;
```

Figura 3.78. Test bench del decodificador.

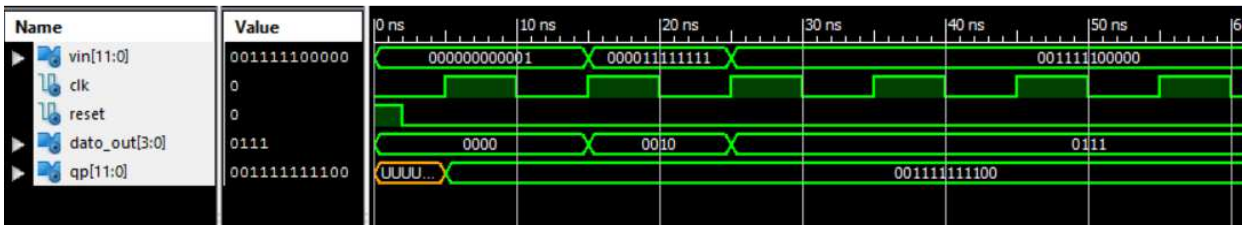


Figura 3.79. Simulación del funcionamiento del decodificador.

3.3.9. Bloque eliminador de relleno

3.3.9.1. Objetivo y funcionalidad

Este módulo se ha creado con el motivo de la eliminación del relleno creado en el bloque creado en el transmisor. Esto se realiza para recibir los datos de forma correcta, dejando vacío en los momentos los cuales no se envían datos, aunque el canal siga abierto (el led se mantenga encendido).

Este módulo se encuentra al final del receptor, tras el decodificador, cuando los datos se encuentran en buses de cuatro bits, lo que permite eliminar los datos “F” y “0” rellenados.

Al igual que el módulo de relleno, este bloque requiere de una máquina de estados y de un buffer circular en el cual, tras eliminar los buses “F” y “0” relativos al relleno, se escribirán los datos restantes en el buffer, del cual serán leídos por una segunda máquina de estado que a su vez los volcará en la salida del bloque. Tanto el buffer como ambas máquinas de estado se encuentras controlados por el reloj del receptor. Estas dos máquinas se explicarán a continuación:

- **Máquina de entrada:** esta máquina se ocupa de la eliminación de los datos de relleno y luego de escribir los datos restantes en el buffer cíclico. Para la correcta eliminación del relleno, evitando la eliminación de datos “F” y/o “0” no correspondientes al

relleno, se ha decidido establecer un control por marcas que será explicado durante la ilustración del código del bloque.

- **Máquina de salida:** esta sencilla máquina se ocupa únicamente de volcar la información que se encuentra en el buffer en la salida del bloque. Este paso de información se realizará en cada flanco de subida del reloj del receptor.

3.3.9.2. Entradas y Salidas

El bloque tendrá las siguientes entradas y salidas (figura 3.80):

Entradas:

- *clk*: Reloj proveniente del bloque ralentizador, que marcará la llegada de datos y su correspondiente transmisión.
- *reset*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales, en este caso poner a cero los punteros, inicializar los estados de las máquinas y el dato previo de entrada. Está implementado como un pulsador de la FPGA.
- *Dato_in*: Entrada de tipo bus de 4 bits, que recibirá los datos de entrada provenientes del decodificador.

Salidas:

- *Dato_out*: Salida de tipo bus de 4 bits correspondiente a los datos de salida del receptor.

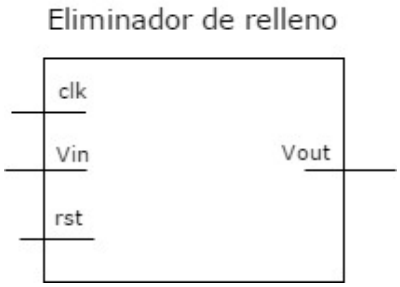


Figura 3.80. Entradas y salidas del bloque eliminador de relleno.

3.3.9.3. Programación VHDL

La programación se divide en dos bloques, la máquina de entrada y la máquina de salida (explicadas anteriormente), ambas dependientes del reloj y del reset. Además de ellas, existen variables y señales que se utilizan en ambos procesos, las cuales son (figura 3.81):

- **Type pila:** Definimos el tipo de nuestro buffer de memoria, el cual será una matriz 8x4 (ocho posiciones con palabras de datos de 4 bits). Lo necesitamos para poder asignarle el tipo a una señal.
- **Shared Variables:** Creamos los iteradores que recorrerán el buffer de memoria. Lo hacemos de tipo *shared*, para que se puedan declarar fuera de los procesos.
- **Señales:**
  - La señal *dato_previo*, se conectará con el dato previo de entrada, ya que la máquina de entrada compara el dato previo con el dato que actualmente le llega, para saber si se trata de un "F" y un "0" consecutivo.
  - *Buffer_datos* es la señal que será de tipo pila, donde se almacenarán los datos para posteriormente transmitirlos.

```
architecture Behavioral of Eliminado_Relleno_3 is

   signal dato_previo : std_logic_vector (3 downto 0);
   Shared Variable puntero_in : integer := 0; -- iterador puntero entrada
   Shared Variable puntero_out : integer := 0; -- iterador puntero salida

   type pila is array   (7 downto 0) of STD_LOGIC_VECTOR (3 downto 0); -- Matriz 8x4
   signal buffer_datos : pila;
```

Figura 3.81. Shared Variables y señales del eliminador de relleno.

Tras definir aquellos elementos que estarán en ambos procesos, comienza la programación de los dos bloques en los que se define el bloque:

- **Máquina de entrada:** Se trata de un proceso dependiente del reloj *clk* y del reset. Este proceso comprueba si la combinación del dato previo (que se va actualizando en cada flanco de bajada del reloj, tras concluir el proceso) y el dato de entrada actual se corresponden con la secuencia de relleno, es decir, se comprueba si el dato previo es "1111" y el dato actual "0000".

  Si esta comprobación es afirmativa los datos que se están analizando actualmente serían de relleno, por lo que tendrían que ser eliminados. Para ello utilizamos una

rutina compuesta de dos *if*. El primer "*if*" sería el de la comparación que se acaba de mencionar, la cual eliminaría el dato previo actual, el dato "F" del relleno. Esta eliminación se aplicaría no volcando el dato en el buffer. Tras esto se introduciría el dato actual, el dato "0" del relleno, en el dato previo y se activaría la marca del ciclo de eliminación del relleno, la cual conduce al segundo "*if*". En este "*if*", al cual se entraría en el siguiente flanco de subida en el caso de que estuviera activada la marca mencionada, se eliminaría el dato previo actual, el dato "0" del relleno. Esta eliminación se aplicaría no volcando el dato en el buffer. Tras esto se introduciría el dato actual en el dato previo y se desactivaría la marca del ciclo de eliminación del relleno, lo que nos llevaría a una nueva comparación de los datos de entrada actual y previo.

En el caso de que la comparación no sea afirmativa y que la marca de relleno no se encuentre activada se volcará el dato previo en el buffer, moviendo el puntero de entrada a la siguiente posición y actualizando el dato previo.

Su código es el siguiente:

```
ENTRADA: process (clk, reset)
variable marca : std_logic:= '0';
   begin

      if (reset = '1') then
         puntero_in := 0;
         dato_previo <= "0000";

      --Si hay un flanco de subida:
      elsif (clk'event and clk = '1') then
         --Si el dato previo es una "F" y el dato actual es "0":
         if (dato_previo = "1111" and dato_in = "0000") then
            --Si esto ocurre se introduce el dato actual en la variable de dato previo, pero no se introduce
            --nada en el buffer, pues si se ha cumplido esto significa que es relleno y por lo tanto se debe
            --eliminar. Se eliminaría la primera parte del relleno, es decir, el dato que se encuentra guardado
            --en el dato previo, el dato "1111" ("F").
            dato_previo <= dato_in;
            --Además se activa la marca de eliminación de relleno.
            marca := '1';
         --Si no se cumple lo anterior y no se encuentra la marca de eliminación de relleno activada.
         elsif (marca = '0') then
            --En este caso, al no estar activada la marca de eliminación del relleno, se introduce el dato
            --anterior en el buffer, de este modo queda almacenada la información que no pertenece al relleno.
            buffer_datos(puntero_in) <= dato_previo;
            --Tras ello movemos la posición del puntero a la posición posterior.
            puntero_in := (puntero_in + 1) mod dim_pila;
            --Por último, se introduce el dato actual en la variable del dato previo.
            dato_previo <= dato_in;
         --Si no se produce ninguno de los casos anteriores, pero se encuentra activada la marca de eliminación
         --de relleno.
         elsif (marca = '1') then
         --Se introduce el dato actual en la variable de dato previo, pero no se introduce nada en el buffer,
         --ya que el dato previo se correspondería con la segunda parte del relleno, el dato "0000" ("0") y por
         --lo tanto se debe eliminar.
            dato_previo <= dato_in;
         --Tras ello se resetea la marca, esto haría que se volviera a analizar los datos posteriores.
            marca := '0';
         end if;
      end if;

end process;
```

Figura 3.82. Código de la máquina de entrada del eliminador de relleno.

- **Máquina de salida:** esta sencilla máquina nos permite leer durante cada flanco de subida del reloj el dato del buffer al que apunta el puntero asociado a la salida y tras ello volcar dicho dato en la salida del bloque.

  Para ello simplemente se crea un proceso que dependa del reset y del reloj. Este proceso realizará lo descrito en el párrafo anterior a partir de un "*if*" que se activará en cada flanco de subida del reloj, leyendo el dato al que apunta el puntero, volcándolo en la salida y por último moviendo el puntero a la siguiente posición del buffer de datos.

```
WORK: process (clk, reset)

   begin

      if (reset = '1') then
         puntero_out := 0;

      --Si hay un flanco de subida:
      elsif (clk'event and clk = '1') then
         --Se lee el valor del dato que se encuentra en la posición del puntero y se vuelca en la salida del bloque
         Dato_out <= buffer_datos(puntero_out);
         --Tras ello movemos la posición del puntero a la posición posterior.
         puntero_out := (puntero_out + 1) mod dim_pila;
      end if;
end process;
```

Figura 3.83. Código de la máquina de salida del eliminador de relleno.

El módulo de relleno se encontrará programado, definiendo las máquinas anteriormente citadas.

3.3.9.4. Simulación VHDL

Hacemos uso de la simulación gracias a la herramienta *Simulate Behavioral Model*, utilizando el *test bench* de nuestro código. En dicho *test bench* (figura 3.84), introducimos valores aleatorios de entrada y comprobamos cómo al introducir un conjunto de datos de relleno "F0" se elimina de forma correcta.

```
clk <= not clk after 5 ns;
reset <= '1' after 0 ns, '0' after 2 ns;
Dato_in <= "1100" after 0 ns, "1000" after 25 ns, "1010" after 35 ns, "1100" after 45 ns, "1000" after 55 ns,
"1111" after 65 ns, "1111" after 75 ns, "1010" after 85 ns, "1111" after 95 ns, "0000" after 105 ns,
"1100" after 115 ns, "1000" after 125 ns;
```

Figura 3.84. Test bench del módulo eliminador de relleno.

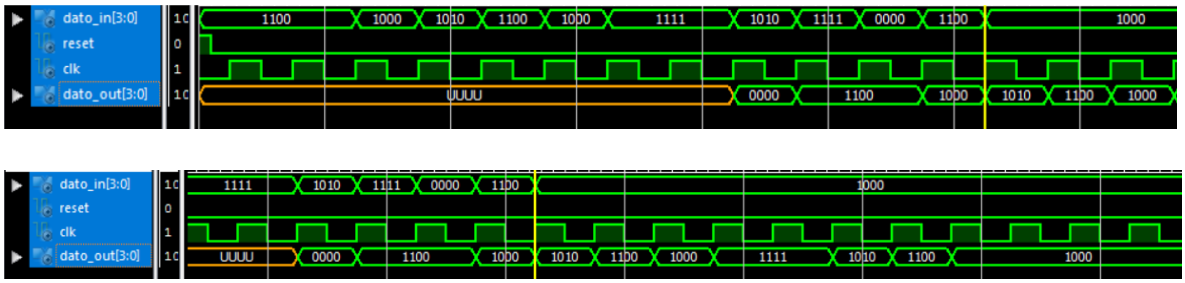


Figura 3.85. Simulaciones del módulo eliminador de relleno.

3.3.10. Instanciación de los módulos en el bloque receptor.

La instanciación de los módulos se realiza de acuerdo al diagrama de bloques que se muestra en la figura 3.36 presentada antes de comenzar la explicación de cada uno de los módulos del bloque receptor. Dicha instanciación se muestra en los anexos del código mostrados tras el último capítulo.

3.4. Interfaz de comunicación con el DAC de la tarjeta.

3.4.1. Objetivo y funcionalidad

El convertidor digital-analógico (DAC), el cual está implementado en la FPGA SPARTAN-3A (figura 3.86), tiene como objetivo ayudarnos a visualizar las señales de nuestro sistema, convirtiendo las señales digitales a analógicas (ofrece un nivel analógico de tensión de salida a partir de una palabra digital de entrada). Las cuales, una vez convertidas, se guiarán desde la salida de los canales del DAC hacia el osciloscopio digital RIGOL, donde las representamos.

Figura 3.86. DAC implementado en la FPGA. [Xilinx07].

Dicho convertidor contendrá cuatro canales de salida, las cuales son compatibles con el bus síncrono *full-duplex* SPI (*Serial Peripheral Interface*). Este bus es la línea de comunicación entre la FPGA (*master*) y el DAC (*slave*) (figura 3.87).

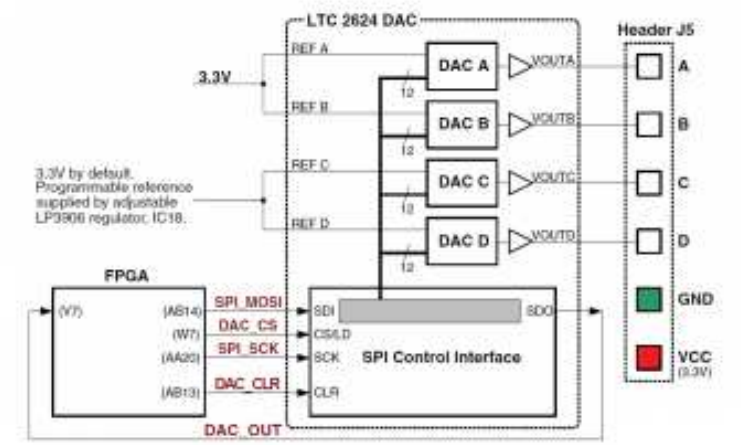


Figura 3.87. Esquema de conexiones del DAC. [Xilinx07].

El protocolo de comunicación del bus (figura 3.88), contendrá los datos a cargar en el DAC (desde el bit 15 hasta el 4), la dirección del canal a utilizar (desde el bit 19 al 16) y el comando, que generalmente será la palabra $c_3c_2c_1c_0$ = "0011", para indicar que la salida del DAC se actualice inmediatamente con la palabra recibida (desde el bit 23 hasta el 20).

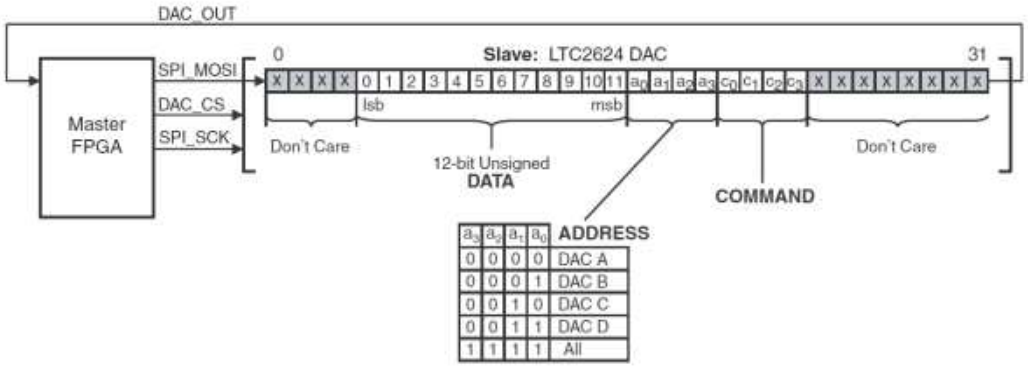


Figura 3.88. Protocolo de comunicación con el DAC. [Xilinx07.

El valor de salida del DAC seleccionado por la dirección (*ADDRESS*) vendrá dado por la siguiente expresión en función de la palabra de datos (DATA):

$$V_{OUT} = \frac{DATA[11:0]}{4\,096} V_{REF}$$

3.4.2. Entradas y Salidas

La interfaz del DAC tendrá las siguientes entradas y salidas (figura 3.89):

Entradas:

- *reloj*: Reloj proveniente del bloque ralentizador (*clk_out*), del cual dependerán los procesos implementados en el bloque.
- *reset*: Se trata de un reset, cuyo objetivo es sincronizar los bloques dándoles valores iniciales. Está implementado como un pulsador de la FPGA.
- *datos*: Palabra digital que se quiere convertir a analógica (bus de 12 bits), esta comprenderá desde el bit 15 hasta el 4. En la práctica hemos implementado dos entradas de datos, una para cada canal del osciloscopio.

Salidas: Se corresponden con las líneas de comunicación con el bus SPI.

- *spi_mosi*: Se trata del protocolo de comunicación del bus, el cual incluye datos, dirección y comando. Se le irá comunicando cada bit del protocolo dependiendo de la variable cuenta que se implementa.
- *spi_sck*: Reloj que comunica la FPGA con el DAC, el cual a cada flanco de subida de la señal *SPI_SCK*, el dato enviado por la línea *SPI_MOSI* es cargado en la memoria de entrada del dispositivo.
- *dac_cs*: Será la señal de salida que active la conversión cuando los 32 bits del protocolo se hayan transmitido. Esta conversión se hará cuando la señal pase a alta.
- *dac_clr*: Señal activa a baja para el reset asíncrono.

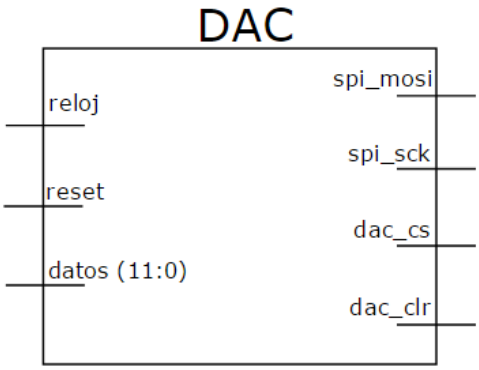


Figura 3.89. Entradas y salidas del DAC.

3.4.3. Programación VHDL

La programación del bloque se compondrá de dos máquinas de estados. La primera se tratará de una máquina Moore (su salida únicamente depende del estado), donde implementamos dos estados, *canal_A* y *canal_B*. Está máquina hará la conmutación entre canales para poder visualizar dos señales a la vez en el osciloscopio, por ello dependiendo del estado, se escogerá los datos y la dirección determinada para cada caso (figura 3.90).

```
WORK: process (reloj, reset)

type state_type is (canal_A, canal_B);
Variable state, next_state : state_type;

   begin

      if (reset = '1') then

         state := canal_A ;  -- Asignación del estado inicial
         next_state := canal_A;  -- Asignación del estado inicial

      elsif (reloj'event and reloj = '0') then

         if (state = canal_A) then
            address <= "0000";
            datos <= datos_A;
         end if;

         if (state = canal_B) then
            address <= "0001";
            datos <= datos_B;
         end if;
```

Figura 3.90. Proceso de cálculo de salida de la máquina Moore del CAD.

Para definir totalmente esta máquina, se debe determinar el paso de un estado a otro, este paso depende de la cuenta que marca la transmisión total del protocolo. Por lo tanto, cuando se cumpla la transmisión total (*cuenta* = 63) se pasará al otro estado (figura 3.91).

```
         case (state) is

               when canal_A =>
                  if (cuenta = 63) then
                     next_state := canal_B;
                  end if;

               when canal_B =>
                  if (cuenta = 63) then
                     next_state := canal_A;
                  end if;
         end case;

         state := next_state;

      end if;
end process;
```

Figura 3.91. Cambio de estados de la máquina Moore del CAD.

Además, se compone de una segunda máquina de estados, la cual se encarga de la transmisión del protocolo controlado por la variable *cuenta* (figura 3.92). El proceso depende del reloj del reset, produciéndose un incremento de la variable por cada flanco del reloj, si la cuenta es igual a uno, se cargan los datos en la señal *memoria_dac* (señal que contendrá el protocolo del bus que irá cargando su contenido en la salida *spi_mosi*) y se prepara para la transmisión de los mismos asignándoles la dirección del canal del DAC.

Tras emitir los 32 bits del protocolo, la señal *dac_cs* se activa a alta, para que el CAD lleve a cabo la conversión. Una vez transmitidos, se resetea la cuenta para que el proceso se vuelva a repetir (además en ese punto se hace el intercambio de estados de la primera máquina de estados). Las señales *spi_sck* y *dac_clr* se obtienen mediante la inversión de la señal de reloj y de reset, respectivamente.

```
   process (reloj ,reset)
   begin
      if reset = '1' then
         --Establece  el comando por defecto
         memoria_dac(23 downto 20) <= "0011";
         --Direcciona al DAC A por defecto
         memoria_dac(19 downto 16) <= address;
         cuenta := 0;
         dac_cs <= '1';
      elsif reloj 'event and reloj = '1' then
         cuenta := cuenta + 1;
         case cuenta is
            when 1 => dac_cs <= '0';
               --Carga los datos
               memoria_dac(15 downto 4) <= std_logic_vector(signed(datos));
               --Direcciona al DAC A
               memoria_dac(19 downto 16) <= address;
               spi_mosi <= memoria_dac (31);

            when 33 => dac_cs <= '1';

            when 64 => cuenta:= 0;

            when others =>
               spi_mosi <= memoria_dac (31-((cuenta -1) mod 32));
         end case;
      end if;
   end process;
   spi_sck <= not(reloj);
   dac_clr <= not(reset);
end Behavioral;
```

Figura 3.92. Proceso de transmisión del bus SPI.

La configuración del CAD quedará definida tras crear las dos máquinas de estados anteriormente citadas.

Capítulo IV:

Circuitería y funcionamiento general del prototipo.

4. Capítulo IV. Circuitería y funcionamiento general del prototipo.

4.1. Introducción

A lo largo de este capítulo, se desarrollarán las competencias asociadas al diseño de los circuitos impresos implementados con el fin de obtener el mejor enlace óptico. Se especificará qué componentes albergan, que son dichos elementos y cómo funcionan en conjunto para lograr el objetivo de cada PCB. Además, se explicará el conexionado de los circuitos, donde relataremos a qué puertos de la FPGA se conectan y el tipo de conexión.

Como conclusión del capítulo, se introducirá el funcionamiento general del prototipo, explicando cómo se comporta nuestro enlace óptico asociando todos los elementos del canal de transmisión.

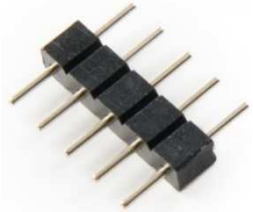
4.2. Conexiones de los circuitos impresos

En este apartado se dará la información relativa a los conectores instalados en las distintas PCB, donde especificaremos qué tipos de conectores son, a qué se conectarán y cuál es su función.

Los tipos de conectores serán los siguientes:

Bornier de 2 pines

SMA



Tira de pines

Figura 4.1. Conectores utilizados durante la fabricación de las PCB.

En primer lugar, el circuito impreso emisor contará con dos conectores. Un conector de tipo Bornier para la alimentación del circuito y una tira de 5 pines destinada a recibir la señal a emitir. La tira de pines se acoplará al conector de expansión J18 de la Spartan-3AN Starter Kit (explicado en el apartado 2.2.3), donde recogerá la información deseada para emitir.

En segundo lugar, la PCB receptora dispondrá de tres conectores. Dos conectores de tipo Bornier, uno para la alimentación a 5V y otro para permitir el acoplamiento del fotodiodo al circuito. Además, contará con un conector hembra del tipo SMA, para transmitir la señal de salida del circuito que se enlazará con el circuito del ADC.

En tercer lugar, el circuito impreso que contiene el ADC (convertidor analógico a digital) albergará dos tipos de conectores: un conector hembra SMA para recibir la señal enviada desde la PCB receptora, la cual se convertirá de analógica a digital para que el circuito digital implementado en la FPGA la trate; tres tiras de pines, dos tiras de 4 y 5 pines que se conectarán a los puertos de expansión J20 y J18 (explicados en el apartado 2.2.3) de la *Spartan-3AN Starter Kit Board*, con el fin de transmitir la señal digital a la FPGA y una tira de 2 pines que servirá como alimentación de la placa.

4.3. Circuitos impresos

Para la correcta consecución del canal de transmisión óptico se diseñan e implementan circuitos impresos específicos de emisión y recepción de señales. Estos diseños tienen como objetivo crear un enlace óptico de gran calidad, que cuya meta sea una reproducción exacta del mensaje enviado.

Los circuitos impresos son los principales elementos del canal, siendo los componentes físicos del mismo. A partir de ellos se emitirá el haz de luz que contendrá la información de transmisión deseada (expresada por la programación cargada en la FPGA) y se hará la correcta recepción del envío.

4.3.1. PCB emisora

El circuito integrado emisor tendrá el cometido de transferir el mensaje de 4 bits proporcionado por la FPGA (incluyendo el bit de relleno). Esta información será transportada por medio de un haz de luz, el cual contendrá toda la información de la señal a emitir.

Para que la funcionalidad de la placa sea la deseada, hemos diseñado el siguiente circuito (ver la figura 4.2):

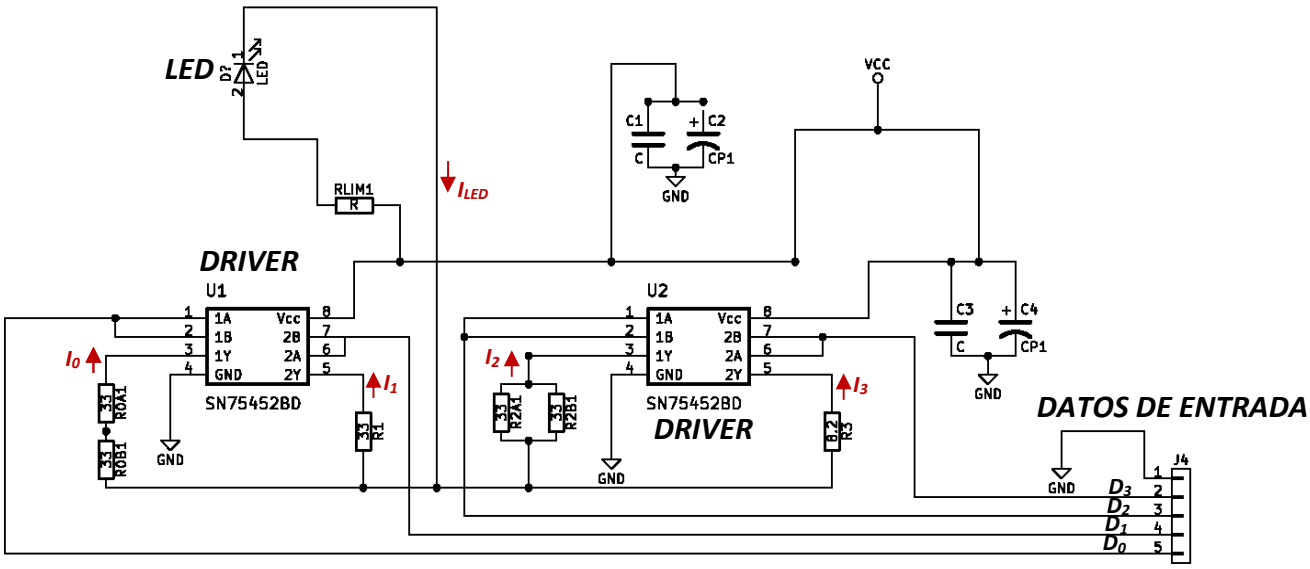


Figura 4.2. Circuito para controlar la intensidad (modulación en amplitud) de la luz emitida por el LED.

Este circuito contará con tres elementos principales: un LED encargado de reproducir el mensaje luminoso, el cual dependiendo de la intensidad de alimentación del mismo, tendrá un nivel de iluminación determinado (la iluminación es directamente proporcional a la intensidad que circula por el LED), y dos *drivers* conmutadores SN75452BD (para altas corrientes), que tienen el cometido de, en base a los datos a emitir, regular la intensidad del LED mediante la conmutación de diferentes ramas constituidas por resistencias ponderadas.

El comportamiento del diseño es muy sencillo. A cada *driver* le entran dos bits del envío $D_3D_2D_1D_0$, un bit para la entrada 1 del *driver* y otro bit para la entrada 2, saliendo un nivel de intensidad según el valor del bit pues cada rama tiene una resistencia de valor diferente, menor cuanto mayor sea el peso del bit (ver figuras 4.2 y 4.3). Observe (figura 4.2) que la entrada 1 del *driver* U1 controlada por D_0 conmuta una rama con dos resistencias en serie de 33 Ω (66 Ω en total); la entrada 2 controlada por D_1 conmuta una rama con una sola resistencia de 33 Ω, por lo que, al activarse, hará circular el doble de corriente por el LED que si se activara la rama controlada por la entrada 1. Para el *driver* U2 controlado por las líneas de datos D_2 y D_3, las resistencias que se conmutan son dos resistencias de 33 Ω en paralelo (16,5 Ω) y 8,2 Ω respectivamente, por lo que dan lugar a corrientes por el LED al activarse que son el doble y el cuádruple de las de la rama controlada por D_1 (cuatro y ocho veces superiores a las de la rama controlada por D_0). De esta forma, combinando la activación o no de estas

cuatro entradas es posible obtener hasta 16 intensidades de corriente diferentes, esto es, un haz lumínico con hasta 16 niveles (16-PAM).

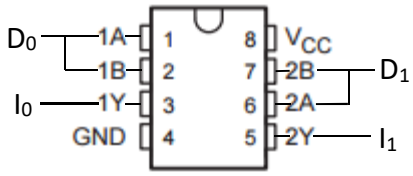


Figura 4.3. Driver conmutador SN75452BD [Driver].

A partir del circuito lógico interno del *driver* (figura 4.4), es posible entender su funcionamiento. Consiste en una puerta NAND, a la que le entra el bit (una de las líneas de datos D_3 a D_0): si este bit es un cero lógico, se polariza la base del transistor haciendo que circule corriente por la salida *Y*.

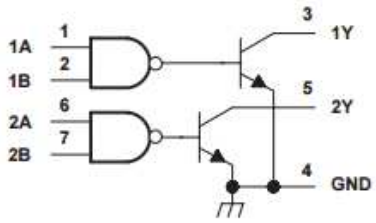


Figura 4.4. Estructura interna del driver [Driver].

Las cuatro ramas de corriente de salida confluyen al mismo punto donde se conecta el LED, sumándose, haciendo que éste se ilumine según el nivel de intensidad dependiente del mensaje deseado. Además, para que el LED reproduzca correctamente los datos, el *driver* tiene un gran nivel de conmutación entre salidas, por lo que varía la frecuencia, haciendo que el LED sea capaz de emitir la información correctamente con el objetivo de que el fotodiodo posterior pueda recoger la variación de intensidad, dibujando la señal anhelada.

Tras diseñar el circuito, proseguimos a fabricar la PCB según lo expuesto en el apartado 2.4 (ver figura 4.5). El resultado es satisfactorio a lo esperado, pudiendo emitir señales de la forma deseada, estos resultados se expondrán en el Capítulo V.

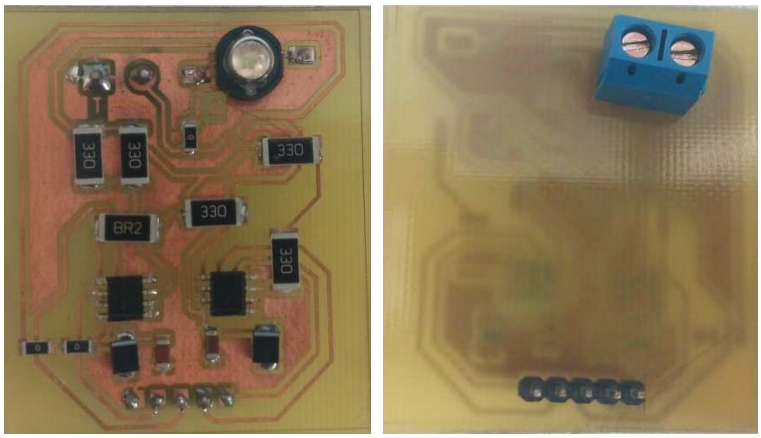


Figura 4.5. Placa emisora.

4.3.2. PCB receptora

El circuito receptor será el encargado de reproducir correctamente la señal analógica transmitida por el LED, teniendo como función el captar la señal más nítida para enlazarla de nuevo a la FPGA a través del ADC, con el objetivo de que este realice el último tratamiento de la información.

El receptor se conectará a un fotodiodo, cuya misión es recibir el haz de luz, de forma que capture la variación de intensidad del mensaje. Para reproducir dicha variación de intensidad de forma adecuada, creamos el siguiente circuito:

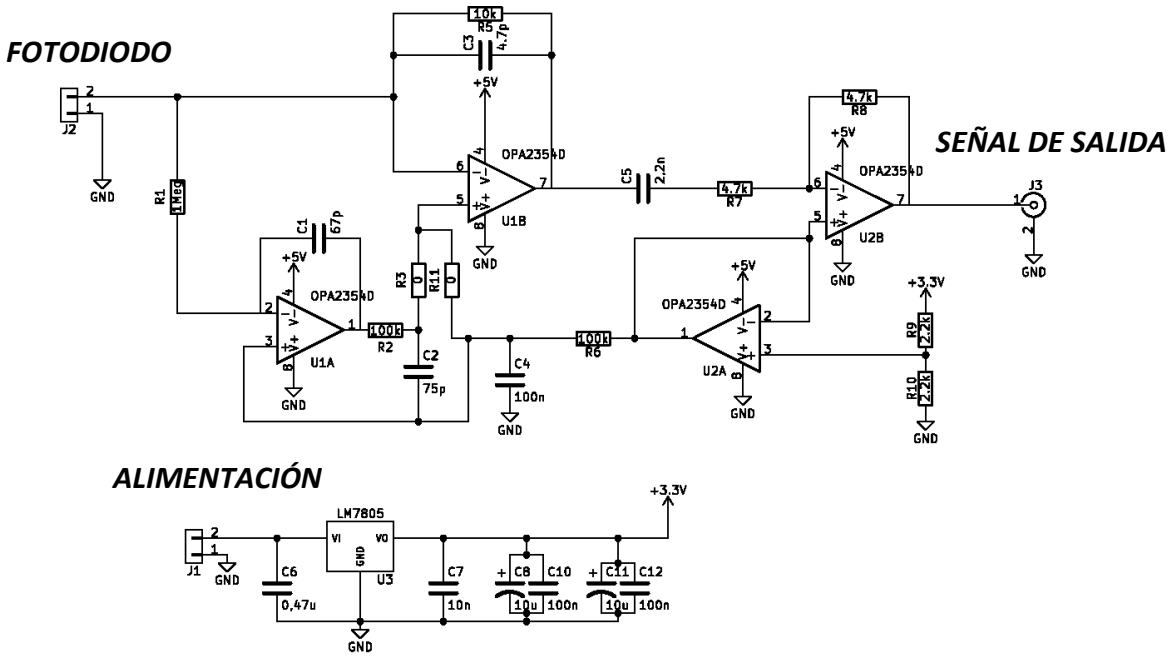


Figura 4.6. Circuito de la placa receptora.

Este circuito se compondrá por tres operacionales, dos OPA2354 (circuitos integrados que usan la tecnología CMOS) (figura 4.7), elegidos por su gran ancho de banda, su bajo ruido y por tener como aplicación el tratamiento de los fotodiodos. Y un regulador de tensión LM7805, que tiene como misión alimentar el circuito a 3,3 V (ver figura 4.8). Además, se colocan condensadores de desacoplo en paralelo, para desacoplar las señales de corriente alterna de la señal de corriente continua del circuito.

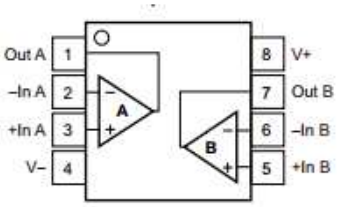


Figura 4.7. Patillaje del amplificador operacional dual OPA2354 [OPA].

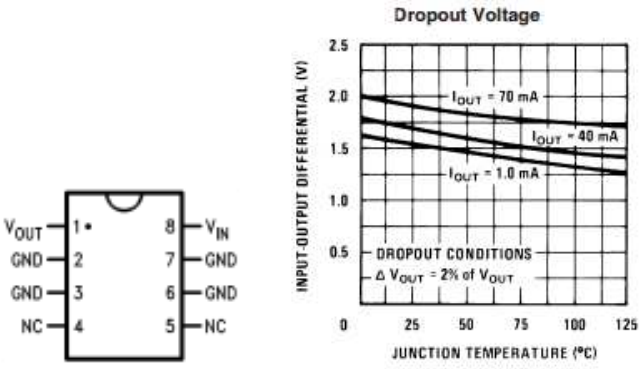


Figura 4.8. Patillaje del regulador de tensión LM7805 y sus curvas características de dropout [Regulador].

Los OPA2354 son integrados con dos amplificadores encapsulados, que serán utilizados de la siguiente manera:

- Integrado U1 (ver figura 4.9):

Albergará el amplificador de trasimpendancia, que tiene como misión convertir la corriente recibida por el fotodiodo a tensión, produciendo a la salida una tensión proporcional a esta, la cual contendrá el mensaje que se mandará al amplificador inversor del integrado U2. Además, se le añade un condensador en paralelo para que actúe como filtro pasa bajas, destinado a la reducción del ruido.

Por otro lado, contiene un amplificador integrador, este no se usará en la práctica. Lo implementamos debido a que el OPA380 (operacional ideal para la recepción de la señal del fotodiodo) lo contenía en su configuración recomendada para estas actividades, con el fin de

alimentar el amplificador de trasimpendancia óptimamente. Pero en la práctica no nos respondía correctamente por lo que decidimos desactivarlo (por lo que no soldamos la resistencia R3 cuyo valor es de cero ohmios).

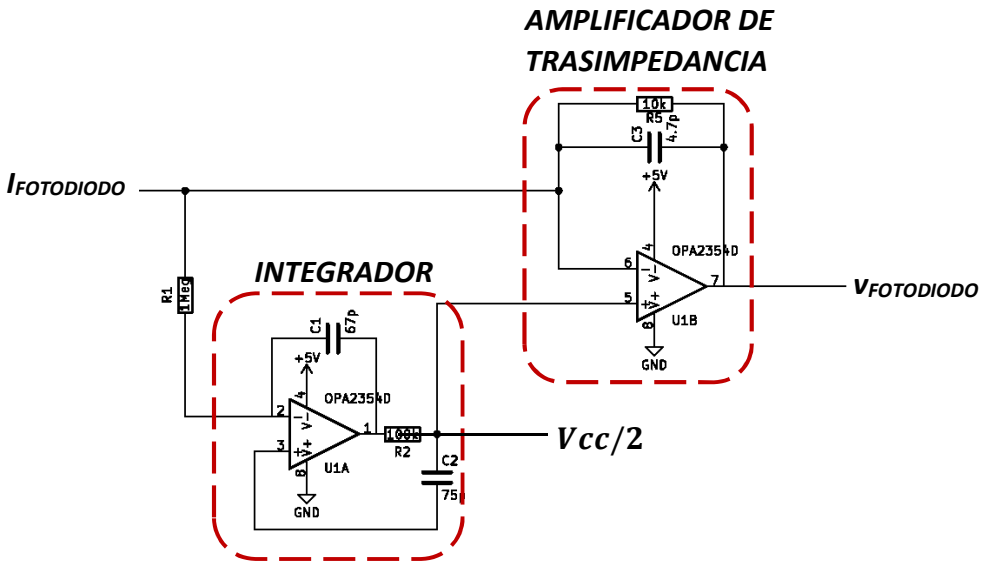


Figura 4.9. Circuitos asociados al integrado U1.

❖ Integrado U2 (ver figura 4.10):

Hará acopio de un seguidor de tensión, el cual tendrá como tensión de entrada la salida del regulador. Esta tensión se encontrará dividida a la mitad debido a un divisor de tensión. Este amplificador servirá para aislar la tensión recibida del partidor de tensión y alimentar las entradas del resto de operacionales, haciendo que polarice el fotodiodo a $Vcc/2$.

Por último, está compuesto por un amplificador inversor de ganancia uno, que ejercerá como un filtro pasa altas, ya que a la entrada se le conecta un condensador en serie. Este condensador ayudará a la reproducción de la señal, por lo que debe de ser de un valor alto para que no descargue rápidamente y se pueda interpretar la señal de forma correcta. La salida del amplificador mostrará la señal de recepción tras pasar por todo el circuito.

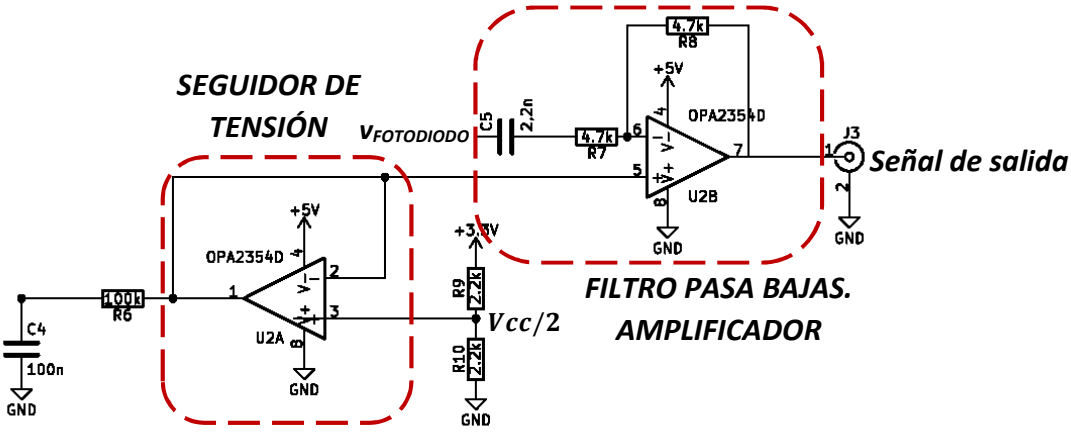


Figura 4.10. Circuitos asociados al integrado U2.

Una vez diseñado la placa eligiendo los componentes y las configuraciones definidas anteriormente, proseguimos a elaborarla físicamente (ver la figura 4.11). Este circuito integrado reproduce satisfactoriamente la señal emitida, aunque con cierto nivel de ruido superpuesto a la misma. Los resultados de los experimentos se mostrarán en el Capítulo V.

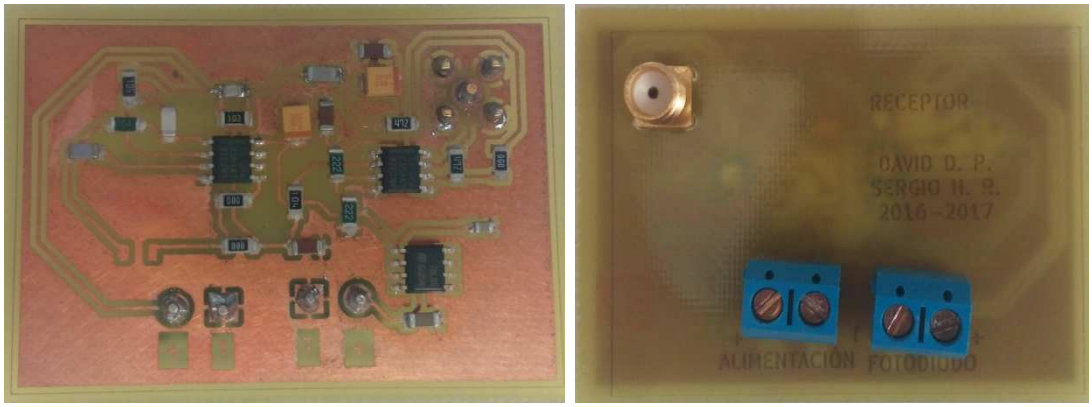


Figura 4.11. Placa receptora.

4.3.3. PCB ADC

El convertidor analógico a digital debe asignar, a cada valor analógico de la señal proveniente de la PCB receptora, un valor digital, obteniendo la señal digital convertida pudiendo permitir el procesado de la misma mediante los bloques VHDL implementados en la FPGA.

El circuito de esta placa contiene un único integrado, un ADS805, el cual es un ADC de alta velocidad de conversión que puede muestrear a 20 MHz. Para su correcta configuración creamos el siguiente circuito (figura 4.12):

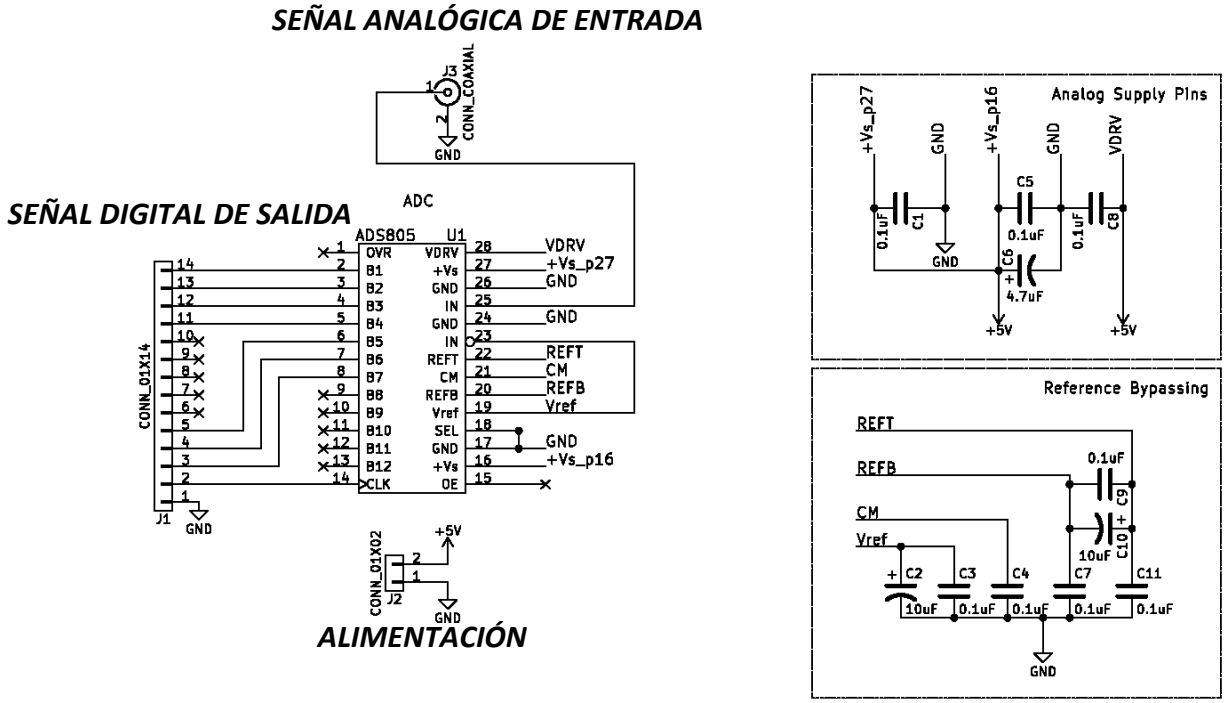
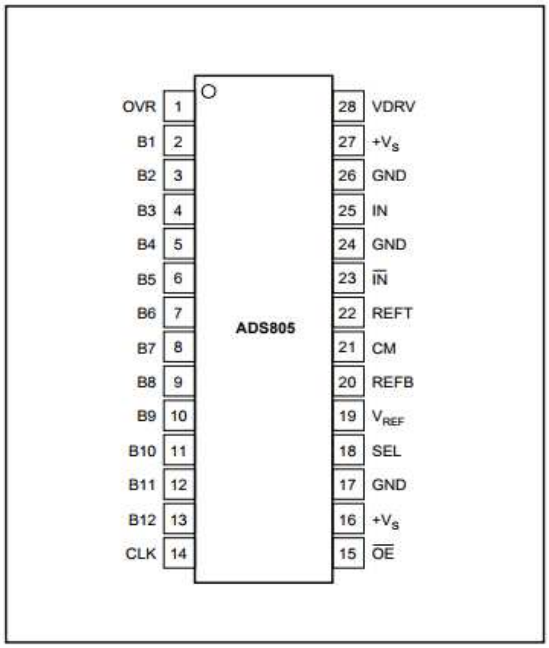


Figura 4.12. Circuito de la placa ADC.

La configuración y descripción de los pines del integrado, son los consecutivos (figura 4.13):



PIN	DESIGNATOR	DESCRIPTION
1	OVR	Over-Range Indicator
2	B1	Data Bit 1 (D11) (MSB)
3	B2	Data Bit 2 (D10)
4	B3	Data Bit 3 (D9)
5	B4	Data Bit 4 (D8)
6	B5	Data Bit 5 (D7)
7	B6	Data Bit 6 (D6)
8	B7	Data Bit 7 (D5)
9	B8	Data Bit 8 (D4)
10	B9	Data Bit 9 (D3)
11	B10	Data Bit 10 (D2)
12	B11	Data Bit 11 (D1)
13	B12	Data Bit 12 (D0) (LSB)
14	CLK	Convert Clock Input
15	$\overline{OE}$	Output Enable. H = High Impedance State. L = LOW or floating, normal operation (internal pull-down resistor).
16	$+V_S$	+5V Supply
17	GND	Ground
18	SEL	Input Range Select
19	V_{REF}	Reference Voltage Select
20	REFB	Bottom Reference
21	CM	Common-Mode Voltage
22	REFT	Top Reference
23	$\overline{IN}$	Complementary Analog Input
24	GND	Ground
25	IN	Analog Input (+)
26	GND	Ground
27	$+V_S$	+5V Supply
28	VDRV	Output Driver Voltage

Figura 4.13. Configuración y descripción de los pines del ADS805 [ADC].

Para que el dispositivo funcione adecuadamente, únicamente se le debe proporcionar una alimentación adecuada (V_S = +5V), una señal dentro de los rangos definidos (de 0 a 5V) según su configuración (figura 4.14), y un reloj cuya frecuencia no superé el límite de 20 MHz.

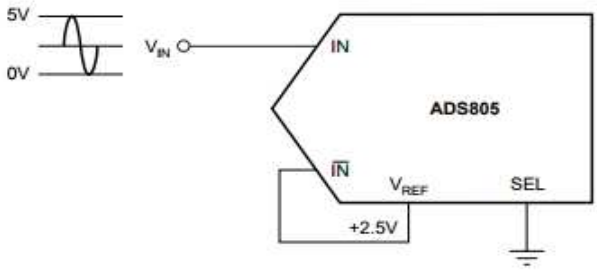


Figura 4.14. Selección del rango de entrada del ADS805 [ADC].

En relación a la referencia de operación del circuito integrado, hemos elegido la configuración recomendada por el fabricante en su *datasheet* (figura 4.15):

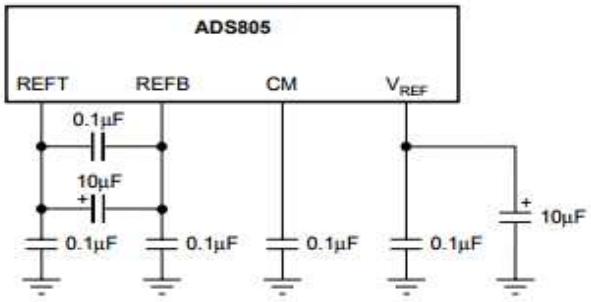


Figura 4.15. Referencia de operación del ADS805 [ADC].

Por último, diseñamos la conexión de los planos de tierra y el desacoplo del circuito (figura 4.16) para las señales analógicas. Estas son de extremada importancia para diseños que trabajan a altas frecuencias.

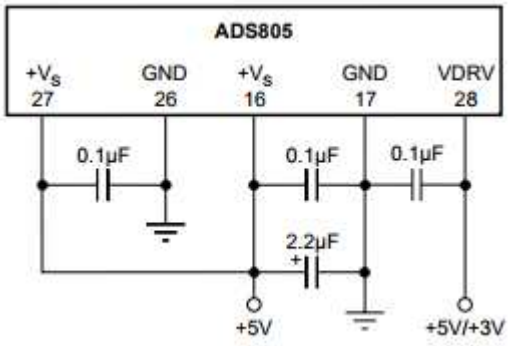


Figura 4.16. Configuración recomendada para grounding and decoupling de los pines analógicos [ADC].

Para finalizar la placa, procedemos a su elaboración física para incorporarla en el enlace óptico (ver figura 4.17). Hemos de indicar que la funcionalidad obtenida con la placa no fue la deseada, pues, aun siguiendo las configuraciones especificadas en su hoja de datos, no fue capaz de convertir la señal analógica en una señal digital. Los detalles correspondientes a su funcionamiento experimental se detallarán en el Capítulo V.

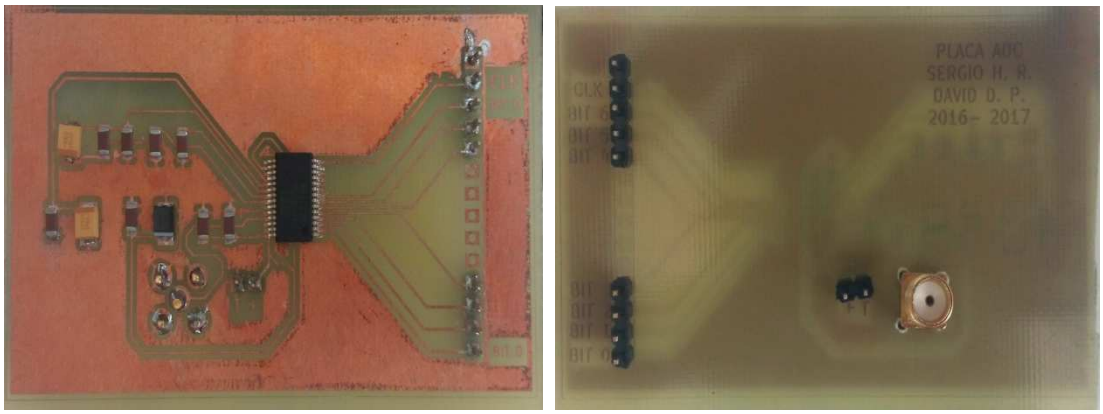


Figura 4.17. Placa ADC.

4.4. Funcionamiento completo del prototipo

El prototipo creado se divide en los diversos componentes que se han ido desarrollando en los apartados previos. El apartado actual se encontrará dedicado a la explicación del funcionamiento conjunto de todos ellos paso por paso enumerando los componentes que actúan en cada momento del proceso. Para clarificar la explicación se adjunta al final de la misma un diagrama que la ilustra.

En primer lugar, se debe decidir si se el objetivo de uso del prototipo es la transmisión de datos o, como método de comprobación de funcionamiento, la generación interna de los datos a transmitir. Esto se debe al uso o no del módulo de generación de datos creado en el transmisor, el cual, si se precisa, generará una secuencia de datos pseudoaleatoria que será transmitida a través del prototipo. En el caso contrario será necesario conectar la fuente de los datos a transmitir a uno de los conectores de entrada o entrada/salida de la tarjeta.

Con los datos ya generados, el sistema comenzará la preparación de la señal para su envío a través del enlace óptico. El primer paso en la preparación de la señal es la conversión de los datos seriales generados en datos paralelos formando vectores de tres bits los cuales se rellenarán con un bit lógico ‘0’ (reservado para uso futuro) para finalmente tener un bus de cuatro bits.

En el caso de que dicho bus de datos no llegue a este punto, es decir, que no se hayan generado los suficientes datos seriales para completar el bloque de cuatro (tres de datos) bits, se producirá un relleno de la señal. Esto se realiza para evitar que el diodo emisor no se apague y se mantenga estable el nivel de iluminación. El relleno se realizará enviando una secuencia de dos bloques, uno primero con todos los bits a '1' y seguidamente otro con todos los bits a '0', los que producirá que el nivel de la luz emitida por el led se mantendrá a una intensidad media.

Llegado a este punto, la señal está preparada para ser transmitida por el circuito emisor. Por lo tanto, esta se enviará a dicho circuito a través de cuatro cables (debido a que el mensaje es de cuatro bits) por uno de los puertos de la tarjeta. En la tarjeta que contiene el LED emisor, los dispositivos *drivers* se ocuparán de conmutar la señal, lo que permitirá realizar los cambios de intensidad lumínica que servirán para la transmisión del mensaje.

La recepción de la señal comienza en el fotodiodo, el cual detectará los cambios en la intensidad lumínica producidos en el circuito anterior y los traducirá en distintos valores de corriente que a través de los distintos amplificadores operacionales y sus respectivos circuitos que se encuentran en el circuito receptor, generarán una señal de tensión que será muy parecida a la enviada al circuito emisor, pero con elevados valores de ruido.

Los valores analógicos de voltaje creados en el circuito receptor serán convertidos a valores digitales de doce bits en el circuito ADC. Estos buses digitales serán reenviados a la tarjeta para realizar el procesado de la señal.

La tarjeta comienza el procesado de la señal con el filtro FIR que, gracias a la especialidad de los filtros digitales en la recuperación de señales, ayudará en gran medida a reducir el ruido. La señal filtrada se muestrea con el *Sample & Hold*, el cual se encontrará controlado por el reloj generado en el módulo generador a partir de los impulsos obtenidos del detector de umbral y del bloque derivador.

Tras ser muestreada, la señal se decodificará para reducir el número de bits, de doce a los cuatro de información originales. De ello se encargará el decodificador que, mediante los valores máximos y mínimos de tensión de la señal de salida del filtro FIR, obtenidos a partir del módulo detector de máximos y mínimos, divide los valores de señal en distintos niveles de tensión (en concreto 15 niveles, debido al número de bits de salida). De esta forma, la señal

queda reducida a cuatro bits los cuales son analizados por el bloque eliminador de relleno que detectará si dichos datos han sido creados por el módulo de relleno antes del envío de los mismos. En caso afirmativo, estos datos serán eliminados, ya que no pertenecen al mensaje inicial.

Con este último análisis la señal obtenida será la misma que se obtiene tras el convertidor serie-paralelo en el extremo transmisor. Tras esto la señal puede ser visualizada, tras ser o no convertida a datos seriales, en otros dispositivos externos al prototipo o utilizarse para otros fines.

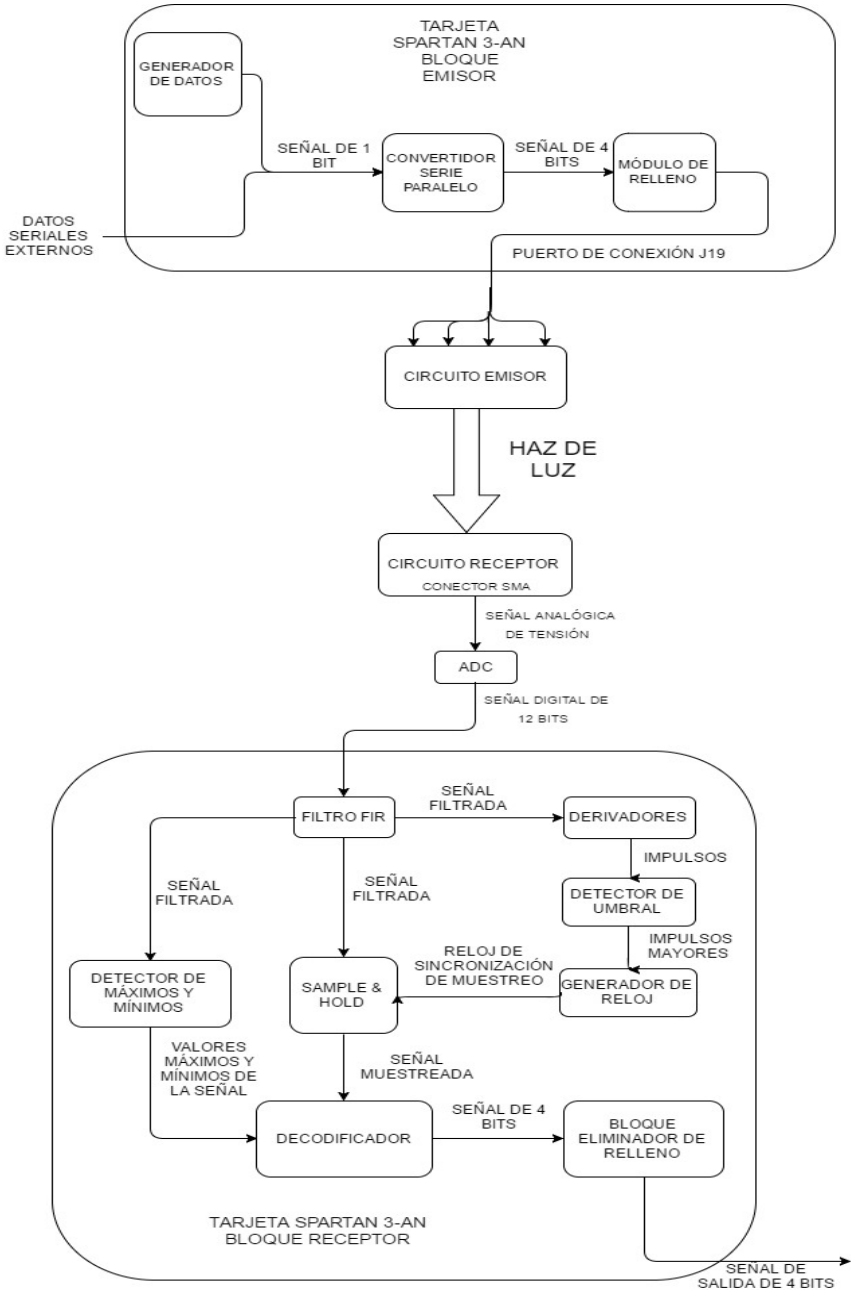


Figura 4.18. Diagrama de bloques del diseño completo implementado.

Capítulo V:

Pruebas de funcionamiento realizadas y resultados obtenidos.

5. Capítulo V. Pruebas de funcionamiento realizadas y resultados obtenidos.

5.1. Introducción.

En el presente capítulo se expondrán las diferentes experimentaciones que se han realizado para la comprobación del funcionamiento de las diversas partes del prototipo y, tras ello, se mostrarán los resultados obtenidos en dichas pruebas, que serán ilustrados mediante capturas realizadas al osciloscopio digital RIGOL.

5.2. Pruebas parciales de funcionamiento.

Este apartado se encontrará dividido en las diferentes pruebas realizadas, las cuales se basan en la comprobación de los diferentes módulos, por lo que tendremos:

- Comprobación de funcionamiento del bloque emisor digital.
- Comprobación de funcionamiento del bloque receptor digital.
- Funcionamiento del bloque digital.
- Test de conmutación de los *drivers* del emisor. Funcionamiento PCB emisora.
- Pruebas del enlace óptico.
- Test de conversión de la placa del conversor analógico-digital.

5.2.1. Comprobación de funcionamiento del bloque emisor digital.

Estas sencillas comprobaciones se realizaron durante la redacción del código VHDL, para asegurar el correcto funcionamiento de los módulos del bloque emisor. Las pruebas consistieron en la simulación de la llegada de los datos gracias al generador de datos pseudoaleatorios el cual proporcionaba los datos seriales que se procesaban en los diferentes módulos. Durante el capítulo tres relativo a la programación se muestran muchas de las simulaciones realizadas durante estas experimentaciones, dando como resultado final del procesado la señal mostrada en la figura 5.1.

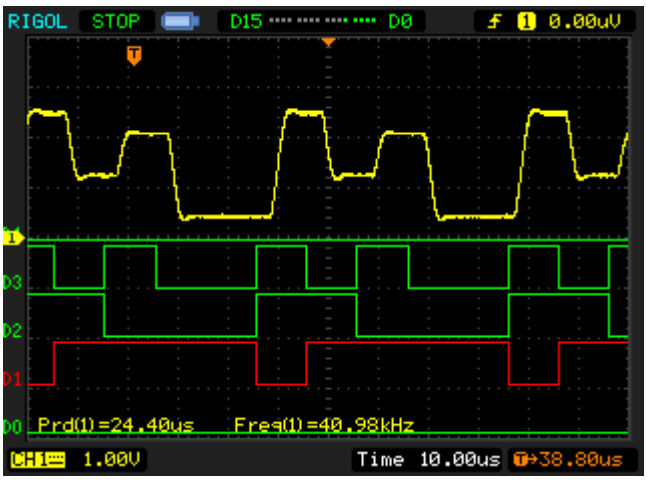


Figura 5.1. Salida del bloque emisor digital.

5.2.2. Comprobación de funcionamiento del bloque receptor digital.

Al igual que las comentadas en el apartado anterior estas se realizaron durante la redacción del código VHDL. Las pruebas consistieron en la simulación de la transmisión de los datos de forma digital mediante la conexión digital de la salida del bloque de relleno del emisor al filtro FIR y gracias al generador de datos pseudoaleatorios se simulaba la transmisión completa. Tal y como ocurre con el bloque emisor en el tercer capítulo se muestran muchas de las simulaciones realizadas durante estas experimentaciones, dando como resultado final del procesado la señal mostrada en la figura 5.2.

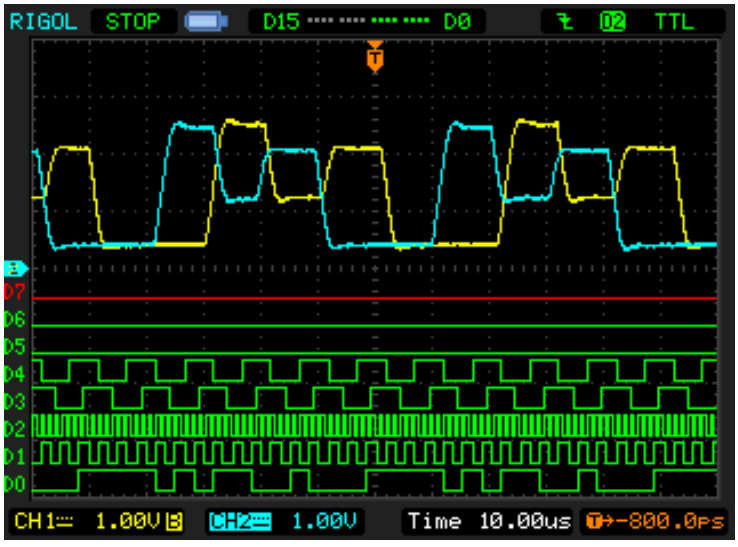


Figura 5.2. Salida del bloque receptor digital (canal 1, amarillo) en comparación con la salida del emisor digital (canal dos, azul).

5.2.3. Funcionamiento del bloque digital.

Para confirmar que todo el bloque digital y con ello la programación en VHDL marcha de la forma correcta se ha realizado una prueba definitiva en la cual se han conectado a través de los conectores J18 y una serie de cables eléctricos dos tarjetas SPARTAN 3AN. La primera de ellas se encuentra programada con el código emisor, por lo que se obtendrá el bus de cuatro bits de salida del bloque de relleno, estos datos serán enviados a la segunda tarjeta, la cual se encontrará programada con el código receptor. Esto, a diferencia de las pruebas anteriores nos permiten asegurar que funciona, ya que, la única diferencia que se produce entre este montaje y el del prototipo final es la sustitución del enlace óptico por los cables eléctricos utilizados quedando dispuesto de la manera mostrada en las figuras a continuación.

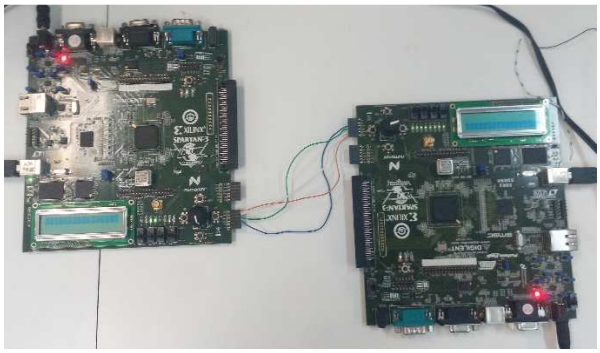


Figura 5.3. Conexión cableada entre las tarjetas para la simulación.

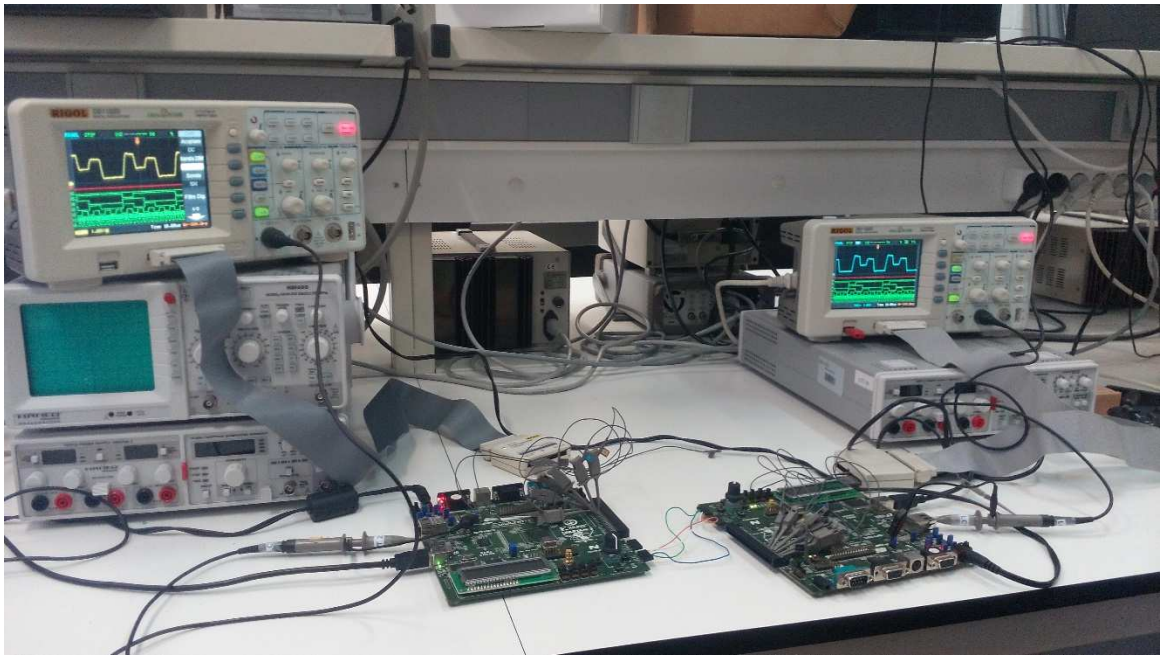


Figura 5.4. Montaje de las pruebas de funcionamiento digital.

Tras la realización de los test nombrados previamente se ha demostrado que el circuito digital creado mediante el código actúa de la forma esperada, dado que la señal que es enviada por el bloque emisor se consigue reproducir tras el procesamiento de la misma en el receptor tal y como se expone en las siguientes imágenes.

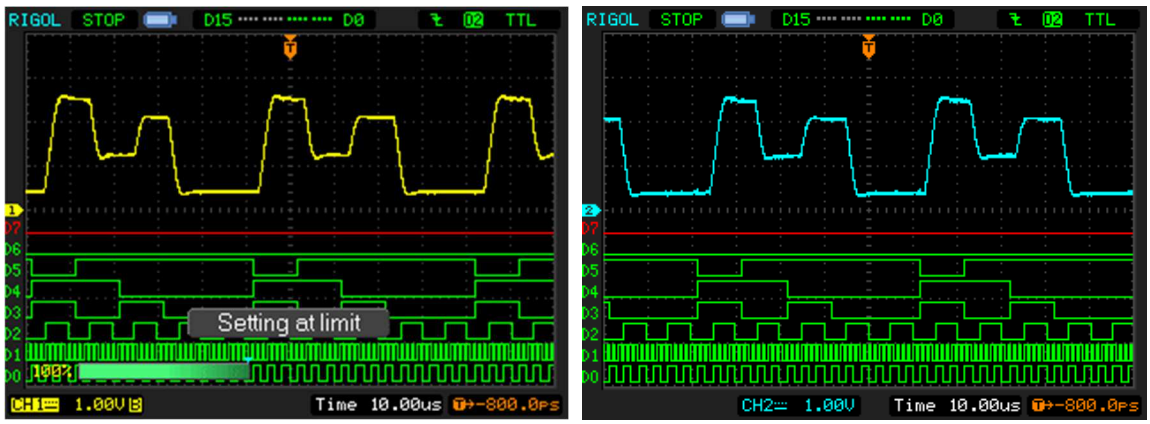


Figura 5.5. Comparación de señales a las salidas del emisor (amarilla) y del receptor (azul).

5.2.4. Test de conmutación de los *drivers* del emisor. Pruebas PCB emisora.

La primera de las pruebas realizadas sobre los circuitos analógicos se utiliza para confirmar la buena marcha del circuito emisor. Esta prueba se basa en el estudio del mensaje enviado por el diodo LED del circuito. Para ello se han cortocircuitado las entradas digitales (líneas $D_4D_3D_2D_1$) de los *drivers*, y se ha introducido una señal cuadrada, originada en un generador de señales, en todas las entradas al mismo tiempo. De este modo, el circuito, mediante las distintas variaciones de corriente eléctrica, provoca que el LED envíe dicha señal generada a partir de la variación de la intensidad lumínica. Para comprobar que dichas variaciones se realizan de forma correcta se ha utilizado una señal de muy baja frecuencia, menor a treinta hertzios, esto permite que el parpadeo ocasionado por los ciclos de alta y baja de la señal cuadrada sea visible al ojo humano y así poder ver que la conmutación que se produce en los drivers es acertada. El montaje utilizado en este test se muestra en la figura 5.6, generando el haz de luz presentado en la figura 5.7.

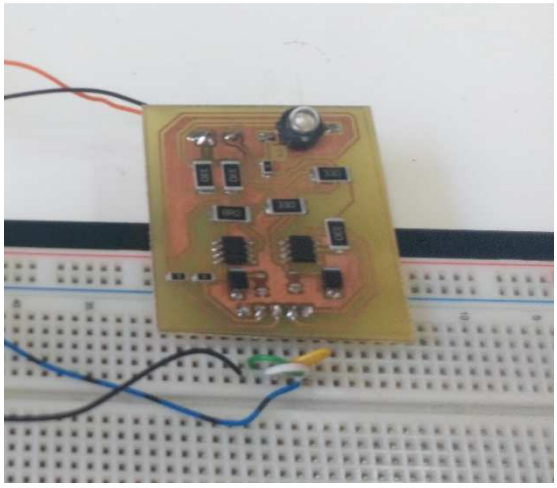


Figura 5.6. Montaje de prueba de conmutación de los drivers mediante cortocircuito de las entradas.

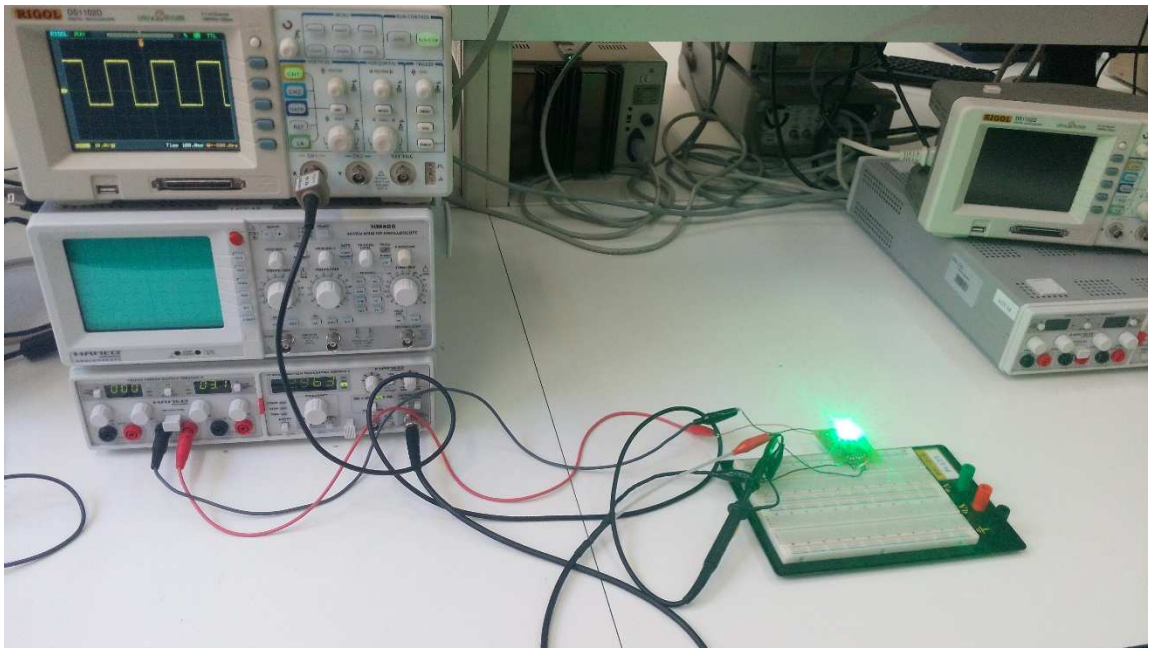


Figura 5.7. Montaje de completo con la entrada de una onda cuadrada de 2.963 Hz.

5.2.5. Pruebas del enlace óptico.

Tras asegurar que el circuito emisor transmite las señales que son introducidas en él el siguiente paso es confirmar que el circuito receptor recibe las variaciones lumínicas enviadas y las convierte en las señales originadas en el generador de señales. Para ello se ha mantenido el montaje utilizado en el test previo colocando enfrente del diodo LED el fotodiodo, lo que permitiría recibir las variaciones emitidas, por último, se ha conectado una sonda de osciloscopio a la salida del circuito receptor, lugar donde se debería visualizar la señal emitida, pero con una reducción de la amplitud debido a que los valores de intensidad lumínica no

contemplan el voltaje de la señal emitida. Esta prueba dejaría el enlace de la manera que se presenta en las figuras 5.8 y 5.9.

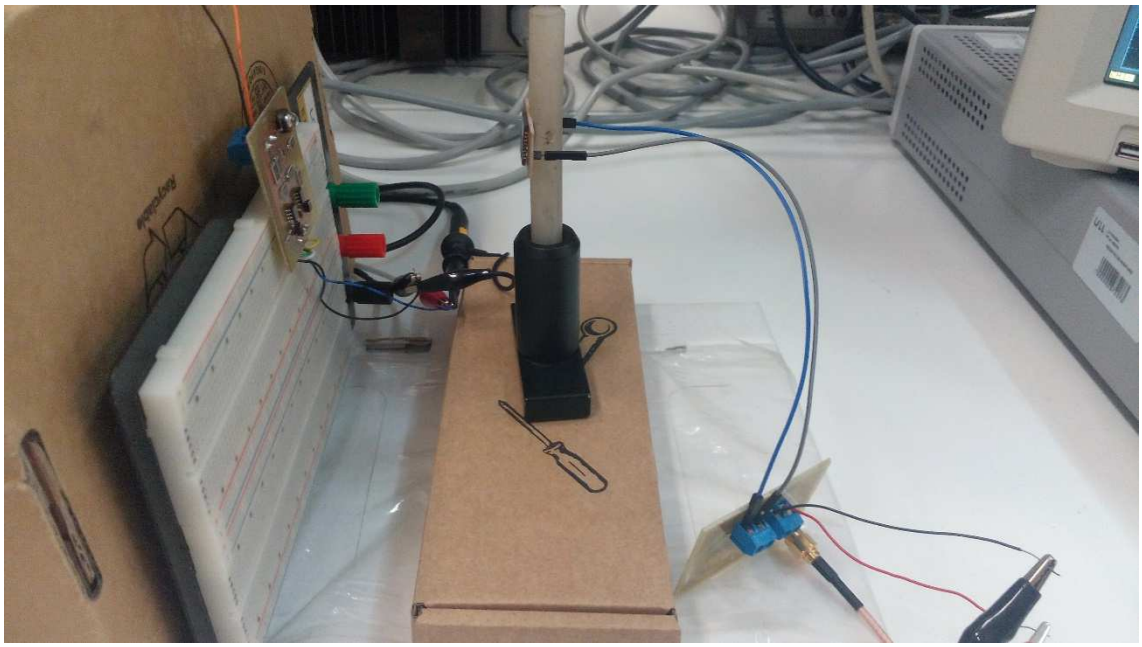


Figura 5.8. Primer montaje de pruebas de funcionamiento del enlace óptico.

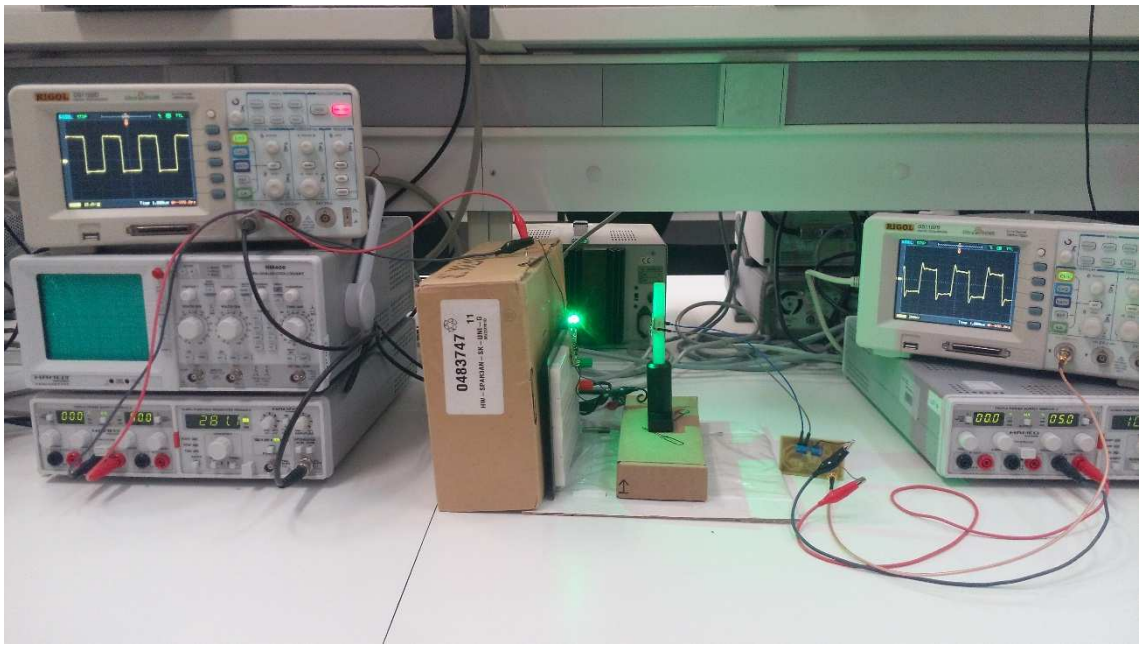


Figura 5.9. Montaje de completo con la entrada de una onda cuadrada de 281.1 kHz.

Los resultados obtenidos en estas pruebas han sido satisfactorios pese a que en algunos momentos los niveles de ruido encontrados eran demasiado altos, lo que provocó el cambio de valores en ciertos condensadores en el esquemático final. La señal obtenida y su comparación con la señal emitida se muestran en la figura a continuación.

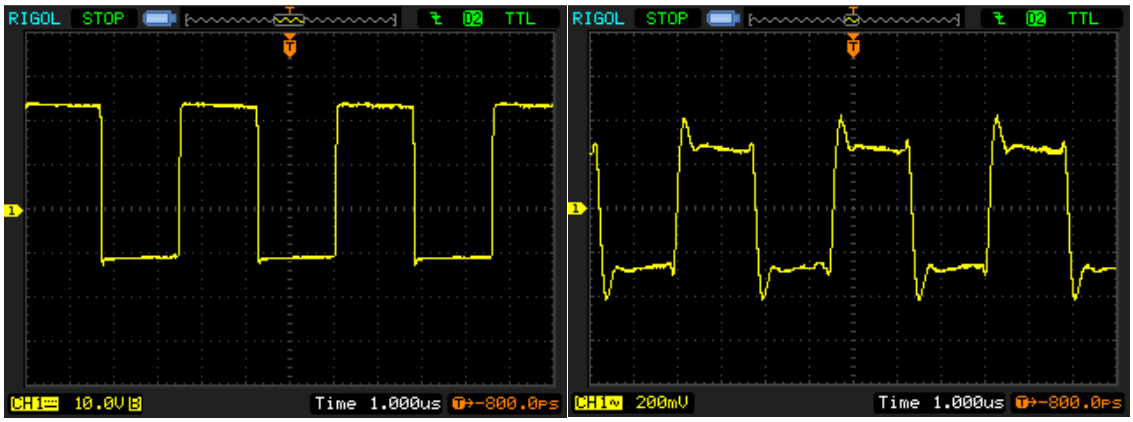


Figura 5.10. Comparación de señales a la entrada del emisor (imagen de la izquierda) y a la salida del receptor (imagen de la derecha).

Con la confirmación del correcto funcionamiento del enlace a una entrada analógica se ha decidido realizar la misma prueba que en el caso anterior, pero acoplando la tarjeta SPARTAN 3AN programada con un código de prueba al circuito del diodo LED. El código utilizado consistía en la creación de una escalera que mostraría los distintos niveles de tensión que se podrían obtener mediante la conmutación de los drivers. De esta forma se podría analizar el comportamiento del enlace ante una entrada digital tal y como sucedería en el prototipo final. En este caso el montaje es el mostrado en la figura 5.11, obteniendo la señal presente en la figura 5.12.

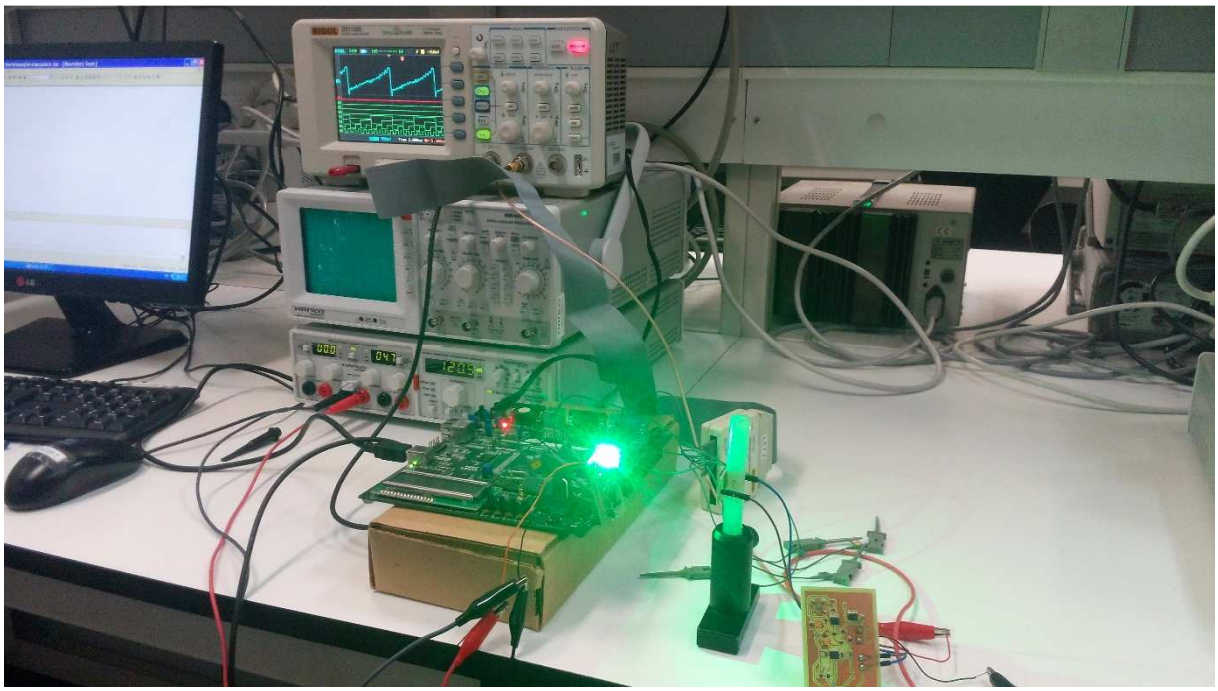


Figura 5.11. Segundo montaje de pruebas de funcionamiento del enlace óptico. Utilización de un código de pruebas para la creación de la señal.

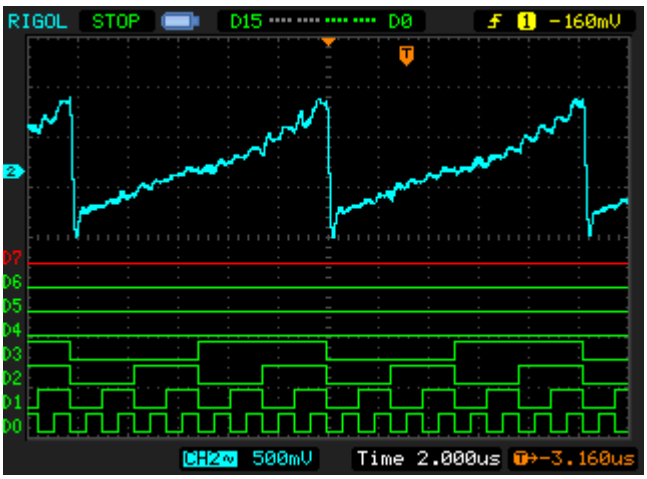


Figura 5.12. Señal de salida de la prueba de escalera del enlace óptico.

Como podemos ver la señal simula aproximadamente una escalera, pero con un pequeño transitorio generado en el flanco en el cual todos los bits toman el valor cero. Esta prueba nos permitió detectar que había un defecto en unos de los *drivers*, por lo que se procedió a su sustitución, tras lo cual se observó el funcionamiento correcto mostrado en la figura 5.12.

Analizado el comportamiento del enlace ante entradas digitales se programó la tarjeta con el código creado para el bloque emisor. En este caso la señal que se recibiría en el circuito receptor debería ser similar a la que se obtiene a la salida del módulo convertidor serie-paralelo. El montaje es el mismo que el utilizado en la prueba anterior y la señal obtenida se muestra en la figura 5.13 comparada con la de salida del módulo emisor digital.

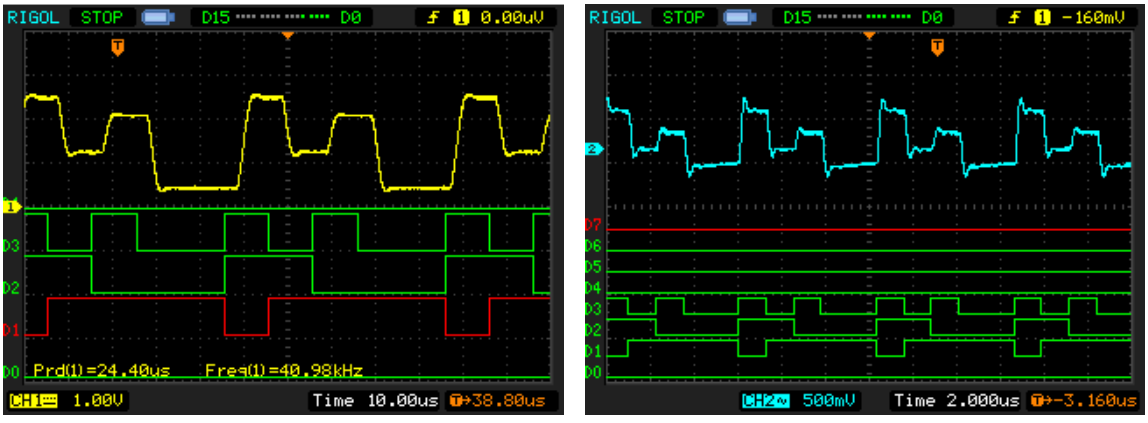


Figura 5.13. Comparación de señales a la salida del módulo emisor digital (imagen de la izquierda) y a la salida del receptor (imagen de la derecha).

En las figuras anteriores, las líneas D_3 a D_0 se corresponden con los datos procedentes del conversor serie-paralelo, donde el bit menos significativo se asocia al bit reservado que siempre está a '0'. Es importante señalar que la diferencia de frecuencias de funcionamiento

observadas en ambas figuras se debe al hecho de que el DAC contenido en la tarjeta de desarrollo de la SPARTAN no puede trabajar a tasas muy altas, por lo que ambas capturas fueron obtenidas en situaciones distintas, sólo a efectos de observar su similitud. Por el contrario, en la figura de la derecha se observa el correcto funcionamiento del enlace óptico a una tasa de símbolo de aproximadamente 1 MHz, lo que daría lugar a tasas de datos de hasta 4 Mbit/s.

5.2.6. Test de la placa del conversor analógico-digital.

Con motivo de realizar la conversión analógico-digital de la señal de salida del receptor para poder ser reenviada a la tarjeta con motivo de ser procesada en ella. Por esa razón se ha creado a placa que será analizada en este apartado.

Para el testeo de esta placa se ha ensamblado misma con el conector J17 de la tarjeta SPARTAN 3AN y se ha introducido una señal sinusoidal generada externamente a través del conector SMA integrado en la PCB y se ha programado la tarjeta para recibir los datos convertidos por el ADC y convertirlos de nuevo a datos analógicos, a partir del DAC de la tarjeta para poder ser visualizados en el osciloscopio, quedando el montaje de la simulación de la manera siguiente (figura 5.14).

Figura 5.14. Montaje de prueba del conversor analógico-digital.

5.3. Pruebas finales de funcionamiento.

Pese a que la señal obtenida no era del todo satisfactoria se ha decidido realizar el montaje completo del canal de comunicación. Para ello se han conectado todos los elementos tal y como se describe durante el capítulo IV, y se observará la salida en un primer momento del

ADC para más tarde visualizar la salida final del prototipo. Este montaje es el mostrado en las figuras 5.15 y 5.16.

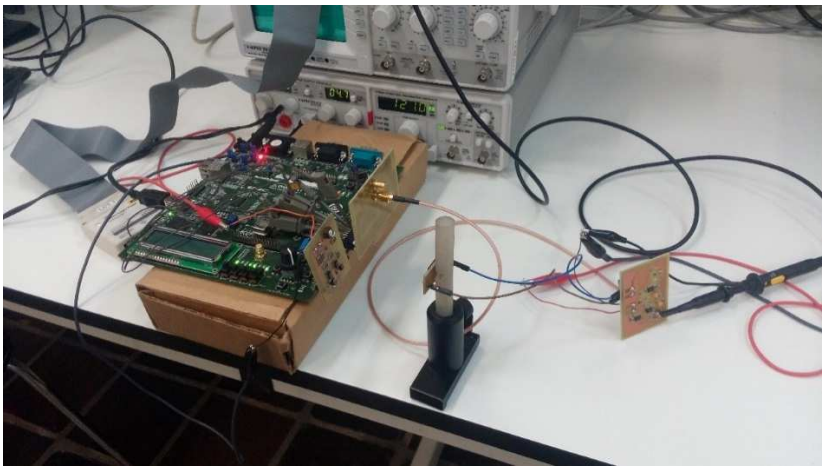


Figura 5.15. Montaje de testeo del prototipo final.

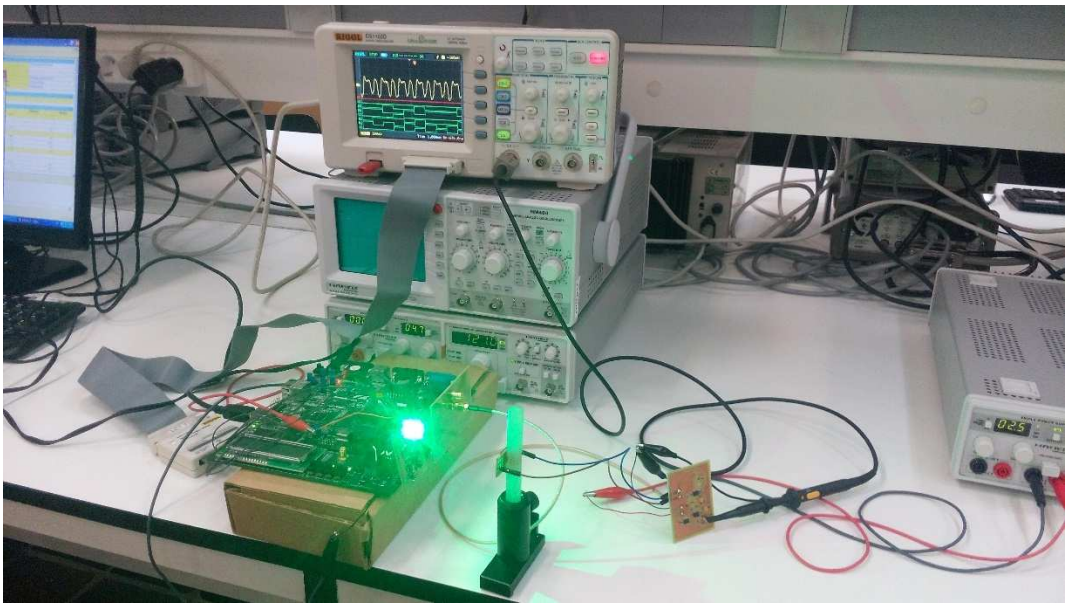


Figura 5.16. Montaje y funcionamiento del prototipo final.

Los resultados obtenidos en estas pruebas no han sido satisfactorios ya que no se ha podido llevar a cabo la conversión.

Debido a los problemas de funcionamiento nombrados no se ha podido continuar con el proyecto en sí mismo, ya que era estrictamente necesaria la conversión analógica-digital de la señal porque sin ello no sería imposible el procesamiento de la misma y reproducirla de manera correcta a la salida del canal de comunicación. La imposibilidad del cambio del

conversor por otro modelo diferente se debe a la necesidad de utilizar uno con elevadas velocidades de conversión, y, por lo tanto, nos hemos visto obligados a dejar pendiente esta última parte del trabajo, a expensas de conseguir un nuevo conversor del cual podamos obtener la salida digital necesaria y además permita una conversión adecuada a la frecuencia de envío de la señal.

Capítulo VI:

Presupuesto.

6. Capítulo VI. Presupuesto.

6.1. Introducción

En el transcurso de este capítulo se explicará detalladamente el presupuesto del prototipo, incluyendo en él los costes materiales del mismo (coste de cada PCB y de las FPGA utilizadas) y el coste de la mano de obra.

6.2. Costes materiales

Los costes materiales contendrán los precios unitarios de cada elemento necesario para la fabricación de los circuitos impresos, al igual que el coste de las herramientas para llevar a cabo el enlace óptico.

Tabla 6.1. Coste de la PCB receptora

DESCRIPCIÓN	*CANTIDAD*	*PRECIO UNITARIO(€)*	*PRECIO TOTAL(€)*
Condensadores SMD	7	0,35	2,43
Condensadores SMD de tántalo	3	0,16	0,49
Resistencias SMD	11	0,53	5,81
OPA2354	2	2,87	5,74
Regulador 78L05	1	0,36	0,36
Conector SMA	1	1,73	1,73
Fotodiodo S7510	1	23,08	23,08
Placa repro circuit 100 x 160	1	5,2	5,2
Conector Bornier 2 pines	2	0,5	1
TOTALES	29		45,83 €

Tabla 6.2. Coste de la PCB ADC			
DESCRIPCIÓN	*CANTIDAD*	*PRECIO UNITARIO(€)*	*PRECIO TOTAL(€)*
ADC ADS805	1	17,26	17,26
Condensador SMD 100nF	8	1,28	10,24
Condensador SMD de Tántalo	3	0,16	0,48
Conector SMA	1	1,73	1,73
Conector macho-macho para PCB	1	0,92	0,92
Placa repro circuit 100 x 160	1	5,2	5,2
TOTALES	15		35,83 €

Tabla 6.3. Coste de la PCB emisora			
DESCRIPCIÓN	*CANTIDAD*	*PRECIO UNITARIO(€)*	*PRECIO TOTAL(€)*
Driver SN75452B	2	0,69	1,38
Resistencias SMD POTENCIA	9	0,26	2,37
Condensador SMD 100nF	2	1,28	2,56
Condensador SMD de Tántalo	2	0,16	0,32
Conector Bornier 2 pines	1	0,5	0,5
HIGH POWER LED HP803PG	1	6,75	6,75
Conector macho-macho para PCB	1	0,92	0,92
Placa repro circuit 100 x 160	1	5,2	5,2
TOTALES	18		20,00 €

Tabla 6.4. Costes totales de los materiales			
DESCRIPCIÓN	*CANTIDAD*	*PRECIO UNITARIO(€)*	*PRECIO TOTAL(€)*
PCB RECEPTOR	1	45,83	45,83
PCB EMISOR	1	20,00	20,00
PCB ADC	1	35,83	35,83
SPARTAN 3-AN Starter Kit	2	263,17	526,34
TOTALES	5		628,01 €

6.3. Costes de mano de obra

Las actividades realizadas para la consecución del prototipo se dividen en cuatro bloques: programación, fabricación de los circuitos impresos, optimización e implementación del enlace óptico y documentación. Teniendo en cuenta el número de horas consumidas en cada bloque, calculamos el coste de mano de obra.

Tabla 6.5. Coste de mano de obra

CONCEPTO	*CANTIDAD (h)*	*COSTE UNITARIO(€/h)*	*COSTE TOTAL (€)*
Tiempo de programación	100	20	2000
Tiempo de fabricación PCB	30	20	600
Tiempo de optimización y implementación del enlace óptico	120	20	2400
Tiempo de documentación	50	20	1000
TOTALES	300 h		6.000 €

6.4. Costes generales del prototipo

Para finalizar el capítulo calculamos los costes generales del prototipo, en los cuales se asociarán los costes materiales totales con los costes de mano de obra.

Tabla 6.6. Costes generales del prototipo

CONCEPTO	*COSTE TOTAL (€)*
Coste de mano de obra	6000
Costes totales de los materiales	628,01
TOTALES	6628,01 €

Capítulo VII:

Conclusiones.

7. Capítulo VII. Conclusiones.

7.1. Introducción

Este capítulo tratará de dar nuestro punto de vista acerca del trabajo realizado, dando nuestro grado de satisfacción y explicando las aptitudes desarrolladas para llevar a cabo el prototipo planteado.

7.2. Conclusiones

El canal de transmisión óptico implementado nos ha permitido conocer una tecnología de comunicación que está en auge. Este nuevo medio, denominado VLC (Visible Light Communication, comunicación por luz visible) transfiere información por medio de señales lumínicas, dando innumerables oportunidades de uso al consumidor. Esta tecnología, contando con un estudio en profundidad de sus recursos y apoyo económico, puede ser un gran exponente en la comunicación a medio plazo.

En nuestro caso, a la hora de realizar el enlace óptico, hemos detectado que la señal recibida no era del todo nítida y, además, contaba con una atenuación de la amplitud. Estas pequeñas deficiencias en el canal de transmisión son debidas a dos tipos de interferencias, las provenientes del circuito (ruidos internos de los componentes, en su mayoría) y las originarias por medios externos, en su caso la luz ambiental, la cual afecta notoriamente a la capacidad del enlace óptico. Aún con estas pequeñas alteraciones, hemos visto que en ambiente controlado se puede obtener una señal que simule a la emitida con un pequeño grado de incorrección. Implementándose mejoras en este canal de transmisión (con especial énfasis a reducir la sensibilidad a la luz ambiental y a las sombras) se lograría conseguir un medio seguro y eficiente para traspasar información.

Por otro lado, no hemos podido conseguir visualizar el comportamiento completo del canal de transmisión. Esto se debe a los problemas relativos al circuito de conversión analógico a digital, que no ha permitido el procesado final de la señal de recepción, lo que podría haber mejorado el grado de reproducción de la misma.

Para finalizar, a pesar de no haber alcanzado el propósito marcado al inicio del proyecto, hemos adquirido amplios conocimientos en las diversas materias en las que se compone el trabajo (ética de trabajo, electrónica, fabricación de circuitos de impresos, programación

hardware, tratamiento de señales…), lo que será de gran ayuda para nuestro futuro como ingenieros.

Bibliografía

Bibliografía

- [1]:http://www.panduboys.com/tech/li-fi-technology-is-100-times-faster-than-wi-fi-technology
- [Boston University 16]: https://www.bu.edu/eng/2016/09/16/wifi-to-lifi/, Boston University
- [IETE Journal of Research, 2013]: http://www.tandfonline.com/doi/abs/10.4103/0377-2063.126953 , Improved Indoor Visible Light Communication with PAM and RLS Decision Feedback Equalizer, IETE Journal of Research, Kasun D. Bandara, Pararajasingam Niroopan & Chung Yeon-Ho.
- [Xilinx07] UG334 (v1.0): "Spartan-3A/3AN Starter Kit Board User Guide", Xilinx®, 2007.
- [2]:https://campusvirtual.ull.es/1617/pluginfile.php/318846/mod_resource/content/5/temario/tema3/DYTECI_Tema3a.pdf, Diseño y tecnología de de circuitos impresos, Universidad de La Laguna.
- [3]: https://www.rtelecom.net/es/product/118/osciloscopio-digital-100mhz-1gsa-s-rigol-ds1102d.html
- [4]: "Práctica 4. Diseño en VHDL de un modulador BPSK", Sistemas de Comunicación - Máster en Ingeniería Industrial.
- [5]:http://www.camera-sdk.com/p_54-how-to-implement-circular-buffer-video-recording-in-c-onvif.html
- [6]:https://www.eumus.edu.uy/eme/ensenanza/electivas/dsp/presentaciones/clase10.pdf, EUM, eMe 2011.
- [7]: https://es.wikipedia.org/wiki/FIR_(Finite_Impulse_Response) , Wikipedia.
- [8]: https://www.slideshare.net/RRjZ/d-flip-flop, Laxmi Narain College of Technology.
- [9]: http://queesmemoria.blogspot.com.es/2011/08/que-es-memoria.html, Tecnología de las Computadoras, Prof. Ing. Mauricio Vistosi, 2011.
- [10]:https://es.scribd.com/doc/144968006/Muestreo-y-Retencion, MUESTREO Y RETENCION, Cesar Patiño Ramírez, Wendy Vidal Ríos.
- [Driver]: "PERIPHERAL DRIVERS FOR HIGH-CURRENT SWITCHING AT VERY HIGH SPEEDS", Texas Instruments, DECEMBER 1976 – REVISED SEPTEMBER 1999.

❖ [OPA]: “OPAx354 250-MHz, Rail-to-Rail I/O, CMOS Operational Amplifiers”, Texas Instruments, MARCH 2002–REVISED JUNE 2016.

❖ [Regulador]: “LM78LXX Series 3-Terminal Positive Regulators”, National Semiconductor, May 2003

❖ [ADC]: “12-Bit, 20MHz Sampling ANALOG-TO-DIGITAL CONVERTER”, Burr-Brown Products from Texas Instruments, JANUARY 1997 – REVISED NOVEMBER 2002.

Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO DE FIN DE GRADO:

Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

ANEXOS

Alumnos: David Dorta Pimentel

Sergio Hernández Romera

Tutores: Oswaldo B. González Hernández

José Carlos San Luis Leal

Julio, 2017

Contenido

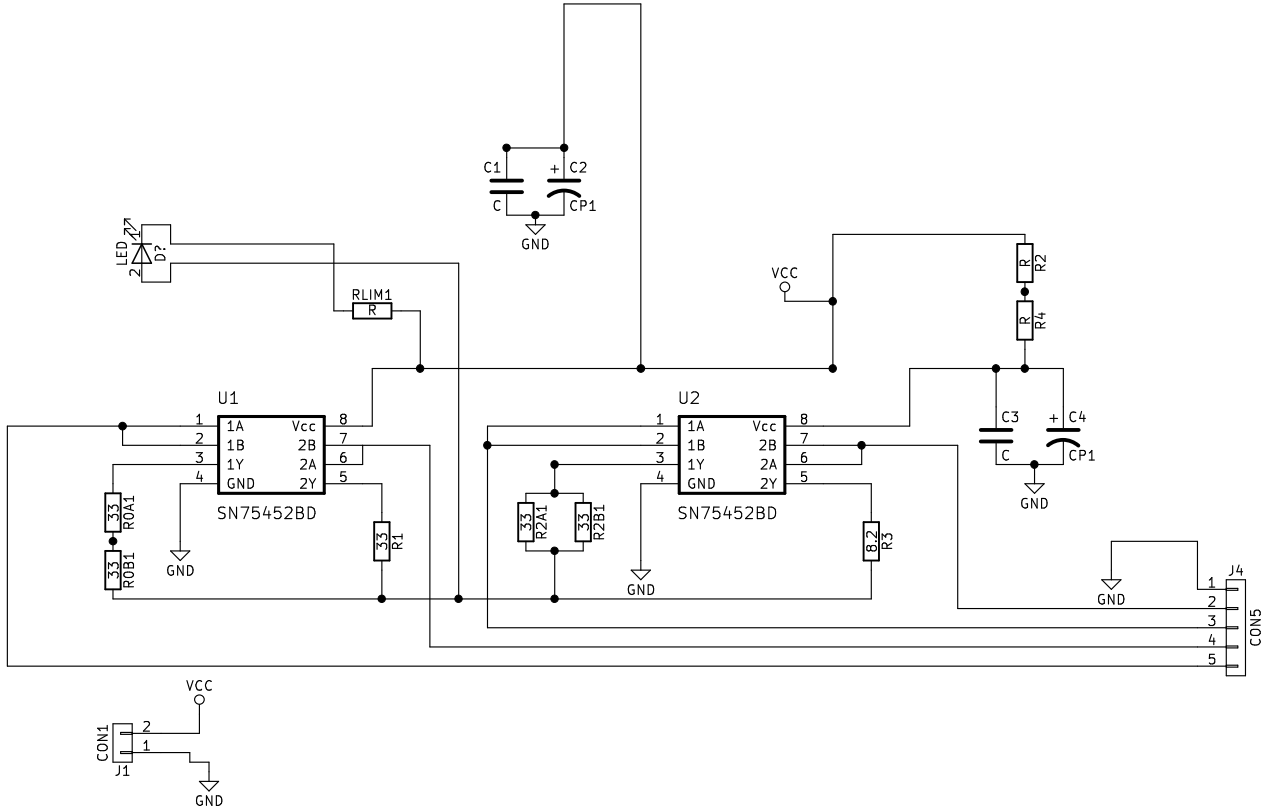
Anexo 1:
Planos

1. Planos y esquemáticos.

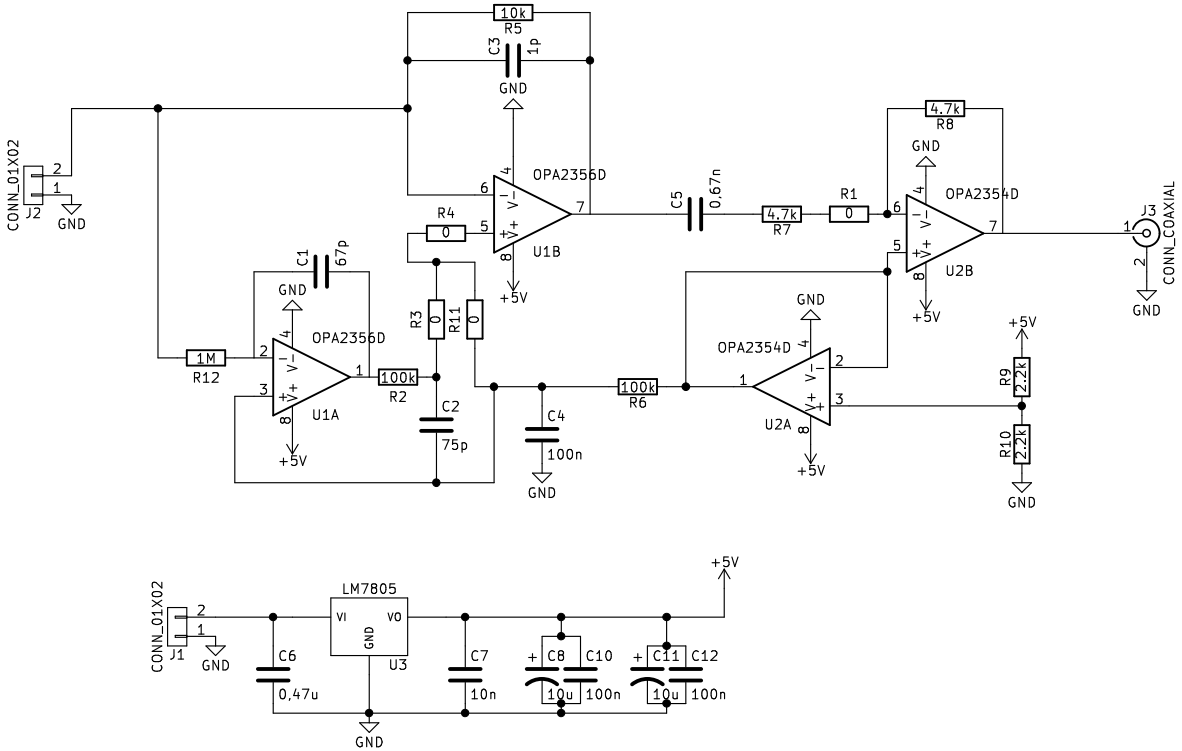
1.1. Esquemático de la PCB emisora

1.2. Esquemático de la PCB receptora

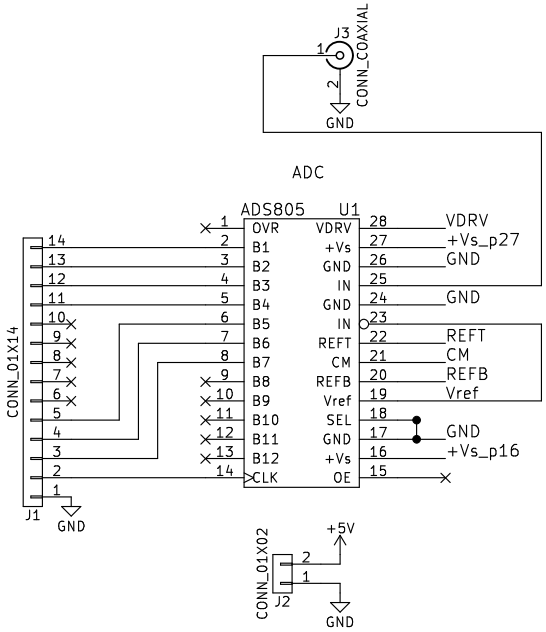
1.3. Esquemático de la PCB del convertidor analógico-digital (ADC).

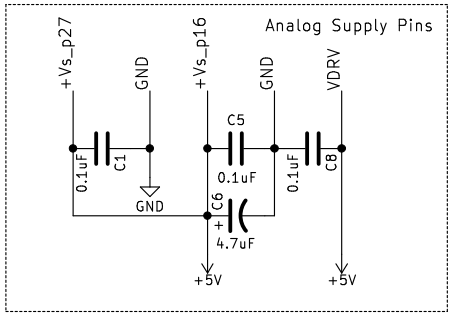


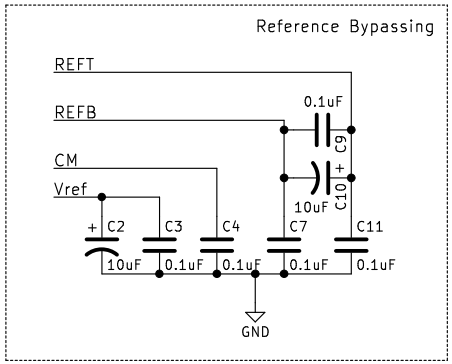
Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	PLACA EMISORA			Nº Plano: 1



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED			
	Fecha	Autor	Universidad de La Laguna
Dibujado	03/07/2017	David Dorta P.	Grado en Ingeniería Electrónica, Industrial y Automática
Comprobado	03/07/2017	Sergio Hdez. R.	
Id. s. normas	UNE-EN-DIN		Universidad de La Laguna
Escala:	PLACA RECEPTORA		Nº Plano: 2







Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	ULL Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	PLACA DEL CONVERSOR ANALÓGICO DIGITAL			Nº Plano: 3

Anexo 2:
Código VHDL

2. Código VHDL empleado.

2.1. Módulo emisor digital.

2.1.1. Generador de números aleatorios.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_ARITH.ALL;
4. use IEEE.STD_LOGIC_UNSIGNED.ALL;
5. use work.constantes.ALL;
6.
7. entity gen_datos is
8.         generic (Nreg : positive := N) ;
9.         port ( clk, reset : in STD_LOGIC;
10.           datos , sinc : out STD_LOGIC) ;
11.   end gen_datos;
12.
13.   architecture Behavioral of gen_datos is
14.
15.       component registro is
16.           port ( clk , preset ,D : in std_logic;
17.                                    Q : out std_logic);
18.       end component registro;
19.
20.       signal sig_xor : std_logic ;
21.       signal Q_int : std_logic_vector (0 to Nreg-1);
22.
23.       begin
24.           Generador_datos : for I in 0 to Nreg-1 generate
25.               Reg00 : if ( I=0) generate
26.                   Reg0 : Registro port map ( clk ,reset
   ,sig_xor,Q_int(0)) ;
27.               end generate ;
28.
29.               Regs : if I>0 generate
30.                   Reg : Registro port map ( clk ,reset,Q_int (I-1)
   ,Q_int (I)) ;
31.               end generate ;
32.           end generate;
33.
34.       datos <= Q_int (Nreg-1);
35.       sig_xor <= Q_int(0) xor Q_int (Nreg-1);
36.       sinc <= and_vector(Q_int);
37.
38.       end Behavioral;
```

2.1.2. Convertidor Serie-Paralelo.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use ieee.std_logic_arith.all;
4. use ieee.std_logic_unsigned.all;
```

```
5.
6. entity Serial_to_Parallel is
7. port( Parallel_Data : out std_logic_vector(2 downto 0);
8.         Clk, Serial_Data, Reset : in std_logic);
9. end Serial_to_Parallel;
10.
11.   architecture Behavioral of Serial_to_Parallel is
12.       signal content: std_logic_vector(2 downto 0);
13.
14.   begin
15.       process(Clk, Reset)
16.        variable cuenta : natural range 0 to 64;
17.       begin
18.           if Reset = '1' then
19.                   content <= "000";
20.                   cuenta := 0;
21.               elsif Clk'event and Clk = '1' then
22.                   cuenta := cuenta + 1;
23.               content <=  Serial_Data & content(2 downto 1);
24.                       if (cuenta = 3) then
25.                           Parallel_Data <= content;
26.                           cuenta := 0;
27.                       end if;
28.           end if;
29.       end process;
30.
31.
32.   end Behavioral;
```

2.1.3. Módulo de relleno.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use work.constantes.ALL;
4.
5. entity Relleno is
6.     Port ( Dato_in : in  std_logic_vector (N-1 downto 0);
7.               reset : in  STD_LOGIC;
8.               estado : out std_logic_vector (1 downto 0);
9.            clk_out : in  STD_LOGIC;
10.                   clk_in : in  STD_LOGIC;
11.                Dato_out : out  std_logic_vector (N-1 downto
   0));
12.      end Relleno;
13.
14.      architecture Behavioral of Relleno is
15.
16.
17.      signal dato_previo : std_logic_vector (N-1 downto 0);
18.      Shared Variable puntero_in : integer := 0; -- iterador
   puntero entrada
19.      Shared Variable puntero_out : integer := 0; -- iterador
   puntero salida
20.      type pila is array(dim_pila-1 downto 0) of std_logic_vector
   (N-1 downto 0); -- Matriz 8x4
21.      signal buffer_datos : pila;
22.
23.      begin
24.
25.      ENTRADA: process (clk_in, reset)
```

```
26.
27.      begin
28.
29.      if (reset = '1') then
30.          puntero_in := 0;
31.          dato_previo <= previo;
32.
33.      elsif ( clk_in'event and clk_in = '0') then
34.
35.          if ( dato_previo = detecta1 and Dato_in = detecta2 )
   then
36.
37.                  buffer_datos(puntero_in) <= relleno1;
38.                  buffer_datos((puntero_in + 1) mod dim_pila) <=
   relleno2;
39.                  buffer_datos((puntero_in + 2) mod dim_pila)<=
   relleno2;
40.
41.                  puntero_in := (puntero_in + 3) mod dim_pila;
42.
43.          else
44.
45.              buffer_datos(puntero_in) <= Dato_in;
46.              puntero_in := (puntero_in + 1) mod dim_pila;
47.
48.          end if;
49.
50.          dato_previo <= Dato_in;
51.
52.      end if;
53.
54.      end process;
55.
56.      WORK: process (clk_out, reset)
57.
58.      type state_type is (st_normal, st_relleno1, st_relleno2);
59.      Variable state, next_state : state_type;
60.
61.      begin
62.
63.      if (reset = '1') then
64.
65.              state := st_normal ;  -- Asignaci□el estado inicial
66.              next_state := st_normal;  -- Asignaci□el estado
   inicial
67.              puntero_out := 0;
68.
69.      elsif (clk_out'event and clk_out = '1') then
70.
71.
72.          if (state = st_normal) then
73.
74.              Dato_out <= buffer_datos(puntero_out);
75.              estado <= "00";
76.          end if;
77.
78.          if (state = st_relleno1) then
79.                  Dato_out <= relleno1;
80.                  estado <= "01";
81.          end if;
82.
```

```
83.          if (state = st_relleno2) then
84.                  Dato_out <= relleno2;
85.                  estado <= "10";
86.          end if;
87.
88.
89.          case (state) is
90.
91.          when st_normal =>
92.                  if ( puntero_in /= puntero_out) then
93.                     next_state := st_normal;
94.                          puntero_out := (puntero_out + 1) mod
   dim_pila;
95.                      else
96.                          next_state := st_relleno1;
97.                      end if;
98.
99.          when st_relleno1 =>
100.                    next_state := st_relleno2;
101.
102.         when st_relleno2 =>
103.                         next_state := st_normal;
104.
105.       end case;
106.
107.         state := next_state;
108.
109.     end if;
110.     end process;
111.
112.     end Behavioral;
```

2.1.4. Ralentizador.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. USE ieee.numeric_std.ALL;
4. use work.constantes.ALL;

5. entity Ralentizador is
6.     Port ( clk_50MHz, rst : in  STD_LOGIC;
7.            clk_out : out  STD_LOGIC;
8.            clk_rec_out : out  STD_LOGIC;
9.            clk_sample : out  STD_LOGIC);
10.     end Ralentizador;
11.
12.     architecture Behavioral of Ralentizador is
13.
14.         signal clk, clk_rec: std_logic := '0';
15.
16.     begin
17.
18.         process (clk_50MHz, rst)
19.
20.             constant cuenta_ralentizador : natural := 504;
21.             variable cuenta,cuenta_rec : natural range 0 to
   cuenta_ralentizador :=0;
22.
23.             begin
24.
25.                 if rst = '1' then
```

```
26.                     cuenta := 0;
27.                     cuenta_rec := 0;
28.                     clk <= '0';
29.                     clk_rec <= '0';
30.
31.                 elsif rising_edge (clk_50MHz) then
32.
33.                     if cuenta < cuenta_ralentizador/(LogL*2)
   then
34.                         clk <= '1';
35.                     else
36.                         clk <= '0';
37.                     end if;
38.
39.                     if cuenta_rec < cuenta_ralentizador/(2*L)
   then
40.                         clk_rec <= '1';
41.                     else
42.                         clk_rec <= '0';
43.                     end if;
44.
45.                     cuenta := (cuenta + 1);
46.
47.                     if cuenta = (cuenta_ralentizador/LogL) then
48.                         cuenta := 0;
49.                     end if;
50.
51.                     cuenta_rec := cuenta_rec + 1;
52.
53.                     if cuenta_rec = (cuenta_ralentizador/L)
   then
54.                         cuenta_rec := 0;
55.                     end if;
56.
57.                 end if;
58.
59.         end process;
60.
61.         clk_out <= clk;
62.         clk_rec_out <= clk_rec;
63.
64.         process(clk, rst)
65.
66.             constant cuenta_ralentizador : natural := 504;
67.             variable cuenta : natural range 0 to
   cuenta_ralentizador :=0;
68.
69.             begin
70.
71.                 if rst = '1' then
72.                     cuenta := 0;
73.                     clk_sample <= '0';
74.
75.                 elsif rising_edge (clk_rec) then
76.
77.                     if cuenta < L/2 then
78.                         clk_sample <= '1';
79.                     else
80.                         clk_sample <= '0';
81.                     end if;
82.
```

```
83.                     cuenta := (cuenta + 1);
84.
85.                     if cuenta = (L) then
86.                         cuenta := 0;
87.                     end if;
88.
89.                 end if;
90.
91.         end process;
92.
93. end Behavioral;
```

2.1.5. Instanciación de los módulos en el bloque emisor.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_ARITH.ALL;
4. use IEEE.STD_LOGIC_UNSIGNED.ALL;
5. USE ieee.numeric_std.ALL;
6. use work.constantes.ALL;
7.
8. entity TRANSMISOR is
9.     Port ( clk_50MHz : in  STD_LOGIC;
10.                rst : in  STD_LOGIC;
11.                   dac_cs : OUT std_logic;
12.                   dac_clr : OUT std_logic;
13.                   spi_mosi : OUT std_logic;
14.                   spi_sck : OUT std_logic;
15.                   clk_rec, clk_out, clk_sample : out  STD_LOGIC;
16.                Datos_paralelos_2 : out std_logic_vector(3 downto
    0);
17.                   led_sinc: OUT std_logic);
18.     end TRANSMISOR;
19.
20.     architecture Behavioral of TRANSMISOR is
21.
22.         --COMPONENTES DEL TRANSMISOR
23.             --Generador de datos:
24.                 COMPONENT gen_datos
25.                 PORT(
26.                     clk : IN std_logic;
27.                     reset : IN std_logic;
28.                     datos : OUT std_logic;
29.                     sinc : OUT std_logic
30.                     );
31.                 END COMPONENT;
32.
33.                 COMPONENT com_dac_A
34.                 PORT(
35.                     reloj : IN std_logic;
36.                     reset : IN std_logic;
37.                     datos_A : IN std_logic_vector(11 downto 0);
38.                     datos_B : IN std_logic_vector(11 downto 0);
39.                     dac_cs : OUT std_logic;
40.                     dac_clr : OUT std_logic;
41.                     spi_mosi : OUT std_logic;
42.                     spi_sck : OUT std_logic
43.                 );
44.                 END COMPONENT;
45.
46.             --Convertidor Serie-Paralelo:
47.                 COMPONENT Serial_to_Parallel
48.                 PORT(
49.                     Clk : IN std_logic;
50.                     Serial_Data : IN std_logic;
51.                     Reset : IN std_logic;
52.                     Parallel_Data : OUT std_logic_vector(2 downto
    0)
53.                     );
54.                 END COMPONENT;
55.
```

```
56.             --Ralentizador:
57.                 COMPONENT Ralentizador
58.                 PORT(
59.                     clk_50MHz : IN std_logic;
60.                     rst : IN std_logic;
61.                     clk_out : OUT std_logic;
62.                     clk_rec_out : OUT std_logic;
63.                     clk_sample : OUT std_logic
64.                     );
65.                 END COMPONENT;
66.
67.         --SEcLES DEL TRANSMISOR
68.             signal Dato_serie, clk : std_logic := '0';
69.             signal datos_paralelos_1 : std_logic_vector(2 downto 0)
   := (others => '0');
70.             signal datos_paralelos_3 : std_logic_vector(11 downto
   0) := (others => '0');
71.
72.     begin
73.
74.                 Inst_com_dac_A: com_dac_A PORT MAP(
75.                     reloj => clk_50MHz,
76.                     reset => rst,
77.                     dac_cs => dac_cs,
78.                     dac_clr => dac_clr,
79.                     spi_mosi => spi_mosi,
80.                     spi_sck => spi_sck,
81.                     datos_A => datos_paralelos_3,
82.                     datos_B => datos_paralelos_3
83.                 );
84.         --INSTANCIACIONES DEL TRANSMISOR
85.             --Generador de datos:
86.                 Inst_gen_datos: gen_datos PORT MAP(
87.                     clk => clk,
88.                     reset => rst,
89.                     datos => Dato_serie,
90.                     sinc => led_sinc
91.                 );
92.
93.             --Convertidor Serie-Paralelo:
94.                 Inst_Serial_to_Parallel: Serial_to_Parallel PORT
   MAP(
95.                     Parallel_Data => datos_paralelos_1,
96.                     Clk => clk,
97.                     Serial_Data => Dato_serie,
98.                     Reset => rst
99.                 );
100.
101.                datos_paralelos_3 <= datos_paralelos_1 &
   "000000000";
102.                datos_paralelos_2 <= datos_paralelos_1 & '0';
103.
104.            --Ralentizador:
105.                Inst_Ralentizador: Ralentizador PORT MAP(
106.                    clk_50MHz => clk_50MHz,
107.                    rst => rst,
108.                    clk_out => clk,
109.                    clk_rec_out => clk_rec,
110.                    clk_sample => clk_sample
111.                );
112.
```

```
113.             clk_out <= clk;
114.
115.     end Behavioral;
```

2.2. Módulo receptor digital.

2.2.1. Filtro FIR.

```
1.  LIBRARY ieee;
2.  USE ieee.std_logic_1164.ALL;
3.  USE ieee.numeric_std.ALL;
4.  USE ieee.std_logic_signed.ALL;
5.  use work.constantes.ALL;
6.
7.  ENTITY FILTRO_FIR IS
8.  PORT (  clk : in std_logic;
9.              rst : in std_logic;
10.                 x : in palabra;
11.                 y : out palabra -- Salida (Factor de crecimiento 0,
    ya que h(k)= 1/N
12.     );
13.     END FILTRO_FIR;
14.
15.     ARCHITECTURE Behavioral OF FILTRO_FIR IS
16.
17.         component Retardador is
18.             port (clk, rst: in std_logic;
19.                     d: in palabra;
20.                     q: out palabra);
21.         end component;
22.
23.         type memoria is array (L - 1 downto 0) of palabra; -- Array
    (palabra, )
24.         signal mem_inter, mem_entrada : memoria; -- Retardos
25.         signal mem_salida : palabra :=(others=>'0');
26.         constant relleno_ceros : std_logic_vector(LogL-1 downto 0)
    := (others=>'0');
27.
28.     begin
29.
30.         Bloque_Retardador: for I in 0 to L - 1 generate
31.             Retardador0 : if (I=0) generate
32.                 Retardador_00 : Retardador port map (clk, rst, x,
    mem_entrada(0));
33.             end generate;
34.             Retardador_1_to_3 : if I>0 generate
35.                 Reg : Retardador port map (clk, rst, mem_entrada(I-
    1), mem_entrada(I));
36.             end generate;
37.         end generate;
38.
39.         Suma_FIR: for k in 0 to L - 1 generate
40.             Suma_0: if (k=0) generate
41.                 Suma_00: mem_inter(0) <=
    palabra(shift_right(signed(mem_entrada(0)),LogL));
42.             end generate;
43.             Suma_otra: if (k>0) generate
44.                 sum_otra: mem_inter(k) <=
    palabra(signed(shift_right(signed(mem_entrada(k)),LogL)) +
    signed(mem_inter(k-1)));
45.             end generate;
46.         end generate Suma_FIR;
```

```
47.           y <= mem_inter(L-1);
48.
49.       end Behavioral;
```

2.2.2. Bloque derivador.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  use IEEE.STD_LOGIC_ARITH.ALL;
4.  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5.  USE ieee.numeric_std.ALL;
6.  use work.constantes.ALL;
7.
8.  entity Bloque_derivador is
9.      Port ( Derivar : in  palabra;
10.                clk : in  STD_LOGIC;
11.                rst : in  STD_LOGIC;
12.                Derivada_2 : out  palabra);
13.     end Bloque_derivador;
14.
15.     architecture Behavioral of Bloque_derivador is
16.
17.         COMPONENT Derivador
18.         PORT(
19.             clk : IN std_logic;
20.             rst : IN std_logic;
21.             Entrada_derivada : IN palabra;
22.             Derivada : OUT palabra
23.             );
24.         END COMPONENT;
25.
26.         signal derivada_1 : palabra := (others=>'0');
27.
28.     begin
29.
30.         Derivador_1: Derivador PORT MAP(
31.             clk => clk,
32.             rst => rst,
33.             Entrada_derivada => Derivar,
34.             Derivada => derivada_1
35.         );
36.
37.         Derivador_2: Derivador PORT MAP(
38.             clk => clk,
39.             rst => rst,
40.             Entrada_derivada => derivada_1,
41.             Derivada => Derivada_2
42.         );
43.
44. end Behavioral;
```

2.2.3. Detector de umbral.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  USE ieee.numeric_std.ALL;
4.  use work.constantes.ALL;
5.
6.  entity Detector_umbral is
7.      Port ( Impulso_in : in  palabra;
```

```
8.            Impulso_out : out  palabra;
9.            clk : in  STD_LOGIC;
10.                 rst : in  STD_LOGIC);
11.     end Detector_umbral;
12.
13.     architecture Behavioral of Detector_umbral is
14.
15.         signal Impulso_abs : palabra := (others=>'0');
16.
17.     begin
18.
19.         Impulso_abs <= palabra(abs(signed(Impulso_in)));
20.
21.         process (clk, rst)
22.
23.         begin
24.
25.             if rst = '1' then
26.                 Impulso_out <= "000000000000";
27.
28.             elsif (clk'event and clk ='1') then
29.
30.                 if (Impulso_abs > umbral) then
31.                     Impulso_out <= Impulso_abs;
32.                 else
33.                     Impulso_out <= "000000000000";
34.                 end if;
35.
36.             end if;
37.
38.         end process;
39.
40. end Behavioral;
```

2.2.4. Generador de reloj.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  USE ieee.numeric_std.ALL;
4.  use work.constantes.ALL;
5.
6.  entity Generador_reloj is
7.      Port ( clk :     in std_logic;
8.             reset : in  STD_LOGIC;
9.           sincronizador  : in  palabra;
10.                Sampleo : out  STD_LOGIC);
11.     end Generador_reloj;
12.
13.     architecture Behavioral of Generador_reloj is
14.         constant cero : palabra := (others => '0');
15.     begin
16.
17.         process(clk, reset)
18.             constant cuenta_ralentizador : natural := 504;
19.             variable cuenta : natural range 0 to
    cuenta_ralentizador :=0;
20.
21.             begin
22.
23.                 if reset = '1' then
24.                     cuenta := 0;
```

```
                    Sampleo <= '0';

                elsif rising_edge (clk) then
                    if (sincronizador /= palabra(to_signed(0,
  nbits))) then
                        cuenta:= 3;
                    end if;

                    if cuenta < L/2 then
                        Sampleo <= '1';
                    else
                        Sampleo <= '0';
                    end if;
                    cuenta := (cuenta + 1);
                    if cuenta = (L) then
                    cuenta := 0;
                    end if;
                end if;
            end process;
end Behavioral;
```

2.2.5. Bloque retardador.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.constantes.ALL;

entity Retardador is
    Port ( clk : in std_logic;
           rst : in std_logic;
           d : in palabra;
           q : out palabra);
  end Retardador;


  architecture Behavioral of Retardador is
      begin
          process (clk) is
          begin
              if rising_edge(clk) then
                  if (rst='1') then
                      q <= "000000000000";
                  else
                      q <= d;
                  end if;
              end if;
          end process;
end Behavioral;
```

2.2.6. *Sample & Hold.*

```
library IEEE;
use work.constantes.ALL;
use IEEE.STD_LOGIC_1164.ALL;

entity SAMPLE is
    Port ( Sample_Hold : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
```

```
8.            Vin : in  palabra;
9.            Vout : out  palabra);
10.     end SAMPLE;
11.
12.     architecture Behavioral of SAMPLE is
13.
14.     begin
15.
16.     process (Sample_Hold, Reset)
17.
18.         variable Vin_hold : palabra :=(others=>'0');
19.
20.     begin
21.         if (Reset = '1') then
22.             Vout <= (others=>'0');
23.
24.        elsif (Sample_Hold'event and Sample_Hold='1') then
25.
26.                 Vin_hold := Vin;
27.
28.        end if;
29.
30.         Vout <= Vin_hold;
31.
32.     end process;
33.
34. end Behavioral;
```

2.2.7. Detector de máximos y mínimos.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  use IEEE.STD_LOGIC_ARITH.ALL;
4.  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5.  use work.constantes.ALL;
6.
7.  entity Max_min is
8.      Port ( Vin : in  palabra;
9.             Vmax : out  palabra;
10.                Vmin : out  palabra;
11.                rst : in  STD_LOGIC;
12.                clk : in  STD_LOGIC);
13.     end Max_min;
14.
15.     architecture Behavioral of Max_min is
16.
17.     begin
18.
19.         process(clk, rst)
20.
21.             variable Valor_max: palabra := (others=>'0');
22.             variable Valor_min: palabra := "011111111111";
23.             variable cuenta: integer := 0;
24.         begin
25.             if rst = '1' then
26.                 Vmax <= "000000000000";
27.                 Vmin <= "011111111111";
28.
29.             elsif (clk'event and clk = '1') then
30.                 cuenta := cuenta + 1;
```

```
31.
32.                 if (Valor_max < Vin) then
33.                     Valor_max := Vin;
34.                 end if;
35.
36.                 if (Valor_min > Vin) then
37.                     Valor_min := Vin;
38.                 end if;
39.                     Vmin <= Valor_min;
40.                     Vmax <= Valor_max;
41.             end if;
42.         end process;
43.
44. end Behavioral;
```

2.2.8. Decodificador.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  use work.constantes.ALL;
4.  USE ieee.numeric_std.ALL;
5.
6.  entity DECODIFICADOR is
7.      Port ( Vin   : in  palabra;
8.             Dato_out : out  STD_LOGIC_VECTOR (3 downto 0);
9.             clk   : in  STD_LOGIC;
10.                reset : in  STD_LOGIC;
11.                   Vmin, Vmax : in palabra
12.                   );
13.     end DECODIFICADOR;
14.
15.     architecture Behavioral of DECODIFICADOR is
16.
17.
18.      type Vector_q is array (2**N -2 downto 0) of palabra;
19.     signal Vqi: Vector_q;
20.     signal q: std_logic_vector (nbits-1 downto 0):= "000000000000";
21.
22.     begin
23.
24.         process (clk, reset)
25.
26.             variable cuenta, valor : natural range 0 to (2**N)-1 :=
    0;
27.         begin
28.             if (reset = '1') then
29.                     Dato_out <= "0000";
30.                     valor := 0;
31.
32.             elsif (clk'event and clk = '1') then
33.                valor := 0;
34.                 for cuenta in 0 to ((2**N)-2) LOOP
35.                     if signed(Vin) > signed(Vqi(cuenta)) then
36.                         valor := cuenta + 1;
37.                     end if;
38.                 end LOOP;
39.
40.                 Dato_out <= std_logic_vector (to_signed(valor,4));
41.             end if;
42.         end process;
```

```
43.
44.         process (Vmin,Vmax,reset)
45.         variable vq : Vector_q;
46.         begin
47.                 q <= std_logic_vector(signed(signed(Vmax) -
    signed(Vmin))/(to_signed((2**nbits) - 1, nbits)));
48.
49.                 vq(0) := palabra(signed(Vmin) +
    signed(shift_right(signed(q),1)));
50.                 for cuenta in 1 to ((2**N)-2) LOOP
51.                     vq(cuenta) := palabra(signed(vq(cuenta-
    1))+signed(q));
52.                 end LOOP;
53.                 Vqi <= vq;
54.         end process;
55.
56. end Behavioral;
```

2.2.9. Instanciación de los módulos en el bloque receptor.

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  USE ieee.numeric_std.ALL;
4.  use work.constantes.ALL;
5.
6.  entity RECEPTOR is
7.      Port ( clk_50MHz : in  STD_LOGIC;
8.             rst : in  STD_LOGIC;
9.                Entrada_receptor: in std_logic_vector (3 downto 0);
10.                     clk_out, clk_rec_out, clk_sample_out: out
    std_logic;
11.                     Entrada_visual: out std_logic_vector (3 downto
    0);
12.                     dac_cs : OUT std_logic;
13.                     dac_clr : OUT std_logic;
14.                     spi_mosi : OUT std_logic;
15.                     spi_sck : OUT std_logic;
16.                 Salida_deco : out  STD_LOGIC_VECTOR (3 downto 0));
17.     end RECEPTOR;
18.
19.     architecture Behavioral of RECEPTOR is
20.
21.         --COMPONENTES DEL RECEPTOR
22.
23.             --Sample & Hold: este m□o se ocupa de muestrear la
    se□obtenida del FIR, obteniendo una salida en forma de escalera que
    pueda ser
24.                 --decodificada en la se□de 4 bits enviada por el
    emisor. El muestreo se realiza cuando se detecta un flanco de subida
    en la entrada
25.                 --"Sample_Hold" la cual es conectada a trav鰻de la
    se "Sample" al generador de reloj, en ese momento se copia el valor
    de la se
```

```
26.                 --de salida del filtro FIR (Vin), valor en el cual
    se mantendr^a se□de salida "Vout" hasta que se vuelva a producir un
    nuevo
27.                 --flanco.
28.                 COMPONENT SAMPLE
29.                 PORT(
30.                     Sample_Hold : IN std_logic;
31.                     Reset : IN std_logic;
32.                     Vin : in palabra;
33.                     Vout : OUT palabra
34.                     );
35.                 END COMPONENT;
36.
37.             --Generador de reloj: en este m□o se genera la se□de
    reloj que se utilizar^ara el muestreo de la se□en el Sample & Hold. El
38.                 --reloj se genera de acuerdo a los impulsos
    obtenidos tras la comparaci□e la salida de los derivadores con el
    umbral, de este modo
39.                 --el Sample & Hold muestrear^uando haya un impulso,
    es decir que lo har^uando se produzca una variaci□n la se□de entrada.
40.                 COMPONENT Generador_reloj
41.                 PORT(
42.                     clk : IN std_logic;
43.                     reset : IN std_logic;
44.                     sincronizador : IN palabra;
45.                     Sampleo : OUT std_logic
46.                     );
47.                 END COMPONENT;
48.
49.             --Bloque de retardadores: para este m□o se ha generado
    un conjunto de registro que nos permiten retardar el reloj generado
    en el
50.                 --anterior m□o para que el muestreo se realice en
    el momento correcto.
51.                 COMPONENT registro
52.                 PORT(
53.                     clk, preset, D : in std_logic;
54.                     Q : out std_logic
55.                     );
56.                 END COMPONENT registro;
57.
58.             --Detector de m̧mos y mmos: el motivo de este m□o es la
    detecci□e los valores m̧mos y mmos de la se□de modo que sean
59.                 --utilizables por el decodificador en su conversi□e
    nbits a N(de 12 a 4 bits).
60.                 COMPONENT Max_min
61.                 PORT(
62.                     Vin : IN palabra;
63.                     rst : IN std_logic;
64.                     clk : IN std_logic;
65.                     Vmax : OUT palabra;
66.                     Vmin : OUT palabra
67.                     );
68.                 END COMPONENT;
69.
70.                     COMPONENT Retardador
71.         PORT(
72.             clk : IN std_logic;
73.             rst : IN std_logic;
74.             d : IN palabra;
```

```
75.             q : OUT palabra
76.             );
77.         END COMPONENT;
78.
79.             --Filtro FIR: el Filtro FIR (Finite Impulse Response)
    es un tipo de filtro digital con el cual se obtiene una respuesta con
    un nmero
80.                 --finito de nmeros no nulos a una entrada impulso.
    Esto se aplica a nuestro caso al detectar los cambios en la entrada
    del mismo,
81.                 --obteniendo una se□de salida formando distintas
    rampas ser□muestreadas en el Sample & Hold.
82.                 COMPONENT FILTRO_FIR
83.                 PORT(
84.                     clk : IN std_logic;
85.                     rst : IN std_logic;
86.                     x : IN palabra;
87.                     y : OUT palabra
88.                     );
89.                 END COMPONENT;
90.
91.             --Decodificador: Este elemento permite la conversi□e la
    se□de nbits bits que ha sido muestreada en la se□de N bits que es
92.                 --enviada por el emisor. Esta conversi□e basa en
    los valores mḫmos y mmos de la se□de salida del Filtro FIR, los
    cuales
93.                 --generan una escala de cuantificaci□n la cual los
    valores mḫmo y mmo coinciden con los obtenidos en el detector de
    mḫmos
94.                 --y mmos. Esta escala se encuentra dividida en dos
    elevado al nmero de bits de salida del decodificador menos 1 (2**N -1
    = 15)
95.                 --niveles de decisi□/span>
96.                 COMPONENT DECODIFICADOR
97.                 PORT(
98.                     Vin : IN palabra;
99.                     clk : IN std_logic;
100.                    reset : IN std_logic;
101.                    Dato_out : OUT std_logic_vector(3 downto 0);
102.                    Vmin, Vmax : in palabra
103.                    );
104.                END COMPONENT;
105.
106.            --Derivadores: Para la generaci□e los impulsos es
    necesario derivar la se□de salida del filtro FIR en dos ocasiones.
    Esto lo
107.                --realizamos mediante dos restadores y una serie de
    bloques retardadores que nos permitan obtener la se□retrasada para
    cumplir
108.                --la aproximaci□igital de la derivada.
109.                COMPONENT Bloque_derivador
110.                PORT(
111.                    Derivar : IN palabra;
112.                    clk : IN std_logic;
113.                    rst : IN std_logic;
114.                    Derivada_2 : OUT palabra
115.                    );
116.                END COMPONENT;
117.
```

```
118.            --Comparador de umbral: Este bloque recibe los impulsos
   del bloque de derivadores y, tras pasar por valor absoluto
   (obteniendo s□impulsos
119.                --positivos), los compara con el valor umbral
   almacenado en el paquete de constantes. De 醬se obtienen los impulsos
   que gu la generaci /span>
120.                --del reloj de muestreo.
121.                COMPONENT Detector_umbral
122.                PORT(
123.                    Impulso_in : IN palabra;
124.                    clk : IN std_logic;
125.                    rst : IN std_logic;
126.                    Impulso_out : OUT palabra
127.                    );
128.                END COMPONENT;
129.
130.        COMPONENT com_dac_A
131.        PORT(
132.            reloj : IN std_logic;
133.            reset : IN std_logic;
134.            datos_A : IN palabra;
135.            datos_B : IN palabra;
136.            dac_cs : OUT std_logic;
137.            dac_clr : OUT std_logic;
138.            spi_mosi : OUT std_logic;
139.            spi_sck : OUT std_logic
140.            );
141.        END COMPONENT;
142.
143.                COMPONENT Ralentizador
144.                PORT(
145.                    clk_50MHz : IN std_logic;
146.                    rst : IN std_logic;
147.                    clk_out : OUT std_logic;
148.                    clk_rec_out : OUT std_logic
149.                    );
150.                END COMPONENT;
151.
152.        --SEcLES DEL RECEPTOR
153.            signal clk_sample_ret : std_logic_vector(L-1 downto 0)
   := (others=>'0');
154.            signal filtro_out, salida_sampleada, Impulsos_1,
   Impulsos_2, Suma_paralelos, retardo_1, retardo_2, Vmax,
   Entrada_palabra, resta_1, resta_2 : palabra := (others=>'0');
155.            signal Vmin : palabra := "011111111111";
156.            signal clk_sampleo, clk_rec : std_logic := '0';
157.
158.            signal Suma_sample, Suma_FIR, Suma_impulsos,
   Suma_impulsos_2: palabra := (others=>'0');
159.
160.    begin
161.        Suma_sample <= palabra(signed(salida_sampleada) +
   to_signed(2**(nbits - 1), nbits));
162.        Suma_FIR <= palabra(signed(filtro_out) +
   to_signed(2**(nbits - 1), nbits));
163.        Suma_impulsos <= palabra(signed(resta_2) +
   to_signed(2**(nbits - 1), nbits));
164.        Suma_impulsos_2 <= palabra(signed(Impulsos_2) +
   to_signed(2**(nbits - 1), nbits));
165.
166.
```

```
167.         Inst_com_dac_A: com_dac_A PORT MAP(
168.             reloj => clk_50MHz,
169.             reset => rst,
170.             dac_cs => dac_cs,
171.             dac_clr => dac_clr,
172.             spi_mosi => spi_mosi,
173.             spi_sck => spi_sck,
174.             datos_A => Suma_sample,
175.             datos_B => Entrada_palabra
176.         );
177.         --INSTANCIACIм DE COMPONENTES
178.             --Sample & Hold: La se□a muestrear es la salida del FIR
    (filtro_out). Este muestreo se realiza a partir de la se□de reloj
179.                 --retrasada del generador de reloj
    (clk_sample_ret(5), el nmero del vector corresponde al nmero de
    retrasos realizados).
180.                 Inst_SAMPLE: SAMPLE PORT MAP(
181.                     Sample_Hold => clk_sample_ret(0),
182.                     Reset => rst,
183.                     Vin => filtro_out,
184.                     Vout => salida_sampleada
185.                 );
186.
187.             --Bloque Retardador: La se□de reloj obtenida del
    generador de reloj se retarda tantas veces como sea necesario
    obteniendo la
188.                 --se□de muestreo del Sample & Hold.
189.                 Bloque_Retardador: for I in 0 to L-1 generate
190.                     Retardador0 : if (I=0) generate
191.                         Retardador_00 : registro port map (clk_rec,
    rst, clk_sampleo,clk_sample_ret(0));
192.                     end generate;
193.                     Retardador_otro : if I>0 generate
194.                         Reg : registro port map (clk_rec, rst,
    clk_sample_ret(I-1),clk_sample_ret(I));
195.                     end generate;
196.                 end generate;
197.
198.             --Generador de reloj: Este m□o recibe los impulsos
    sincronizadores de la comparaci□on el umbral y a partir de los mismo
    genera
199.                 --la se□de reloj.
200.                 Inst_Generador_reloj: Generador_reloj PORT MAP(
201.                     clk => clk_rec,
202.                     reset => rst,
203.                     sincronizador => Impulsos_2,
204.                     Sampleo => clk_sampleo
205.                 );
206.
207.             --Filtro FIR: A partir de la se□captada por el receptor
    genera otra se□ basada en esta misma, pero a trav鰻de distintas
    rampas.
208.                 --Para introducir la se captada en el filtro es
    necesario sumarle a la misma el valor de dos elevado a nbits menos 1
    (2**(nbits-1))
209.                 Entrada_palabra <= palabra(signed(Entrada_receptor)
    & to_signed(0, nbits - 4));
210.                 Suma_Paralelos <= palabra(signed(Entrada_palabra) +
    to_signed(2**(nbits - 1), nbits));
211.
212.                 Inst_FILTRO_FIR: FILTRO_FIR PORT MAP(
```

```
213.                    clk => clk_rec,
214.                    rst => rst,
215.                    x => Suma_Paralelos,
216.                    y => filtro_out
217.                );
218.
219.            --Decodificador: A este m□o se le introduce la
   se□muestreada por el Sample & Hold y la convierte en la salida de
   nuestro bloque receptor
220.                Inst_DECODIFICADOR: DECODIFICADOR PORT MAP(
221.                    Vin => salida_sampleada,
222.                    Dato_out => salida_deco,
223.                    clk => clk_rec,
224.                    reset => rst,
225.                    Vmax => Vmax,
226.                    Vmin => Vmin
227.                );
228.
229.            --Detector de m̧mos y mmos: Este m□o analizar^a se□de
   salida del FIR, a partir de la cual generar^os m̧mos y mmos valores
   de
230.                --la funci□ue servir□de gupara el decodificador.
231.                Inst_Max_min: Max_min PORT MAP(
232.                    Vin => filtro_out,
233.                    Vmax => Vmax,
234.                    Vmin => Vmin,
235.                    rst => rst,
236.                    clk => clk_rec
237.                );
238.
239.            --Derivadores: El bloque de derivadores generar^os
   impulsos, que deber□se comparados con el umbral, a partir de la
   se□del FIR
240.                Inst_Bloque_derivador: Bloque_derivador PORT MAP(
241.                    Derivar => filtro_out,
242.                    clk => clk_rec,
243.                    rst => rst,
244.                    Derivada_2 => Impulsos_1
245.                );
246.
247.            --Comparador de umbral: Este bloque recibe los impulsos
   del bloque de derivadores y, tras pasar por valor absoluto
   (obteniendo s□impulsos
248.                --positivos), los compara con el valor umbral
   almacenado en el paquete de constantes. De 醬se obtienen los impulsos
   que gu la generaci /span>
249.                --del reloj de muestreo.
250.                Inst_Detector_umbral: Detector_umbral PORT MAP(
251.                    Impulso_in => Impulsos_1,
252.                    Impulso_out => Impulsos_2,
253.                    clk => clk_rec,
254.                    rst => rst
255.                );
256.
257.                Inst_Ralentizador: Ralentizador PORT MAP(
258.                    clk_50MHz => clk_50MHz,
259.                    rst => rst,
260.                    clk_out => clk_out,
261.                    clk_rec_out => clk_rec
262.                );
```

```
263.
264.
265.         Retardador_1: Retardador PORT MAP(
266.             clk => clk_rec,
267.             rst => rst,
268.             d => filtro_out,
269.             q => retardo_1
270.         );
271.
272.         Retardador_2: Retardador PORT MAP(
273.             clk => clk_rec,
274.             rst => rst,
275.             d => resta_1,
276.             q => retardo_2
277.         );
278.                 clk_rec_out <= clk_rec;
279.                 clk_sample_out <= clk_sampleo;
280.                 Entrada_visual <= Entrada_receptor;
281.
282.     end Behavioral;
283.
```

2.3. Interfaz con el DAC.

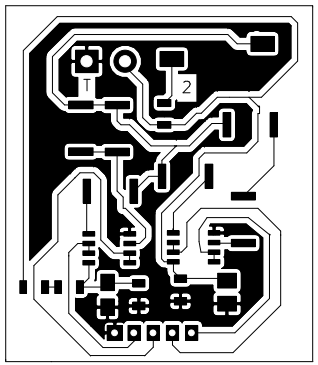
```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_ARITH.ALL;
4. use IEEE.STD_LOGIC_UNSIGNED.ALL;
5. use work.constantes.all;
6. entity com_dac_A is
7.         Port (reloj : in STD_LOGIC;
8.                 reset : in STD_LOGIC;
9.                 dac_cs : out STD_LOGIC;
10.                     dac_clr : out STD_LOGIC;
11.                     spi_mosi : out STD_LOGIC;
12.                     spi_sck : out STD_LOGIC;
13.                     datos_A : in palabra;
14.                     datos_B : IN palabra);
15.     end com_dac_A ;
16.
17.     architecture Behavioral of com_dac_A is
18.         signal address : std_logic_vector(3 downto 0) := (others
    =>'0');
19.         signal memoria_dac : std_logic_vector(31 downto 0) :=
    (others =>'0');
20.         signal datos: palabra := (others =>'0');
21.         shared variable cuenta : natural range 0 to 64 := 0;
22.     begin
23.         WORK: process (reloj, reset)
24.
25.         type state_type is (canal_A, canal_B);
26.         Variable state, next_state : state_type;
27.
28.             begin
29.
30.                 if (reset = '1') then
31.
32.                     state := canal_A ;  -- Asignaci□el estado
    inicial
33.                     next_state := canal_A;  -- Asignaci□el estado
    inicial
34.
```

```
35.                 elsif (reloj'event and reloj = '0') then
36.
37.                     if (state = canal_A) then
38.
39.                         address <= "0000";
40.                         datos <= datos_A;
41.                     end if;
42.
43.                     if (state = canal_B) then
44.                         address <= "0001";
45.                         datos <= datos_B;
46.                     end if;
47.
48.                     case (state) is
49.
50.                             when canal_A =>
51.                                 if (cuenta = 63) then
52.                                     next_state := canal_B;
53.                                 end if;
54.
55.                             when canal_B =>
56.                                 if (cuenta = 63) then
57.                                     next_state := canal_A;
58.                                 end if;
59.                     end case;
60.
61.                     state := next_state;
62.
63.                 end if;
64.         end process;
65.
66.         process (reloj ,reset)
67.         begin
68.             if reset = '1' then
69.                 --Establece  el comando por defecto
70.                 memoria_dac(23 downto 20) <= "0011";
71.                 --Direcciona al DAC A por defecto
72.                 memoria_dac(19 downto 16) <= address;
73.                 cuenta := 0;
74.                 dac_cs <= '1';
75.             elsif reloj 'event and reloj = '1' then
76.                 cuenta := cuenta + 1;
77.                 case cuenta is
78.                     when 1 => dac_cs <= '0';
79.                         --Carga los datos
80.                         memoria_dac(15 downto 4) <=
   std_logic_vector(signed(datos));
81.                         --Direcciona al DAC A
82.                         memoria_dac(19 downto 16) <= address;
83.                         spi_mosi <= memoria_dac (31);
84.
85.                     when 33 => dac_cs <= '1';
86.
87.                     when 64 => cuenta:= 0;
88.
89.                     when others =>
90.                         spi_mosi <= memoria_dac (31-((cuenta -1)
   mod 32));
91.                 end case;
92.             end if;
93.         end process;
```

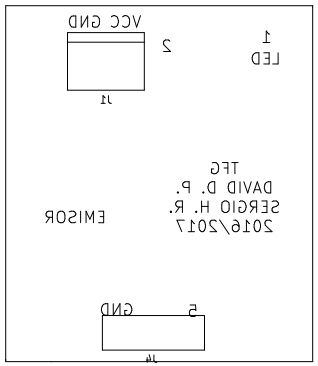
```
94.          spi_sck <= not(reloj);
95.          dac_clr <= not(reset);
96.      end Behavioral;
```

Anexo 3: Fotolitos

3. Fotolitos de los circuitos impresos fabricados.



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	FOTOLITO PLACA EMISORA. CARA DE PISTAS			Nº Plano: 4



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED

	Fecha	Autor
Dibujado	03/07/2017	David Dorta P.
Comprobado	03/07/2017	Sergio Hdez. R.
Id. s. normas	UNE-EN-DIN	

Universidad de La Laguna

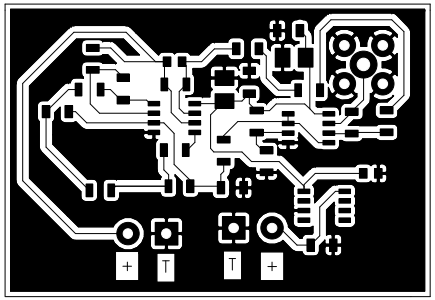
Grado en Ingeniería Electrónica, Industrial y Automática

Universidad de La Laguna

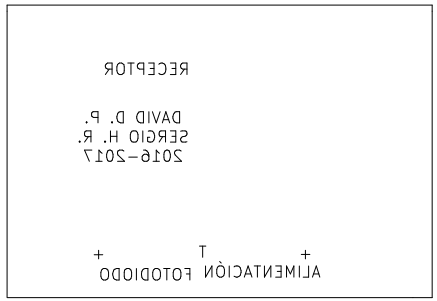
Escala:

FOTOLITO PLACA EMISORA.
CARA DE SERIGRAFÍA.

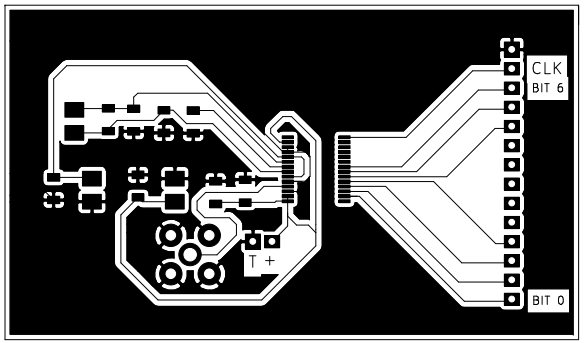
Nº Plano: 5



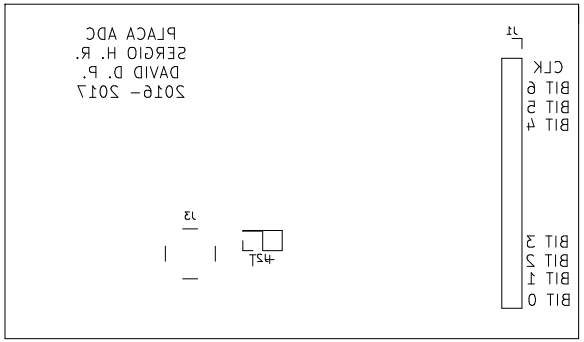
Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	ULL Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	FOTOLITO PLACA RECEPTORA. CARA DE PISTAS			Nº Plano: 6



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	FOTOLITO PLACA RECEPTORA. CARA DE SERIGRAFÍA.			Nº Plano: 7



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	FOTOLITO PLACA DEL CONVERSOR ANALÓGICO DITGITAL. CARA DE PISTAS.			Nº Plano: 8



Diseño en FPGA de un sistema de comunicación PAM mediante lámparas de LED				
	Fecha	Autor	Universidad de La Laguna	Grado en Ingeniería Electrónica, Industrial y Automática
Dibujado	03/07/2017	David Dorta P.		
Comprobado	03/07/2017	Sergio Hdez. R.		
Id. s. normas	UNE-EN-DIN			Universidad de La Laguna
Escala:	FOTOLITO PLACA DEL CONVERSOR ANALÓGICO DITGITAL. CARA DE SERIGRAFÍA.			Nº Plano: 9

Jameco Part Number 758583

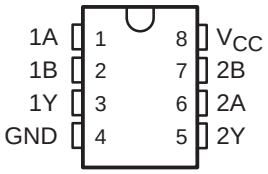
PERIPHERAL DRIVERS FOR HIGH-CURRENT SWITCHING AT VERY HIGH SPEEDS

- **Characterized for Use to 300 mA**
- **High-Voltage Outputs**
- **No Output Latch-Up at 20 V (After Conducting 300 mA)**
- **High-Speed Switching**
- **Circuit Flexibility for Varied Applications**
- **TTL-Compatible Diode-Clamped Inputs**
- **Standard Supply Voltages**
- **Plastic DIP (P) With Copper Lead Frame Provides Cooler Operation and Improved Reliability**
- **Package Options Include Plastic Small-Outline Packages, Ceramic Chip Carriers, and Standard Plastic and Ceramic 300-mil DIPs**

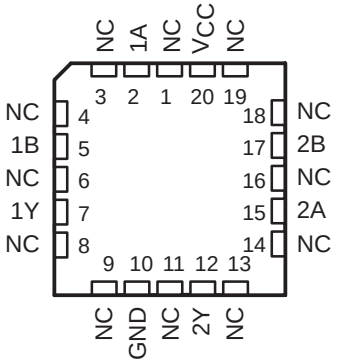
SUMMARY OF DEVICES

DEVICE	LOGIC OF COMPLETE CIRCUIT	PACKAGES
SN55451B	AND	FK, JG
SN55452B	NAND	JG
SN55453B	OR	FK, JG
SN55454B	NOR	JG
SN75451B	AND	D, P
SN75452B	NAND	D, P
SN75453B	OR	D, P
SN75454B	NOR	D, P

SN55451B, SN55452B, SN55453B, SN55454B . . . JG PACKAGE
SN75451B, SN75452B, SN75453B, SN75454B . . . D OR P PACKAGE
(TOP VIEW)



SN55451B, SN55452B
SN55453B, SN55454B . . . FK PACKAGE
(TOP VIEW)



NC – No internal connection

description

The SN55451B through SN55454B and SN75451B through SN75454B are dual peripheral drivers designed for use in systems that employ TTL logic. This family is functionally interchangeable with and replaces the SN75450 family and the SN75450A family devices manufactured previously. The speed of the devices is equal to that of the SN75450 family, and the parts are designed to ensure freedom from latch-up. Diode-clamped inputs simplify circuit design. Typical applications include high-speed logic buffers, power drivers, relay drivers, lamp drivers, MOS drivers, line drivers, and memory drivers.

The SN55451B/SN75451B, SN55452B/SN75452B, SN55453B/SN75453B, and SN55454B/SN75454B are dual peripheral AND, NAND, OR, and NOR drivers, respectively (assuming positive logic), with the output of the logic gates internally connected to the bases of the npn output transistors.

The SN55’ drivers are characterized for operation over the full military range of –55°C to 125°C. The SN75’ drivers are characterized for operation from 0°C to 70°C.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

		SN55'	SN75'	UNIT
Supply voltage, V_{CC} (see Note 1)		7	7	V
Input voltage, V_I		5.5	5.5	V
Inter-emitter voltage (see Note 2)		5.5	5.5	V
Off-state output voltage, V_O		30	30	V
Continuous collector or output current, I_{OK} (see Note 3)		400	400	mA
Peak collector or output current, I_I ($t_W \le 10$ ms, duty cycle $\le 50\%$, see Note 4)		500	500	mA
Continuous total power dissipation		See Dissipation Rating Table		
Operating free-air temperature range, T_A		–55 to 125	0 to 70	°C
Storage temperature range, T_{stg}		–65 to 150	–65 to 150	°C
Case temperature for 60 seconds	FK package	260		°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds	JG package	300		°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds	D or P package		260	°C

NOTES: 1. Voltage values are with respect to network GND, unless otherwise specified.
2. This is the voltage between two emitters of a multiple-emitter transistor.
3. This value applies when the base-emitter resistance (R_{BE}) is equal to or less than 500 Ω.
4. Both halves of these dual circuits may conduct rated current simultaneously; however, power dissipation averaged over a short time interval must fall within the continuous dissipation rating.

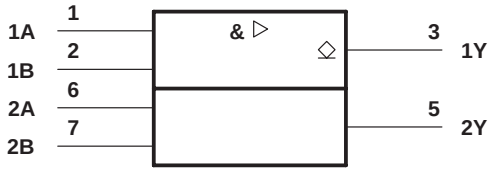
DISSIPATION RATING TABLE

PACKAGE	$T_A \le 25°C$ POWER RATING	DERATING FACTOR ABOVE $T_A = 25°C$	$T_A = 70°C$ POWER RATING	$T_A = 125°C$ POWER RATING
D	725 mW	5.8 mW/°C	464 mW	—
FK	1375 mW	11.0 mW/°C	880 mW	275 mW
JG	1050 mW	8.4 mW/°C	672 mW	210 mW
P	1000 mW	8.0 mW/°C	640 mW	—

recommended operating conditions

	SN55'			SN75'			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level input voltage, V_{IH}	2			2			V
Low-level input voltage, V_{IL}			0.8			0.8	V
Operating free-air temperature, T_A	–55		125	0		70	°C

logic symbol†



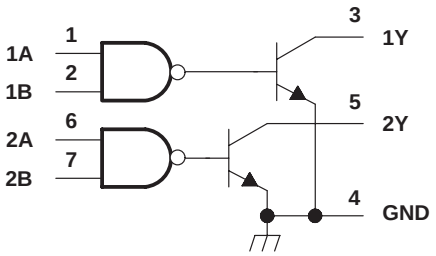
† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC publication 617-12.
Pin numbers shown are for the D, JG, and P packages.

FUNCTION TABLE (each driver)

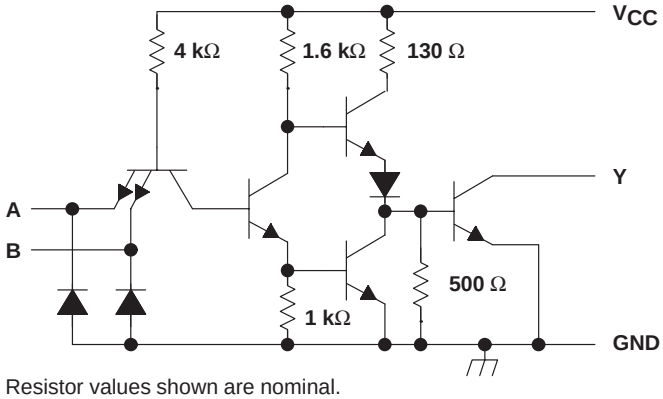
A	B	Y
L	L	L (on state)
L	H	L (on state)
H	L	L (on state)
H	H	H (off state)

positive logic: $Y = AB$ or $\overline{A}+\overline{B}$

logic diagram (positive logic)



schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

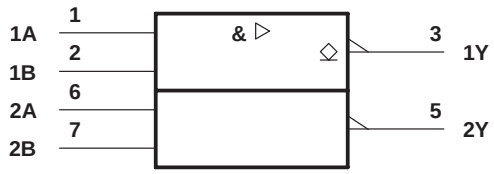
PARAMETER		TEST CONDITIONS‡		SN55451B			SN75451B			UNIT
				MIN	TYP§	MAX	MIN	TYP§	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = 100 mA	V_{IL} = 0.8 V,		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, I_{OL} = 300 mA	V_{IL} = 0.8 V,		0.5	0.8		0.5	0.7	
I_{OH}	High-level output current	V_{CC} = MIN, V_{OH} = 30 V	V_{IH} = MIN,			300			100	µA
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.4 V		–1	–1.6		–1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX,	V_I = 5 V		7	11		7	11	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX,	V_I = 0		52	65		52	65	mA

‡ For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
§ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, R_L = 50 Ω,	C_L = 15 pF, See Figure 1		18	25	ns
t_{PHL}	Propagation delay time, high-to-low-level output					18	25	
t_{TLH}	Transition time, low-to-high-level output					5	8	
t_{THL}	Transition time, high-to-low-level output					7	12	
V_{OH}	High-level output voltage after switching	SN55451B	V_S = 20 V, See Figure 2	$I_O \approx$ 300 mA,	V_S–6.5			mV
		SN75451B			V_S–6.5			

logic symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC publication 617-12.

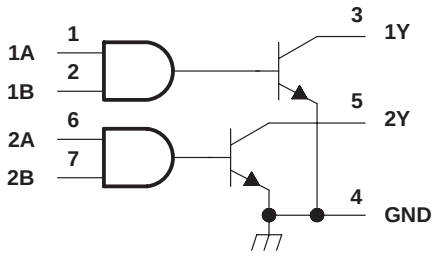
Pin numbers shown are for the D, JG, and P packages.

FUNCTION TABLE (each driver)

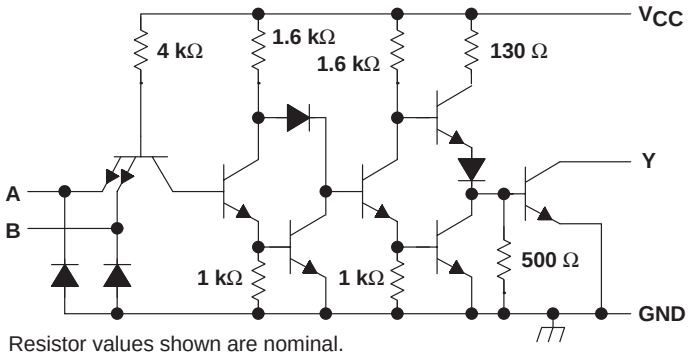
A	B	Y
L	L	H (off state)
L	H	H (off state)
H	L	H (off state)
H	H	L (on state)

positive logic:
$Y = \overline{AB}$ or $\overline{A}+\overline{B}$

logic diagram (positive logic)



schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

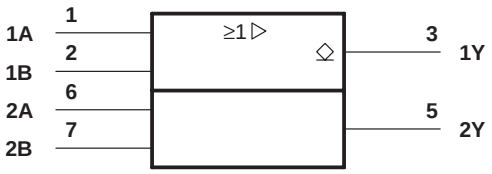
PARAMETER		TEST CONDITIONS‡	SN55452B			SN75452B			UNIT
			MIN	TYP§	MAX	MIN	TYP§	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN, I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IH} = MIN, I_{OL} = 100 mA		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, V_{IH} = MIN, I_{OL} = 300 mA		0.5	0.8		0.5	0.7	
I_{OH}	High-level output current	V_{CC} = MIN, V_{IL} = 0.8 V, V_{OH} = 30 V			300			100	µA
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4 V		–1.1	–1.6		–1.1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX, V_I = 0		11	14		11	14	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX, V_I = 5 V		56	71		56	71	mA

‡ For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
§ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

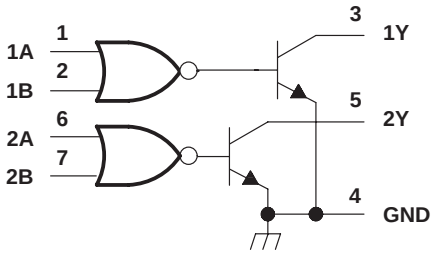
PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, C_L = 15 pF, R_L = 50 Ω, See Figure 1		26	35	ns
t_{PHL}	Propagation delay time, high-to-low-level output				24	35	
t_{TLH}	Transition time, low-to-high-level output				5	8	
t_{THL}	Transition time, high-to-low-level output				7	12	
V_{OH}	High-level output voltage after switching	SN55452B	V_S = 20 V, $I_O \approx$ 300 mA, See Figure 2		V_S–6.5		mV
		SN75452B		V_S–6.5			

logic symbol†



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC publication 617-12.
Pin numbers shown are for the D, JG, and P packages.

logic diagram (positive logic)

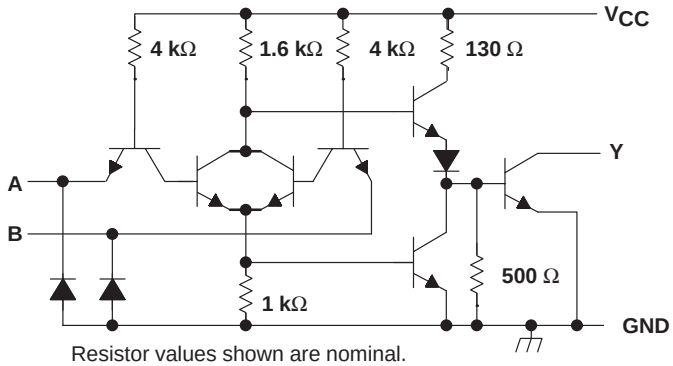


FUNCTION TABLE (each driver)

A	B	Y
L	L	L (on state)
L	H	H (off state)
H	L	H (off state)
H	H	H (off state)

positive logic: $Y = A+B$ or $Y = \overline{\overline{A}\,\overline{B}}$

schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

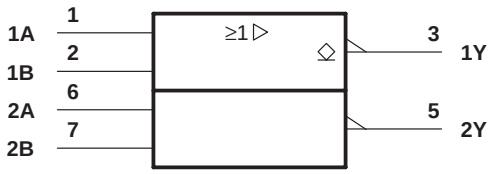
PARAMETER		TEST CONDITIONS‡		SN55453B			SN75453B			UNIT
				MIN	TYP§	MAX	MIN	TYP§	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = 100 mA	V_{IL} = 0.8 V,		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, I_{OL} = 300 mA	V_{IL} = 0.8 V,		0.5	0.8		0.5	0.7	
I_{OH}	High-level output current	V_{CC} = MIN, V_{OH} = 30 V	V_{IH} = MIN,			300			100	µA
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.4 V		–1	–1.6		–1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX,	V_I = 5 V		8	11		8	11	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX,	V_I = 0		54	68		54	68	mA

‡ For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
§ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, R_L = 50 Ω,	C_L = 15 pF, See Figure 1		18	25	ns
t_{PHL}	Propagation delay time, high-to-low-level output					18	25	
t_{TLH}	Transition time, low-to-high-level output					5	8	
t_{THL}	Transition time, high-to-low-level output					7	12	
V_{OH}	High-level output voltage after switching	SN55453B	V_S = 20 V, See Figure 2	$I_O \approx$ 300 mA,	V_S–6.5			mV
		SN75453B			V_S–6.5			

logic symbol†



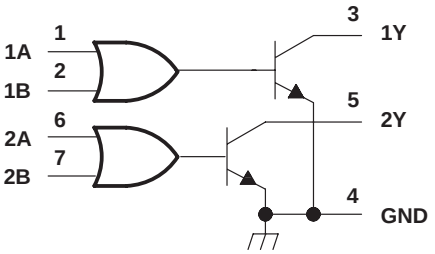
† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC publication 617-12.
Pin numbers shown are for the D, JG, and P packages.

FUNCTION TABLE (each driver)

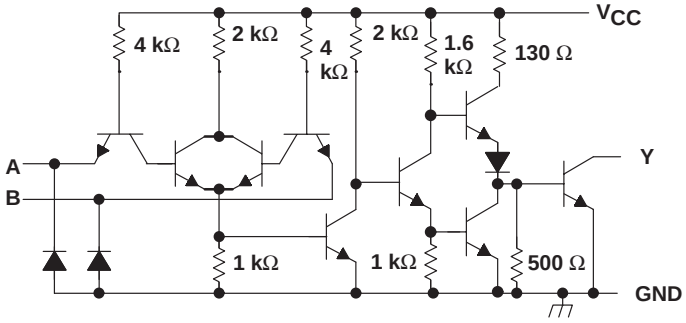
A	B	Y
L	L	H (off state)
L	H	L (on state)
H	L	L (on state)
H	H	L (on state)

positive logic:
$Y = \overline{A+B}$ or $\overline{AB}$

logic diagram (positive logic)



schematic (each driver)



Resistor values shown are nominal.

electrical characteristics over recommended operating free-air temperature range

PARAMETER		TEST CONDITIONS‡		SN55454B			SN75454B			UNIT
				MIN	TYP§	MAX	MIN	TYP§	MAX	
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = –12 mA		–1.2	–1.5		–1.2	–1.5	V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = 100 mA	V_{IH} = MIN,		0.25	0.5		0.25	0.4	V
		V_{CC} = MIN, I_{OL} = 300 mA	V_{IH} = MIN,		0.5	0.8		0.5	0.7	
I_{OH}	High-level output current	V_{CC} = MIN, V_{OH} = 30 V	V_{IL} = 0.8 V,			300			100	µA
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 5.5 V			1			1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.4 V			40			40	µA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.4 V		–1	–1.6		–1	–1.6	mA
I_{CCH}	Supply current, outputs high	V_{CC} = MAX,	V_I = 0		13	17		13	17	mA
I_{CCL}	Supply current, outputs low	V_{CC} = MAX,	V_I = 5 V		61	79		61	79	mA

‡ For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
§ All typical values are at V_{CC} = 5 V, T_A = 25°C.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output		$I_O \approx$ 200 mA, R_L = 50 Ω,	C_L = 15 pF, See Figure 1		27	35	ns
t_{PHL}	Propagation delay time, high-to-low-level output					24	35	
t_{TLH}	Transition time, low-to-high-level output					5	8	
t_{THL}	Transition time, high-to-low-level output					7	12	
V_{OH}	High-level output voltage after switching	SN55454B	V_S = 20 V, See Figure 2	$I_O \approx$ 300 mA,	V_S–6.5			mV
		SN75454B			V_S–6.5			

National Semiconductor

May 2003

LM78LXX Series
3-Terminal Positive Regulators

General Description

The LM78LXX series of three terminal positive regulators is available with several fixed output voltages making them useful in a wide range of applications. When used as a zener diode/resistor combination replacement, the LM78LXX usually results in an effective output impedance improvement of two orders of magnitude, and lower quiescent current. These regulators can provide local on card regulation, eliminating the distribution problems associated with single point regulation. The voltages available allow the LM78LXX to be used in logic systems, instrumentation, HiFi, and other solid state electronic equipment.

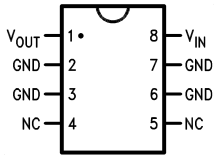
The LM78LXX is available in the plastic TO-92 (Z) package, the plastic SO-8 (M) package and a chip sized package (8-Bump micro SMD) using National's micro SMD package technology. With adequate heat sinking the regulator can deliver 100mA output current. Current limiting is included to limit the peak output current to a safe value. Safe area protection for the output transistors is provided to limit internal power dissipation. If internal power dissipation becomes too high for the heat sinking provided, the thermal shutdown circuit takes over preventing the IC from overheating.

Features

- LM78L05 in micro SMD package
- Output voltage tolerances of ±5% over the temperature range
- Output current of 100mA
- Internal thermal overload protection
- Output transistor safe area protection
- Internal short circuit current limit
- Available in plastic TO-92 and plastic SO-8 low profile packages
- No external components
- Output voltages of 5.0V, 6.2V, 8.2V, 9.0V, 12V, 15V
- See AN-1112 for micro SMD considerations

Connection Diagrams

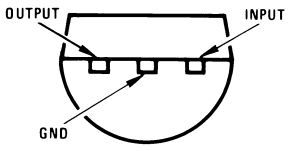
SO-8 Plastic (M)
(Narrow Body)



00774402

Top View

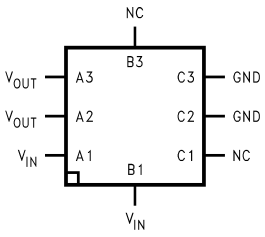
(TO-92)
Plastic Package (Z)



00774403

Bottom View

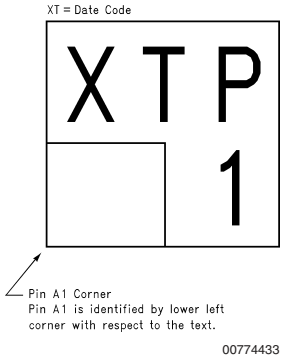
8-Bump micro SMD



00774424

Top View
(Bump Side Down)

micro SMD Marking Orientation



00774433

Top View

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Power Dissipation (Note 5)	Internally Limited
Input Voltage	35V
Storage Temperature	−65˚C to +150˚C
ESD Susceptibility (Note 2)	1kV
Operating Junction Temperature	
SO-8, TO-92	0˚C to 125˚C
micro SMD	−40˚C to 85˚C
Soldering Information	
Infrared or Convection (20 sec.)	235˚C
Wave Soldering (10 sec.)	260˚C (lead time)

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and −40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF.

LM78L05

Unless otherwise specified, V_{IN} = 10V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		4.8	5	5.2	V
		7V ≤ V_{IN} ≤ 20V 1mA ≤ I_O ≤ 40mA (Note 3)	**4.75**		**5.25**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**4.75**		**5.25**	
ΔV_O	Line Regulation	7V ≤ V_{IN} ≤ 20V		18	75	mV
		8V ≤ V_{IN} ≤ 20V		10	54	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		20	60	
		1mA ≤ I_O ≤ 40mA		5	30	
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	8V ≤ V_{IN} ≤ 20V			**1.0**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		40		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 8V ≤ V_{IN} ≤ 16V	47	62		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		−0.65		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			6.7	7	V
θ_{JA}	Thermal Resistance (8-Bump micro SMD)			230.9		˚C/W

LM78L62AC

Unless otherwise specified, V_{IN} = 12V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		5.95	6.2	6.45	V
		8.5V ≤ V_{IN} ≤ 20V 1mA ≤ I_O ≤ 40mA (Note 3)	**5.9**		**6.5**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**5.9**		**6.5**	

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L62AC (Continued)

Unless otherwise specified, V_{IN} = 12V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ΔV_O	Line Regulation	8.5V ≤ V_{IN} ≤ 20V		65	175	mV
		9V ≤ V_{IN} ≤ 20V		55	125	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		13	80	
		1mA ≤ I_O ≤ 40mA		6	40	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	8V ≤ V_{IN} ≤ 20V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		50		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 10V ≤ V_{IN} ≤ 20V	40	46		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.75		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			7.9		V

LM78L82AC

Unless otherwise specified, V_{IN} = 14V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		7.87	8.2	8.53	V
		11V ≤ V_{IN} ≤ 23V 1mA ≤ I_O ≤ 40mA (Note 3)	**7.8**		**8.6**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**7.8**		**8.6**	
ΔV_O	Line Regulation	11V ≤ V_{IN} ≤ 23V		80	175	mV
		12V ≤ V_{IN} ≤ 23V		70	125	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		15	80	
		1mA ≤ I_O ≤ 40mA		8	40	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	12V ≤ V_{IN} ≤ 23V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage	f = 10 Hz to 100 kHz (Note 4)		60		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 12V ≤ V_{IN} ≤ 22V	39	45		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.8		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			9.9		V

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L09AC

Unless otherwise specified, V_{IN} = 15V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		8.64	9.0	9.36	V
		11.5V ≤ V_{IN} ≤ 24V 1mA ≤ I_O ≤ 40mA (Note 3)	**8.55**		**9.45**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**8.55**		**9.45**	
ΔV_O	Line Regulation	11.5V ≤ V_{IN} ≤ 24V		100	200	mV
		13V ≤ V_{IN} ≤ 24V		90	150	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		20	90	
		1mA ≤ I_O ≤ 40mA		10	45	
I_Q	Quiescent Current			2	5.5	mA
ΔI_Q	Quiescent Current Change	11.5V ≤ V_{IN} ≤ 24V			**1.5**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage			70		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 15V ≤ V_{IN} ≤ 25V	38	44		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–0.9		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			10.7		V

LM78L12AC

Unless otherwise specified, V_{IN} = 19V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		11.5	12	12.5	V
		14.5V ≤ V_{IN} ≤ 27V 1mA ≤ I_O ≤ 40mA (Note 3)	**11.4**		**12.6**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**11.4**		**12.6**	
ΔV_O	Line Regulation	14.5V ≤ V_{IN} ≤ 27V		30	180	mV
		16V ≤ V_{IN} ≤ 27V		20	110	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		30	100	
		1mA ≤ I_O ≤ 40mA		10	50	
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	16V ≤ V_{IN} ≤ 27V			**1**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage			80		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 15V ≤ V_{IN} ≤ 25	40	54		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–1.0		mV/˚C

LM78LXX Electrical Characteristics

Limits in standard typeface are for T_J = 25˚C, **Bold typeface applies over 0˚C to 125˚C for SO-8 and TO-92 packages, and –40˚C to 85˚C for micro SMD package.** Limits are guaranteed by production testing or correlation techniques using standard Statistical Quality Control (SQC) methods. Unless otherwise specified: I_O = 40mA, C_I = 0.33µF, C_O = 0.1µF. (Continued)

LM78L12AC (Continued)

Unless otherwise specified, V_{IN} = 19V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			13.7	14.5	V

LM78L15AC

Unless otherwise specified, V_{IN} = 23V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_O	Output Voltage		14.4	15.0	15.6	V
		17.5V ≤ V_{IN} ≤ 30V 1mA ≤ I_O ≤ 40mA (Note 3)	**14.25**		**15.75**	
		1mA ≤ I_O ≤ 70mA (Note 3)	**14.25**		**15.75**	
ΔV_O	Line Regulation	17.5V ≤ V_{IN} ≤ 30V		37	250	mV
		20V ≤ V_{IN} ≤ 30V		25	140	
ΔV_O	Load Regulation	1mA ≤ I_O ≤ 100mA		35	150	
		1mA ≤ I_O ≤ 40mA		12	75	
I_Q	Quiescent Current			3	5	mA
ΔI_Q	Quiescent Current Change	20V ≤ V_{IN} ≤ 30V			**1**	
		1mA ≤ I_O ≤ 40mA			**0.1**	
V_n	Output Noise Voltage			90		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz 18.5V ≤ V_{IN} ≤ 28.5V	37	51		dB
I_{PK}	Peak Output Current			140		mA
$\frac{\Delta V_O}{\Delta T}$	Average Output Voltage Tempco	I_O = 5mA		–1.3		mV/˚C
V_{IN} (Min)	Minimum Value of Input Voltage Required to Maintain Line Regulation			16.7	17.5	V

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. Electrical specifications do not apply when operating the device outside of its stated operating conditions.

Note 2: Human body model, 1.5 kΩ in series with 100pF.

Note 3: Power dissipation ≤ 0.75W.

Note 4: Recommended minimum load capacitance of 0.01µF to limit high frequency noise.

Note 5: Typical thermal resistance values for the packages are:

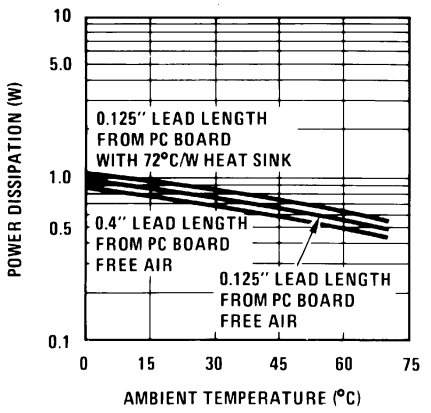
Z Package: θ_{JC} = 60 ˚C/W, = θ_{JA} = 230 ˚C/W

M Package: θ_{JA} = 180 ˚C/W

micro SMD Package: θ_{JA} = 230.9˚C/W

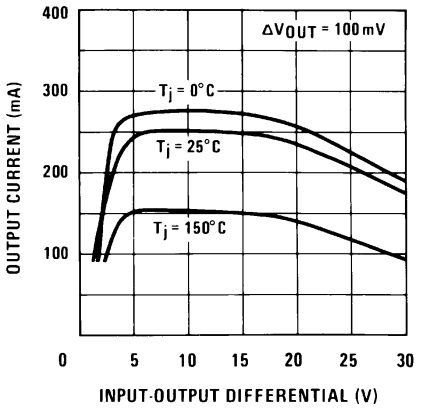
Typical Performance Characteristics

Maximum Average Power Dissipation (Z Package)



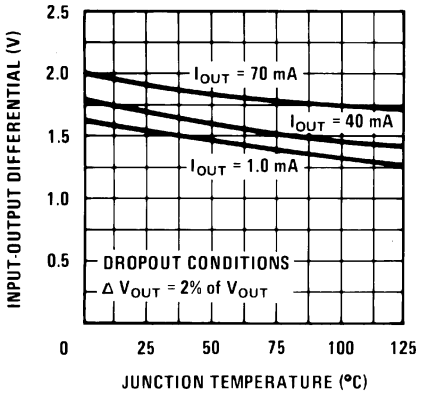
00774414

Peak Output Current



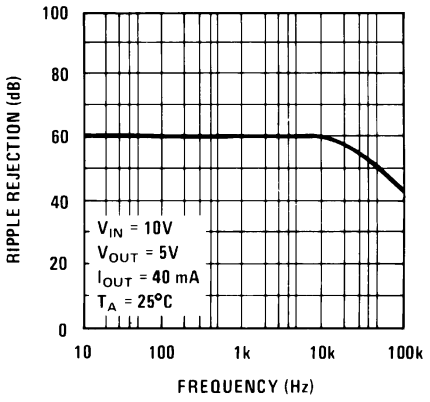
00774416

Dropout Voltage



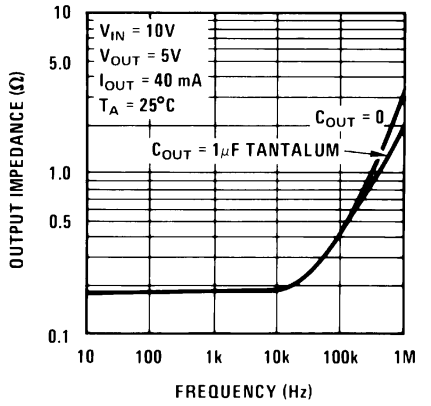
00774417

Ripple Rejection



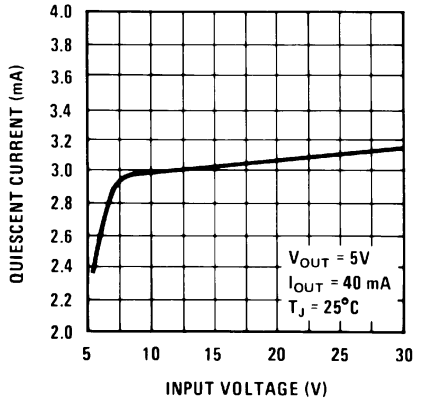
00774418

Output Impedance



00774419

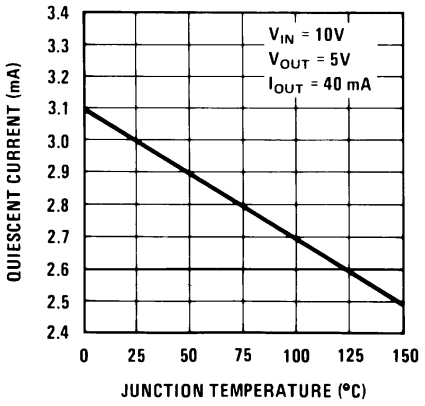
Quiescent Current



00774420

Typical Performance Characteristics (Continued)

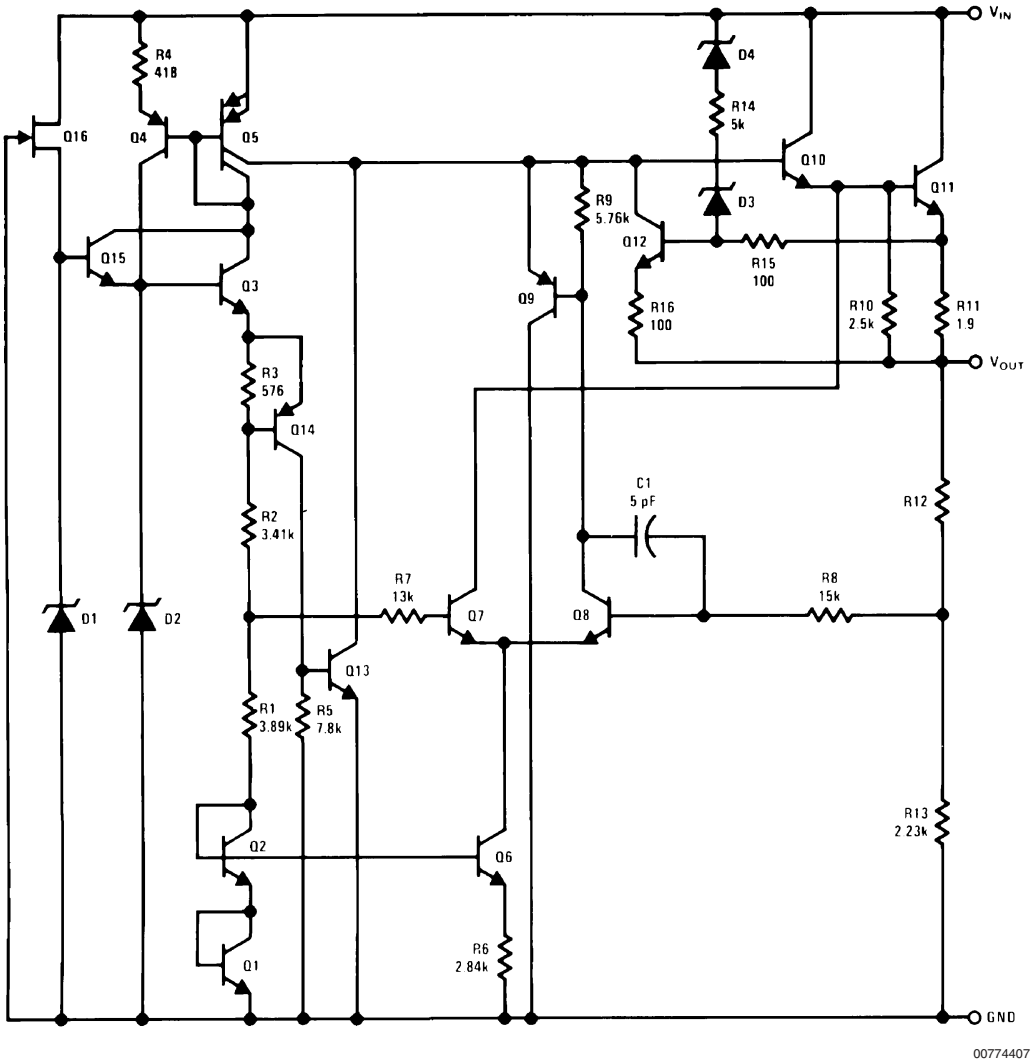
Quiescent Current



00774421

Equivalent Circuit

LM78LXX



00774407

TEXAS INSTRUMENTS

OPA354, OPA2354, OPA4354

SBOS233F – MARCH 2002 – REVISED JUNE 2016

OPAx354 250-MHz, Rail-to-Rail I/O, CMOS Operational Amplifiers

1 Features

- Unity-Gain Bandwidth: 250 MHz
- Wide Bandwidth: 100-MHz GBW
- High Slew Rate: 150 V/µs
- Low Noise: 6.5 nV$\sqrt{\text{Hz}}$
- Rail-to-Rail I/O
- High Output Current: > 100 mA
- Excellent Video Performance:
  - Diff Gain: 0.02%, Diff Phase: 0.09°
  - 0.1-dB Gain Flatness: 40 MHz
- Low Input Bias Current: 3 pA
- Quiescent Current: 4.9 mA
- Thermal Shutdown
- Supply Range: 2.5 V to 5.5 V
- *Micro*SIZE and PowerPAD™ Packages

2 Applications

- Video Processing
- Ultrasound
- Optical Networking, Tunable Lasers
- Photodiode Transimpedance Amps
- Active Filters
- High-Speed Integrators
- Analog-to-Digital (A/D) Converter Input Buffers
- Digital-to-Analog (D/A) Converter Output Amplifiers
- Barcode Scanners
- Communications

3 Description

The OPA354 series of high-speed, voltage-feedback CMOS operational amplifiers are designed for video and other applications requiring wide bandwidth. They are unity-gain stable and can drive large output currents. Differential gain is 0.02% and differential phase is 0.09°. Quiescent current is only 4.9 mA per channel.

The OPA354 series op amps are optimized for operation on single or dual supplies as low as 2.5 V (±1.25 V) and up to 5.5 V (±2.75 V). Common-mode input range extends beyond the supplies. The output swing is within 100 mV of the rails, supporting wide dynamic range.

For applications requiring the full 100-mA continuous output current, single and dual 8-pin HSOP PowerPAD versions are available.

The single version (OPA354) is available in the tiny 5-pin SOT-23 and 8-pin HSOP PowerPAD packages. The dual version (OPA2354) comes in the miniature 8-pin VSSOP and 8-pin HSOP PowerPAD packages. The quad version (OPA4354) is offered in 14-pin TSSOP and 14-pin SOIC packages.

Multichannel version feature completely independent circuitry for lowest crosstalk and freedom from interaction. All are specified over the extended –40°C to 125°C temperature range.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
OPA354	HSOP (8)	4.89 mm × 3.90 mm
	SOT-23 (5)	2.90 mm × 1.60 mm
OPA2354	VSSOP (8)	3.00 mm × 3.00 mm
	HSOP (8)	4.89 mm × 3.90 mm
OPA4354	SOIC (14)	8.65 mm × 3.91 mm
	TSSOP (14)	5.00 mm × 4.40 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Simplified Schematic

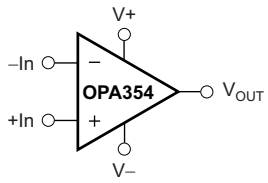


Table of Contents

4 Revision History

Changes from Revision E (March 2002) to Revision F **Page**

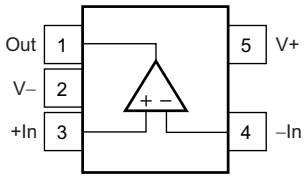
- Added *ESD Ratings* table, *Feature Description* section, *Device Functional Modes*, *Application and Implementation* section, *Power Supply Recommendations* section, *Layout* section, *Device and Documentation Support* section, and *Mechanical, Packaging, and Orderable Information* section ... 1
- Deleted *Package/Ordering Information* table, see POA at the end of the data sheet ... 1
- Renamed *OPAx354 Related Products* table to *Device Comparison Table* ... 3

5 Device Comparison Table

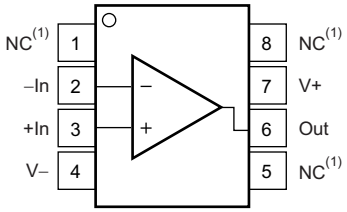
FEATURES	PRODUCT
Shutdown Version of OPA354 Family	OPAx357
200-MHz GBW, Rail-to-Rail Output, CMOS, Shutdown	OPAx355
200-MHz GBW, Rail-to-Rail Output, CMOS	OPAx356
38-MHz GBW, Rail-to-Rail Input/Output, CMOS	OPAx350/OPAx353
75-MHz BW G = 2, Rail-to-Rail Output	OPA2631
150-MHz BW G = 2, Rail-to-Rail Output	OPA2634
100-MHz BW, Differential Input/Output, 3.3-V Supply	THS412x

6 Pin Configuration and Functions

OPA354: DBV Package
5-Pin SOT-23
Top View



OPA354: DDA Package
8-Pin HSOP(2)
Top View

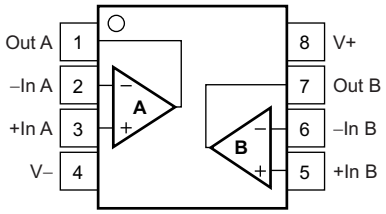


(1) NC means no internal connection.
PowerPAD must be connected to V− or left floating.

Pin Functions: OPA354

PIN			I/O	DESCRIPTION
NAME	SOT-23	HSOP		
–In	4	2	I	Inverting input
+In	3	3	I	Noninverting input
NC	—	1, 5, 8	—	No internal connection (can be left floating)
Out	1	6	O	Output
V–	2	4	—	Negative (lowest) supply
V+	5	7	—	Positive (highest) supply

OPA2354: DGK and DDA Packages
8-Pin VSSOP, HSOP(1)
Top View

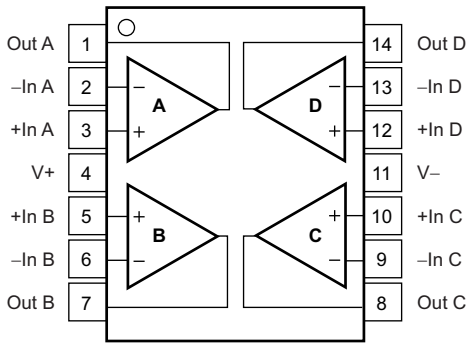


(1) PowerPAD must be connected to V− or left floating.

Pin Functions: OPA2354

PIN			I/O	DESCRIPTION
NAME	VSSOP	HSOP		
–In A	2	2	I	Inverting input, channel A
+In A	3	3	I	Noninverting input, channel A
–In B	6	6	I	Inverting input, channel B
+In B	5	5	I	Noninverting input, channel B
Out A	1	1	O	Output, channel A
Out B	7	7	O	Output, channel B
V–	4	4	—	Negative (lowest) supply
V+	8	8	—	Positive (highest) supply

OPA4354: D and PW Packages
14-Pin SOIC, TSSOP
Top View



Pin Functions: OPA4354

PIN			I/O	DESCRIPTION
NAME	SOIC	TSSOP		
–In A	2	2	I	Inverting input, channel A
+In A	3	3	I	Noninverting input, channel A
–In B	6	6	I	Inverting input, channel B
+In B	5	5	I	Noninverting input, channel B
–In C	9	9	I	Inverting input, channel C
+In C	10	10	I	Noninverting input, channel C
–In D	13	13	I	Inverting input, channel D
+In D	12	12	I	Noninverting input, channel D
Out A	1	1	O	Output, channel A
Out B	7	7	O	Output, channel B
Out C	8	8	O	Output, channel C
Out D	14	14	O	Output, channel D
V–	11	11	—	Negative (lowest) supply
V+	4	4	—	Positive (highest) supply

7 Specifications

7.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)

		MIN	MAX	UNIT
Voltage	Supply voltage, V+ to V−		7.5	V
	Signal input terminals(2)	(V−) − (0.5)	(V+) + 0.5	
Current	Signal input terminals(2)	–10	10	mA
	Output short circuit(3)	Continuous		
Temperature	Operating, T_A	–55	150	°C
	Junction, T_J		150	
	Storage, T_{stg}	–65	150	

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) Input pins are diode-clamped to the power-supply rails. Input signals that can swing more than 0.5 V beyond the supply rails must be current limited to 10 mA or less.
(3) Short-circuit to ground, one amplifier per package.

7.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001(1)	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101(2)	±250	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

7.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

		MIN	MAX	UNIT
V_S	Supply voltage, V– to V+	2.5	5.5	V
	Specified temperature	–40	125	°C

7.4 Thermal Information: OPA354

THERMAL METRIC(1)		OPA354		UNIT
		DBV (SOT-23)	DDA (HSOP)	
		5 PINS	8 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	216.3	42.5	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	84.3	54	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	43.1	26.5	°C/W
ψ_{JT}	Junction-to-top characterization parameter	3.8	8	°C/W
ψ_{JB}	Junction-to-board characterization parameter	42.3	26.4	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	—	3.6	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

7.5 Thermal Information: OPA2354

THERMAL METRIC(1)		OPA2354		UNIT
		DDA (HSOP)	DGK (VSSOP)	
		8 PINS	8 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	40.6	175.9	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	46	67.8	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	20.7	97.1	°C/W
ψ_{JT}	Junction-to-top characterization parameter	5.6	9.3	°C/W
ψ_{JB}	Junction-to-board characterization parameter	20.6	95.5	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	2.5	—	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

7.6 Thermal Information: OPA4354

THERMAL METRIC(1)		OPA4354		UNIT
		D (SOIC)	PW (TSSOP)	
		14 PINS	14 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	83.8	92.6	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	70.7	27.5	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	59.5	33.6	°C/W
ψ_{JT}	Junction-to-top characterization parameter	11.6	1.9	°C/W
ψ_{JB}	Junction-to-board characterization parameter	37.7	33.1	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	—	—	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

SBAS073B – JANUARY 1997 – REVISED NOVEMBER 2002

12-Bit, 20MHz Sampling ANALOG-TO-DIGITAL CONVERTER

FEATURES

- **HIGH SFDR: 74dB at 9.8MHz f_{IN}**
- **HIGH SNR: 68dB**
- **LOW POWER: 300mW**
- **LOW DLE: 0.25LSB**
- **FLEXIBLE INPUT RANGE**
- **OVER-RANGE INDICATOR**

APPLICATIONS

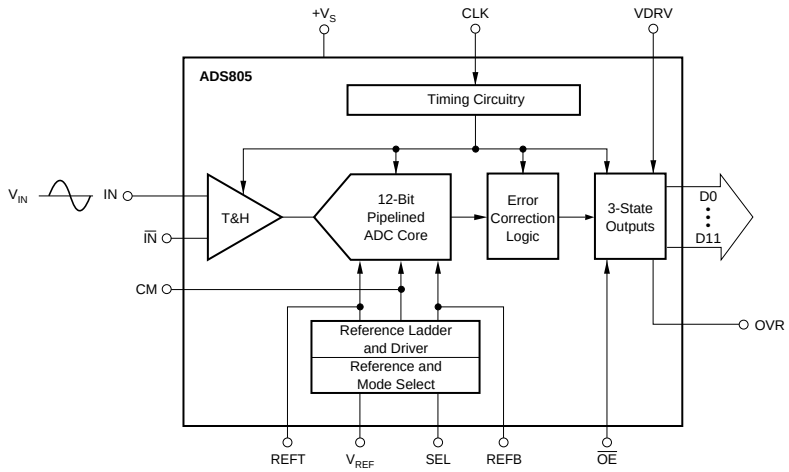
- **STUDIO CAMERAS**
- **IF AND BASEBAND DIGITIZATION**
- **COPIERS**
- **TEST INSTRUMENTATION**

DESCRIPTION

The ADS805 is a 20MHz, high dynamic range, 12-bit, pipelined Analog-to-Digital Converter ADC. This converter includes a high-bandwidth track-and-hold that gives excellent spurious performance up to and beyond the Nyquist rate. This high-bandwidth, linear track-and-hold minimizes harmonics and has low jitter, leading to excellent Signal-to-Noise Ratio (SNR) performance. The ADS805 is also pin-compatible with the 10MHz ADS804 and the 5MHz ADS803.

The ADS805 provides an internal reference or an external reference can be used. The ADS805 can be programmed for a 2Vp-p input range which is the easiest to drive with a single op amp and provides the best spurious performance. Alternatively, the 5Vp-p input range can be used for the lowest input-referred noise of 0.09LSBs rms giving superior imaging performance. There is also the capability to set the input range between 2Vp-p and 5Vp-p, either single-ended or differential. The ADS805 also provides an over-range flag that indicates when the input signal has exceeded the converter's full-scale range. This flag can also be used to reduce the gain of the front end signal conditioning circuitry.

The ADS805 employs digital error techniques to provide excellent differential linearity for demanding imaging applications. Its low distortion and high SNR give the extra margin needed for communications, medical imaging, video, and test instrumentation applications. The ADS805 is available in an SSOP-28 package.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

ABSOLUTE MAXIMUM RATINGS[1]

$+V_S$	+6V
Analog Input	–0.3V to $(+V_S)$ + 0.3V
Logic Input	–0.3V to $(+V_S)$ + 0.3V
Case Temperature	+100°C
Junction Temperature	+150°C
Storage Temperature	+150°C

NOTE: (1) Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Exposure to absolute maximum conditions for extended periods may affect device reliability.

ELECTROSTATIC DISCHARGE SENSITIVITY

This integrated circuit can be damaged by ESD. Texas Instruments recommends that all integrated circuits be handled with appropriate precautions. Failure to observe proper handling and installation procedures can cause damage.

ESD damage can range from subtle performance degradation to complete device failure. Precision integrated circuits may be more susceptible to damage because very small parametric changes could cause the device not to meet its published specifications.

PACKAGE/ORDERING INFORMATION

PRODUCT	PACKAGE-LEAD	PACKAGE DESIGNATOR[1]	SPECIFIED TEMPERATURE RANGE	PACKAGE MARKING	ORDERING NUMBER[2]	TRANSPORT MEDIA, QUANTITY
ADS805	SSOP-28	DB	–40°C to +85°C	ADS805E	ADS805E	Rails, 48
"	"	"	"	"	ADS805E/1K	Tape and Reel, 1000

NOTE: (1) For the most current specifications and package information, refer to our web site at www.ti.com.

ELECTRICAL CHARACTERISTICS

At T_A = full specified temperature range, V_S = +5V, specified input range = 1.5V to 3.5V, and single-ended input and sampling rate = 20MHz, unless otherwise specified.

		ADS805E			
PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
RESOLUTION			12 Bits Tested		
SPECIFIED TEMPERATURE RANGE			–40 to +85		°C
CONVERSION CHARACTERISTICS					
Sample Rate		10k		20M	Samples/s
Data Latency			6		Clk Cycles
ANALOG INPUT					
Standard Single-Ended Input Range		1.5		3.5	V
Optional Single-Ended Input Range		0		5	V
Standard Common-Mode Voltage			2.5		V
Standard Optional Common-Mode Voltage			1		V
Input Capacitance			20		pF
Analog Input Bandwidth	–3dBFS Input		270		MHz
DYNAMIC CHARACTERISTICS					
Differential Linearity Error (Largest Code Error)					
f = 500kHz			±0.25	±0.75	LSB
No Missing Codes			Tested		
Spurious-Free Dynamic Range[1]					
f = 9.8MHz		65	74		dBFS[2]
2-Tone Intermodulation Distortion[3]					
f = 7.7MHz and 7.9MHz (–7dB each tone)			–70		dBc
Signal-to-Noise Ratio (SNR)					
f = 9.8MHz		63	68		dBFS
Signal-to-(Noise + Distortion) (SINAD)					
f = 9.8MHz		62	66		dBFS
Effective Number of Bits at 9.8MHz[4]			10.7		Bits
Input Referred Noise	0V to 5V Input		0.09		LSBs rms
	1.5V to 3.5V Input		0.23		LSBs rms
Integral Nonlinearity Error					
f = 500kHz			±1	±2	LSB
Aperture Delay Time			3		ns
Aperture Jitter			4		ps rms
Over-Voltage Recovery Time	1.5x FS Input		2		ns
Full-Scale Step Acquisition Time			20		ns

NOTES: (1) Spurious-Free Dynamic Range refers to the magnitude of the largest harmonic. (2) dBFS means dB relative to full-scale. (3) 2-tone intermodulation distortion is referred to the largest fundamental tone. This number will be 6dB higher if it is referred to the magnitude of the 2-tone fundamental envelope. (4) Effective number of bits (ENOB) is defined by (SINAD – 1.76)/6.02. (5) Internal 50kΩ pull-down resistor. (6) Includes internal reference. (7) Excludes internal reference.

ELECTRICAL CHARACTERISTICS (Cont.)

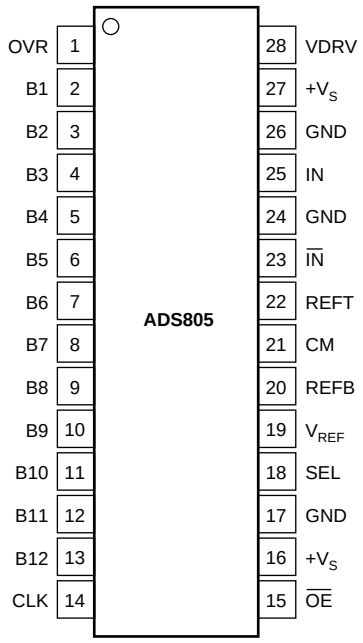
At T_A = full specified temperature range, V_S = +5V, specified input range = 1.5V to 3.5V, and single-ended input and sampling rate = 20MHz, unless otherwise specified.

PARAMETER	CONDITIONS	ADS805E MIN	ADS805E TYP	ADS805E MAX	UNITS
DIGITAL INPUTS					
Logic Family			CMOS Compatible		
Convert Command	Start Conversion		Rising Edge of Convert Clock		
High Level Input Current (V_{IN} = 5V)[5]				±100	µA
Low Level Input Current (V_{IN} = 0V)				10	µA
High Level Input Voltage		+3.5			V
Low Level Input Voltage				+1.0	V
Input Capacitance			5		pF
DIGITAL OUTPUTS					
Logic Family			CMOS/TTL Compatible		
Logic Coding			Straight Offset Binary		
Low Output Voltage	(I_{OL} = 50µA)			0.1	V
Low Output Voltage	(I_{OL} = 1.6mA)			0.4	V
High Output Voltage	(I_{OH} = 50µA)	+4.5			V
High Output Voltage	(I_{OH} = 0.5mA)	+2.4			V
3-State Enable Time	$\overline{OE}$ = L		20	40	ns
3-State Disable Time	$\overline{OE}$ = H		2	10	ns
Output Capacitance			5		pF
ACCURACY (5Vp-p Input Range)	f_S = 2.5MHz				
Zero-Error (Referred to –FS)	At 25°C		0.3	±1.5	%FS
Zero-Error Drift (Referred to –FS)			±5		ppm/°C
Gain Error[6]	At 25°C		0.7	±2.0	%FS
Gain Error Drift[6]			±18		ppm/°C
Gain Error[7]	At 25°C		0.2	±1.5	%FS
Gain Error Drift[7]			±10		ppm/°C
Power-Supply Rejection of Gain	ΔV_S = ±5%	60	70		dB
Reference Input Resistance			1.6		kΩ
Internal Voltage Reference Tolerance (V_{REF} = 2.5V)	At 25°C			±35	mV
Internal Voltage Reference Tolerance (V_{REF} = 1.0V)	At 25°C			±14	mV
POWER-SUPPLY REQUIREMENTS					
Supply Voltage: $+V_S$	Operating	+4.75	+5.0	+5.25	V
Supply Current: $+I_S$	Operating		60	69	mA
Power Dissipation	Operating		300	345	mW
Thermal Resistance, θ_{JA}					
SSOP-28		50		°C/W	

NOTES: (1) Spurious-Free Dynamic Range refers to the magnitude of the largest harmonic. (2) dBFS means dB relative to full-scale. (3) 2-tone intermodulation distortion is referred to the largest fundamental tone. This number will be 6dB higher if it is referred to the magnitude of the 2-tone fundamental envelope. (4) Effective number of bits (ENOB) is defined by (SINAD – 1.76)/6.02. (5) Internal 50kΩ pull-down resistor. (6) Includes internal reference. (7) Excludes internal reference.

PIN CONFIGURATION

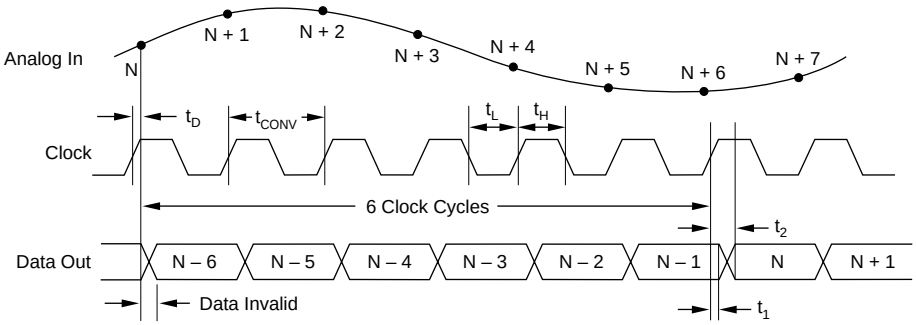
Top View **SSOP**



PIN DESCRIPTIONS

PIN	DESIGNATOR	DESCRIPTION
1	OVR	Over-Range Indicator
2	B1	Data Bit 1 (D11) (MSB)
3	B2	Data Bit 2 (D10)
4	B3	Data Bit 3 (D9)
5	B4	Data Bit 4 (D8)
6	B5	Data Bit 5 (D7)
7	B6	Data Bit 6 (D6)
8	B7	Data Bit 7 (D5)
9	B8	Data Bit 8 (D4)
10	B9	Data Bit 9 (D3)
11	B10	Data Bit 10 (D2)
12	B11	Data Bit 11 (D1)
13	B12	Data Bit 12 (D0) (LSB)
14	CLK	Convert Clock Input
15	$\overline{OE}$	Output Enable. H = High Impedance State. L = LOW or floating, normal operation (internal pull-down resistor).
16	$+V_S$	+5V Supply
17	GND	Ground
18	SEL	Input Range Select
19	V_{REF}	Reference Voltage Select
20	REFB	Bottom Reference
21	CM	Common-Mode Voltage
22	REFT	Top Reference
23	$\overline{IN}$	Complementary Analog Input
24	GND	Ground
25	IN	Analog Input (+)
26	GND	Ground
27	$+V_S$	+5V Supply
28	VDRV	Output Driver Voltage

TIMING DIAGRAM

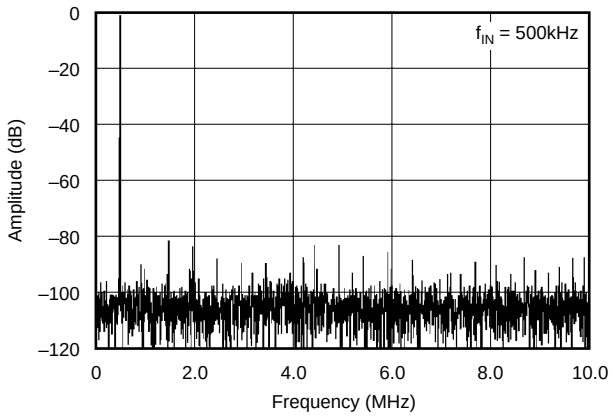


SYMBOL	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{CONV}	Convert Clock Period	50		100µs	ns
t_L	Clock Pulse LOW	24	25		ns
t_H	Clock Pulse HIGH	24	25		ns
t_D	Aperture Delay		3		ns
t_1	Data Hold Time, C_L = 0pF	3.9			ns
t_2	New Data Delay Time, C_L = 15pF max			12	ns

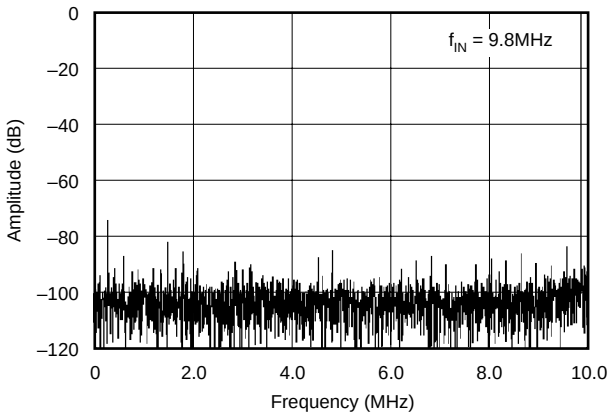
TYPICAL CHARACTERISITCS

At T_A = full specified temperature range, V_S = +5V, specified single-ended input range = 1.5V to 3.5V, and sampling rate = 20MHz, unless otherwise specified.

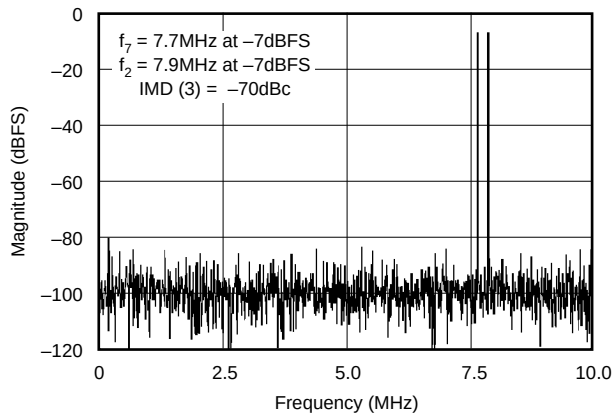
SPECTRAL PERFORMANCE



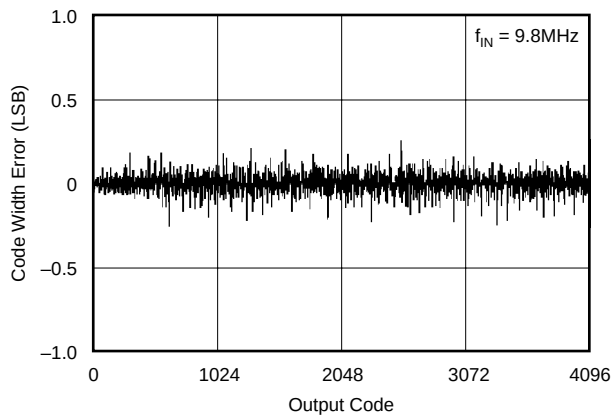
SPECTRAL PERFORMANCE



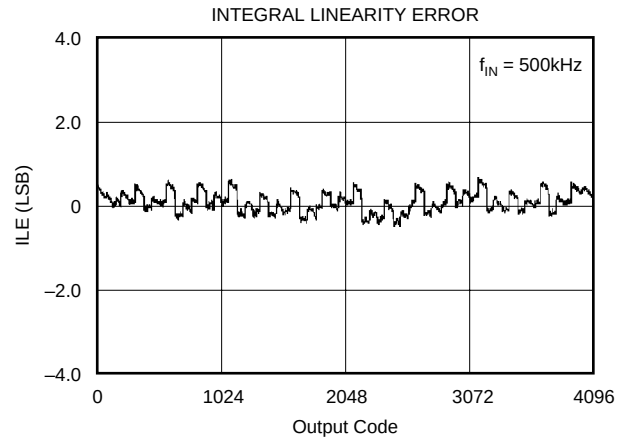
2-TONE INTERMODULATION DISTORTION



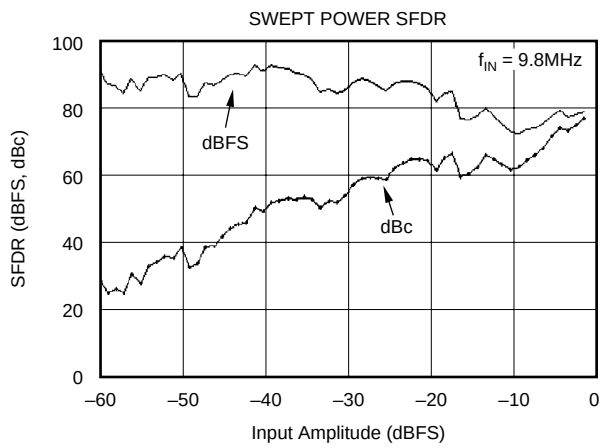
DIFFERENTIAL LINEARITY ERROR



INTEGRAL LINEARITY ERROR



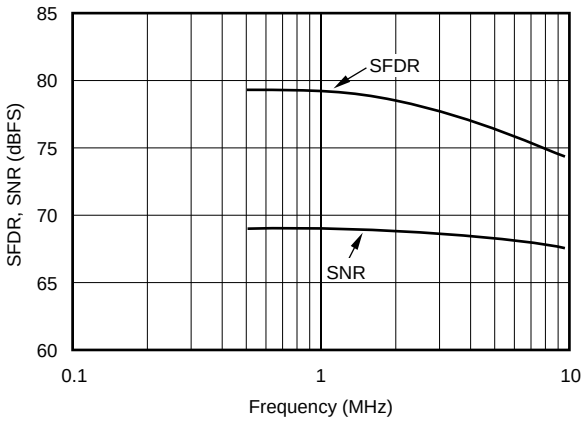
SWEPT POWER SFDR



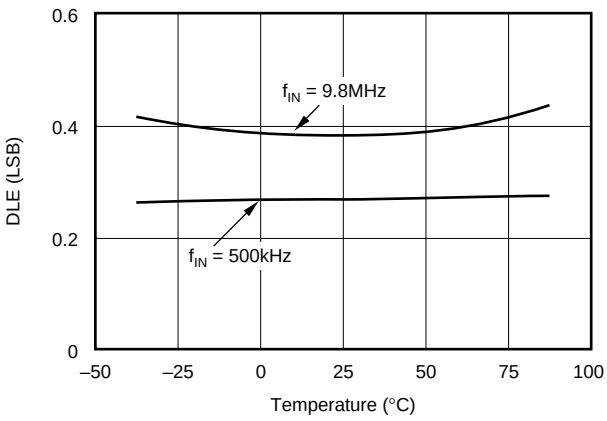
TYPICAL CHARACTERISITCS (Cont.)

At T_A = full specified temperature range, V_S = +5V, specified single-ended input range = 1.5V to 3.5V, and sampling rate = 20MHz, unless otherwise specified.

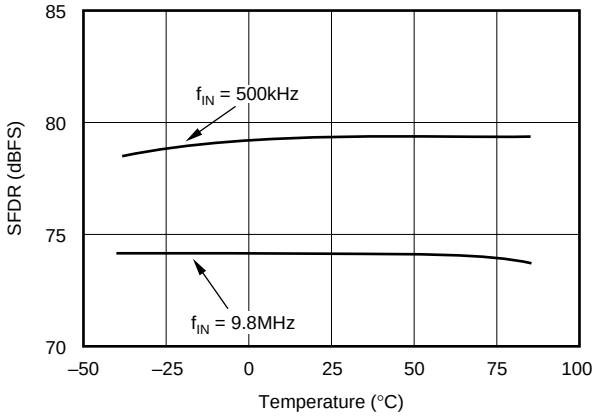
DYNAMIC PERFORMANCE vs INPUT FREQUENCY



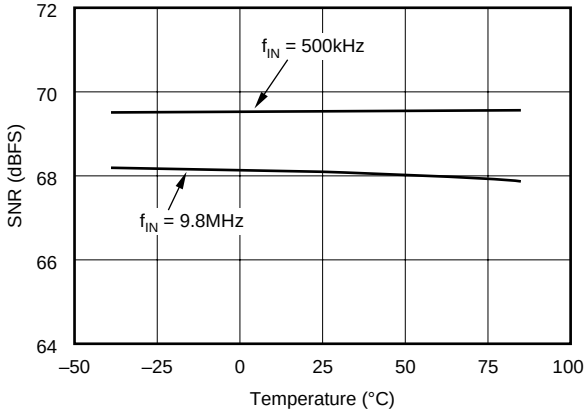
DIFFERENTIAL LINEARITY ERROR vs TEMPERATURE



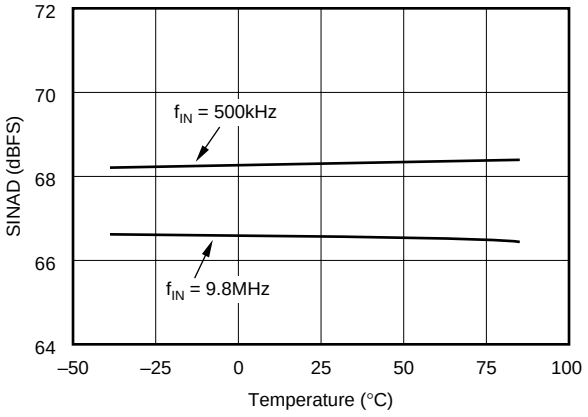
SPURIOUS-FREE DYNAMIC RANGE vs TEMPERATURE



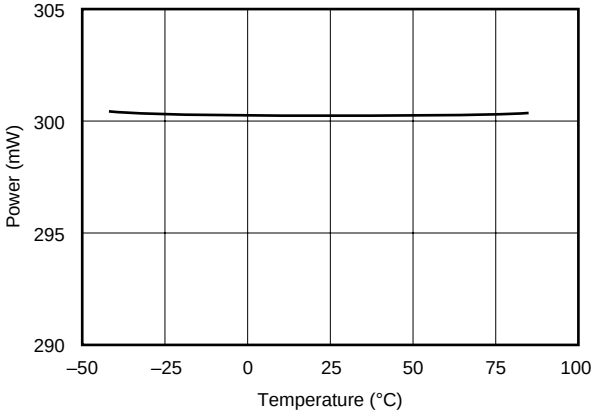
SIGNAL-TO-NOISE RATIO vs TEMPERATURE



SIGNAL-TO-(NOISE + DISTORTION) vs TEMPERATURE



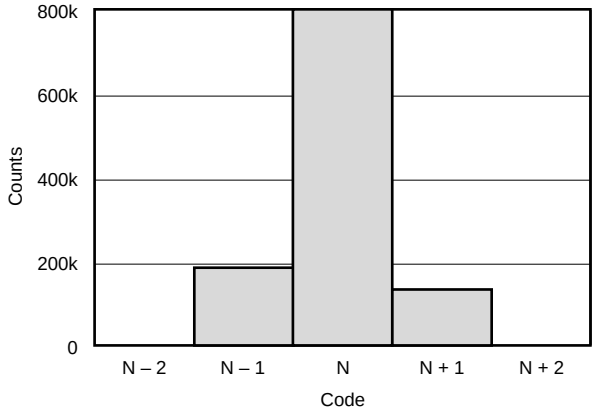
POWER DISSIPATION vs TEMPERATURE



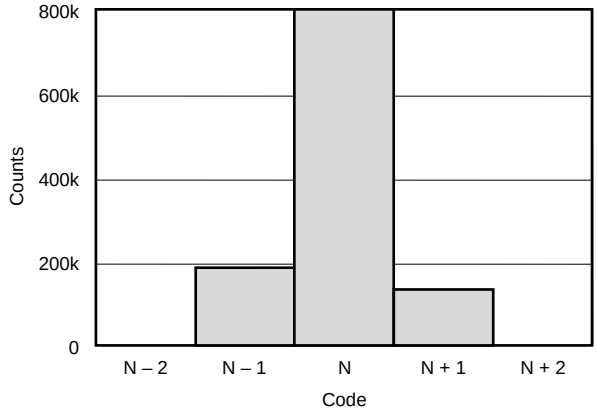
TYPICAL CHARACTERISITCS (Cont.)

At T_A = full specified temperature range, V_S = +5V, specified single-ended input range = 1.5V to 3.5V, and sampling rate = 20MHz, unless otherwise specified.

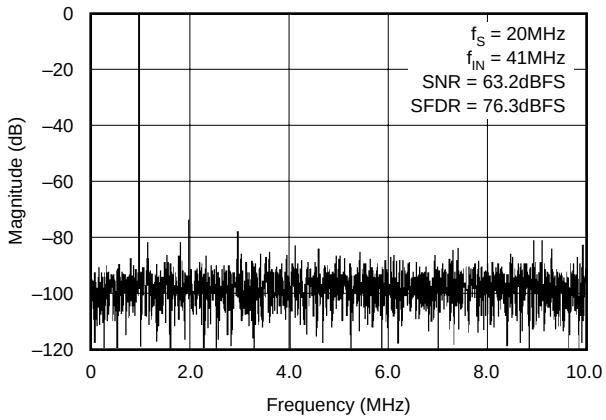
OUTPUT NOISE HISTOGRAM
(DC Input)

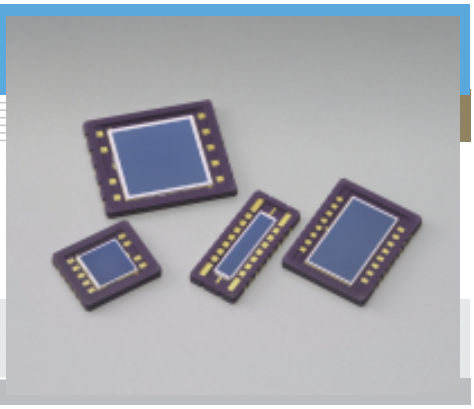


OUTPUT NOISE HISTOGRAM
(DC Input, V_{IN} = 5Vp-p Range)



UNDERSAMPLING
(Differential Input, 2Vp-p)





Si PIN photodiode
S5106, S5107, S7509, S7510

Chip carrier package for surface mount

S5106, S5107, S7509 and S7510 are Si PIN photodiodes sealed in chip carrier packages suitable for surface mount using automated solder reflow techniques. These photodiodes have large active areas, making them suitable for spatial light transmission where a wide field-of-view angle is required. Other applications include POS scanners, power meters and analytical instruments.

Features

- Active area
  S5106: 5 × 5 mm
  S5107: 10 × 10 mm
  S7509: 2 × 10 mm
  S7510: 6 × 11 mm
- Ceramic chip carrier package for surface mount
- Suitable for solder reflow
- High sensitivity

Applications

- Spatial light transmission
- Laser radar
- Power meter
- Bar-code reader

■ General ratings / Absolute maximum ratings

Type No.	Dimensional outline/ Window material *	Active area size (mm)	Effective active area (mm^2)	Absolute maximum ratings			
				Reverse voltage V_R Max (V)	Power dissipation P (mW)	Operating temperature Topr (°C)	Storage temperature Tstg (°C)
S5106	①/R	5 × 5	25	30	50	-40 to +100	-40 to +125
S5107	②/R	10 × 10	100				
S7509	③/R	2 × 10	20				
S7510	④/R	6 × 11	66				

■ Electrical and optical characteristics (Typ. Ta=25 °C, unless otherwise noted)

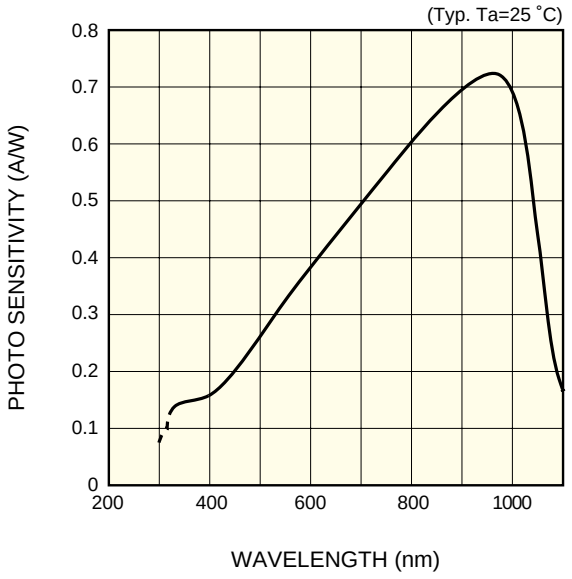
Type No.	Spectral response range λ (nm)	Peak sensitivity wavelength λp (nm)	Photo sensitivity S (A/W)				Short circuit current Isc 100 lx (μA)	Dark current I_D V_R=10 V		Temp. coefficient of I_D T_{CID} (times/°C)	Cut-off frequency fc R_L=50 Ω V_R=10 V (MHz)	Terminal capacitance Ct f=1 MHz V_R=10 V (pF)	NEP V_R=10 V λ=λp (W/$Hz^{1/2}$)
			λp	660 nm	780 nm	830 nm		Typ. (nA)	Max. (nA)				
S5106	320 to 1100	960	0.72	0.45	0.57	0.62	27	0.4	5	1.15	20	40	1.6×10^{-14}
S5107							110	0.9	10		10	150	2.4×10^{-14}
S7509							22	0.5	5		20	40	1.7×10^{-14}
S7510							72	1.0	10		15	80	2.5×10^{-14}

* Window R: Resin coating

Note) S5106, S7509: For mass production, order unit is 100 pieces.
S5107, S7510: For mass production, order unit is 50 pieces.

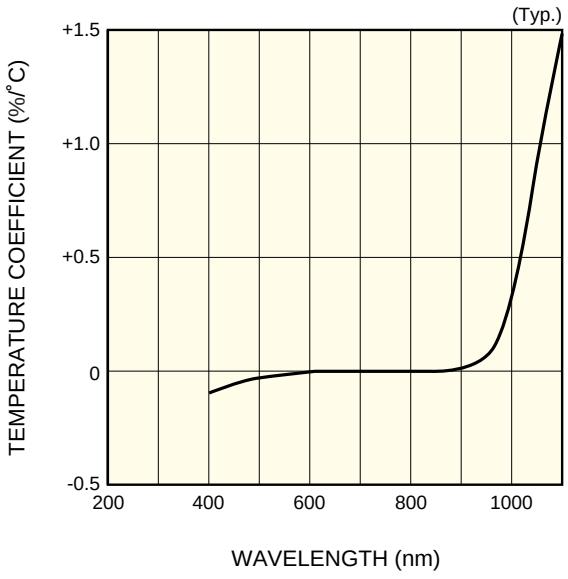
SOLID STATE DIVISION

■ Spectral response



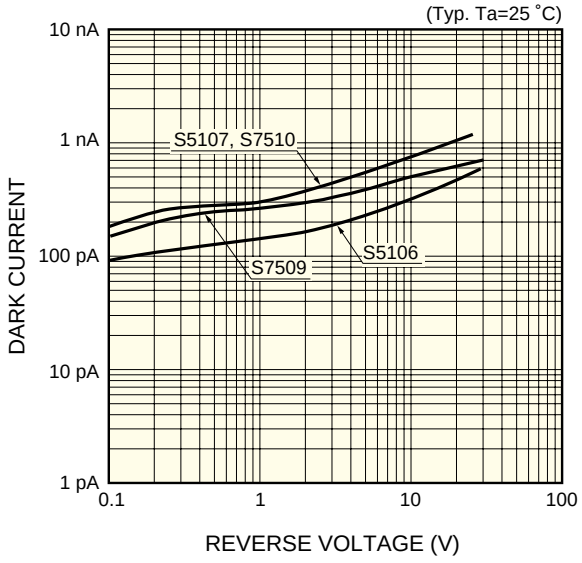
KPINB0165EA

■ Photo sensitivity temperature characteristic



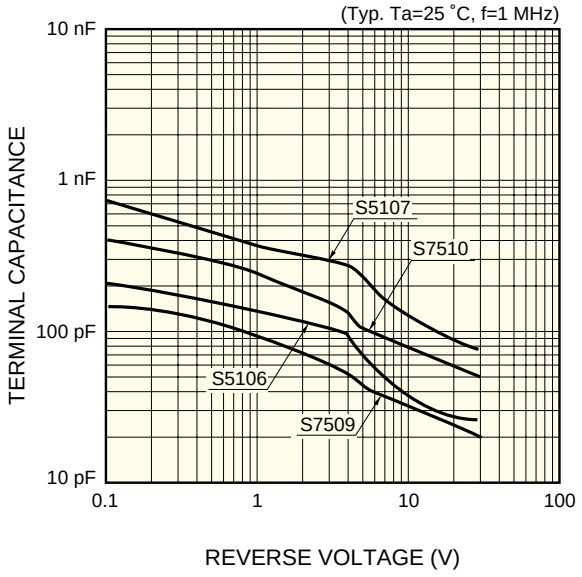
KPINB0093EC

■ Dark current vs. reverse voltage



KPINB0166EA

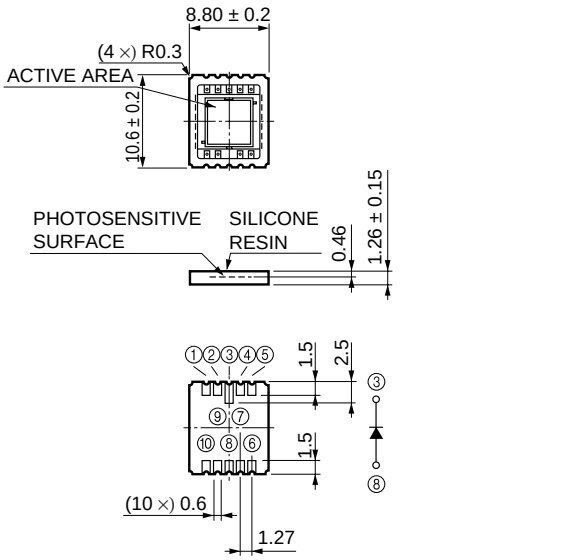
■ Terminal capacitance vs. reverse voltage



KPINB0128EA

■ Dimensional outlines (unit: mm)

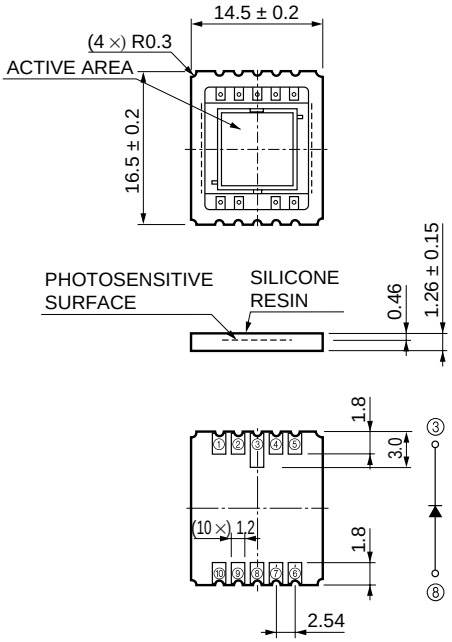
① S5106



NC (excluding pins ③⑧)
Burrs shall protrude no more than 0.3 mm on any side of package.

KPINA0002EE

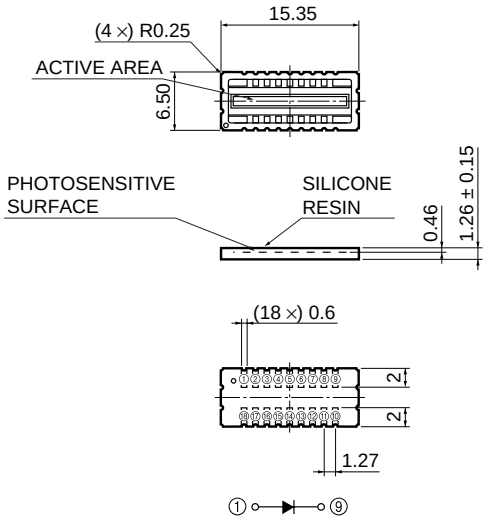
② S5107



NC (excluding pins ③⑧)
Burrs shall protrude no more than 0.3 mm on any side of package.

KPINA0013EC

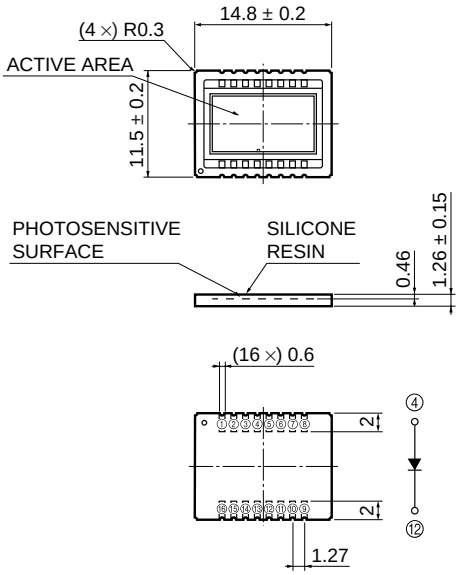
③ S7509



NC (excluding pins ①⑨)
Burrs shall protrude no more than 0.3 mm on any side of package.

KPINA0055EB

④ S7510



NC (excluding pins ④⑫)
Burrs shall protrude no more than 0.3 mm on any side of package.

KPINA0056EB

Precautions for use

- The light input window of this product uses soft silicone resin. Avoid touching the window to keep it from grime and damage that can decrease sensitivity. External force applied to the resin surface may deform or cut off the wires, so do not touch the window to prevent such troubles.
- Use rosin flux when soldering, to prevent the terminal lead corrosion. Reflow oven temperature should be at 260 °C maximum for 5 seconds maximum time under the conditions that no moisture absorption occurs.
  Reflow soldering conditions differ depending on the type of PC board and reflow oven. Carefully check these conditions before use.
- Silicone resin swells when it absorbs organic solvent, so do not use any solvent other than alcohol.
- Avoid unpacking until you actually use this product to prevent the terminals from oxidation and dust deposits or the coated resin from absorbing moisture.
  When the product is stored for 3 months while not unpacked or 24 hours have elapsed after unpacking, perform baking in nitrogen atmosphere at 150 °C for 3 to 5 hours or at 120 °C for 12 to 15 hours before use.

HAMAMATSU PHOTONICS K.K., Solid State Division
1126-1 Ichino-cho, Hamamatsu City, 435-8558 Japan, Telephone: (81) 53-434-3311, Fax: (81) 53-434-5184, www.hamamatsu.com
U.S.A.: Hamamatsu Corporation: 360 Foothill Road, P.O.Box 6910, Bridgewater, N.J. 08807-0910, U.S.A., Telephone: (1) 908-231-0960, Fax: (1) 908-231-1218
Germany: Hamamatsu Photonics Deutschland GmbH: Arzbergerstr. 10, D-82211 Herrsching am Ammersee, Germany, Telephone: (49) 08152-3750, Fax: (49) 08152-2658
France: Hamamatsu Photonics France S.A.R.L.: 19, Rue du Saule Trapu, Parc du Moulin de Massy, 91882 Massy Cedex, France, Telephone: 33-(1) 69 53 71 00, Fax: 33-(1) 69 53 71 10
United Kingdom: Hamamatsu Photonics UK Limited: 2 Howard Court, 10 Tewin Road, Welwyn Garden City, Hertfordshire AL7 1BW, United Kingdom, Telephone: (44) 1707-294888, Fax: (44) 1707-325777
North Europe: Hamamatsu Photonics Norden AB: Smidesvägen 12, SE-171 41 Solna, Sweden, Telephone: (46) 8-509-031-00, Fax: (46) 8-509-031-01
Italy: Hamamatsu Photonics Italia S.R.L.: Strada della Moia, 1/E, 20020 Arese, (Milano), Italy, Telephone: (39) 02-935-81-733, Fax: (39) 02-935-81-741

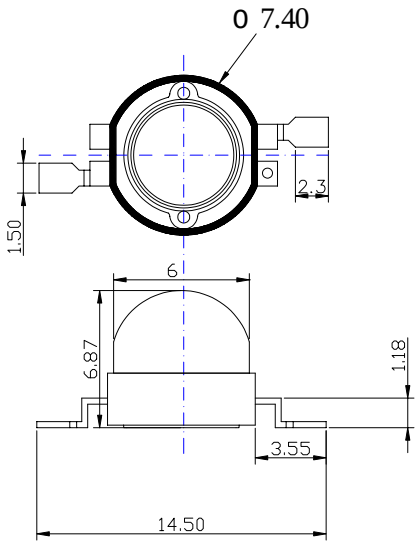
Cat. No. KPIN1033E04
Aug. 2006 DN

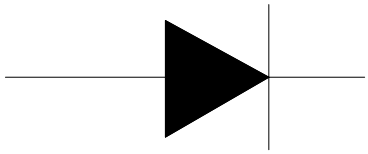
HP-803xx 3W Emitter Power LED Series

Features

- Highest Flux
- Very long operating life (life >> 50000 hr)
- More Energy Efficient than Incandescent and most Halogen lamps
- Low voltage DC operated
- Instant light (less than 100 ns)
- Superior ESD protection

Package Dimensions





Notes

1. Drawings not to scale
2. All dimensions are in millimeters .
3. Tolerance is ±0.1mm unless otherwise noted.
4. Protruded resin under flange is 1.0mm max.
5. Lead spacing is measured where the leads emerge from the package.
6. Specifications are subject to change without notice.
7. Precautions for ESD:

   STATIC SHIELD Electricity and surge damages the LED. It is recommended to use a wrist band or anti-electrostatic glove when handling the LED. All devices, equipment and machinery must be properly grounded

Absolute Maximum Ratings at Ta=25C

Parameter	Symbol	Max	Unit
Power Dissipation	PD	3	W
Pulse Forward Current	IPF	1000	mA
Forward Current	IF	700	mA
Reverse Voltage	VR	5	V
Operating Temperature Range	Topr	- 40 to +85	C
Storage Temperature Range	Tstg	- 40 to + 85	C

Flux Characteristics at 700mA , Junction Temperature, TJ= 25C

Color	Model	Typ.(lumens)	Angle(degree)
White	HP803NW	55 lm	120
Warm White	HP803WW	45 lm	120
Blue	HP803NB	16 lm	120
Cyan	HP803CN	40 lm	120
Pure Green	HP803PG	55 lm	120
Amber	HP803NO	36 lm	120
Red	HP803NR	40 lm	120

Optical Characteristics at 700mA , Junction Temperature, TJ = 25C

Color	CCT / λD			λ	λ/T
	Min.	Typ.	Max.		
White	5000K	6000K	7000K	--	--
Warm White	2850K	3300K	3800K	--	--
Blue	460nm	470nm	475nm	25	0.04
Cyan	490nm	505nm	515nm	30	0.04
Pure Green	520nm	530nm	540nm	35	0.04
Amber	587nm	590nm	595nm	20	0.05
Red	620nm	625nm	645nm	20	0.05

Electrical Characteristics at 700mA , Junction Temperature, TJ = 25C

Color	Forward Voltage			/W Thermal Resistance, Junction To Case	V/T
	Min.	Typ.	Max.		
White	3.8 v	4.0 v	4.2 v	15	-2.0
Warm White	3.8 v	4.0 v	4.2 v	15	-2.0
Blue	3.8 v	4.0 v	4.2 v	15	-2.0
Cyan	3.6 v	3.8 v	4.2 v	15	-2.0
Pure Green	3.6 v	3.8 v	4.2 v	15	-2.0
Amber	2.8 v	3.0 v	3.2 v	18	-2.0
Red	2.8 v	3.0 v	3.2 v	18	-2.0

Notes

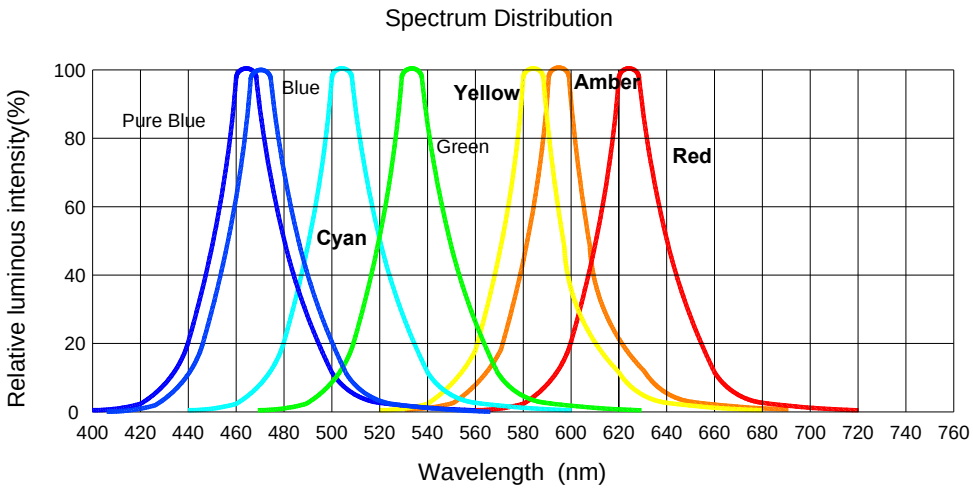
1. Minimum luminous flux or radiometric power performance guaranteed within published operating conditions. maintains a tolerance of ±10% on flux and power measurements
2. Dominant wavelength is derived from the CIE 1931 Chromaticity diagram and represents the perceived color. maintains a tolerance of± 1 nm for dominant wavelength measurements.
3. CCT ±5% tester tolerance.

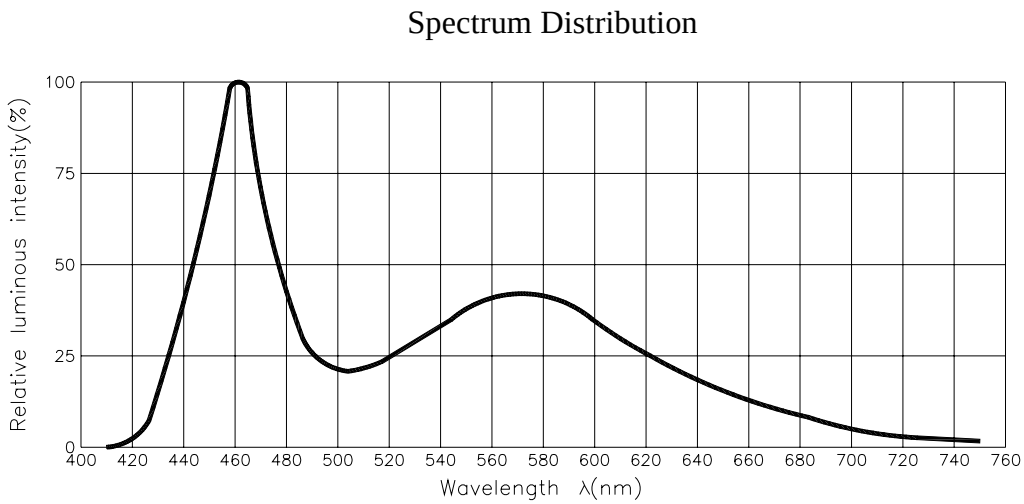
Precautions For Use

- Over–current–proof

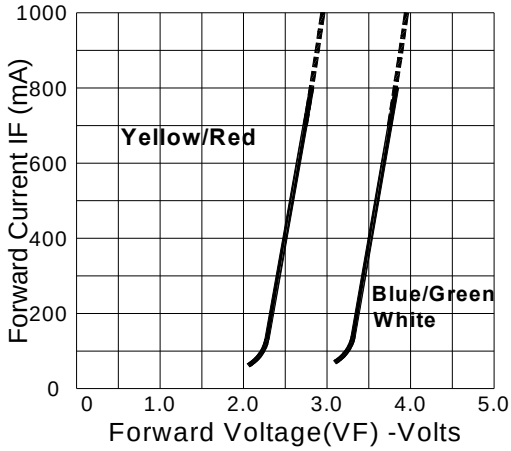
  Customer must apply resistors for protection, otherwise slight voltage shift will cause big current change □Burn out will happen□

Typical Electrical / Optical Characteristics Curves

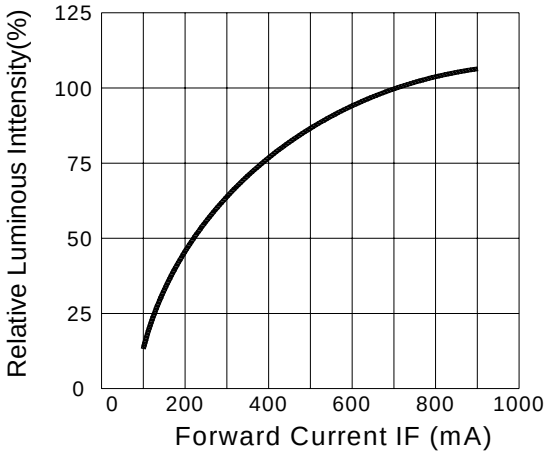




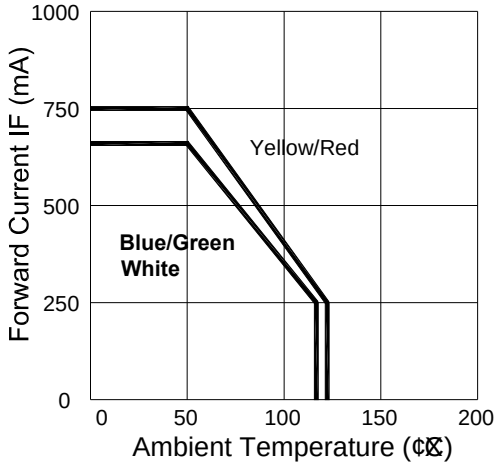
Forward Current VS. Forward Voltage



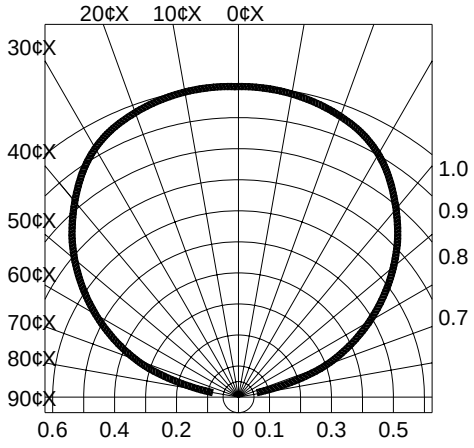
Luminous Intensity VS. Forward Current



Forward Current VS. Ambient Temperature



Radiation Diagram



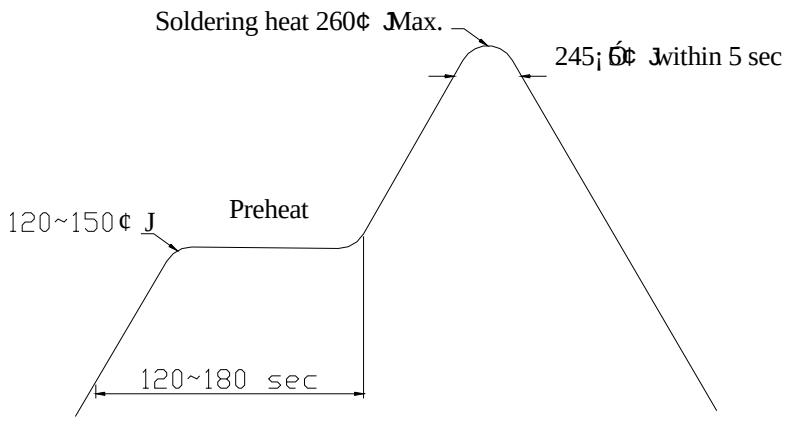
Reliability Test Items and Conditions

No.	Item	Test Conditions				Test time	Ac/Re
1	Solder Heat	260±5C				5 sec	0/1
2	Temperature Cycle	-40C	25C	105C	25C	100cycle	0/1
		30 min	5 min	30 min	5 min		
3	Thermal Shock	-40C		105C		20 cycle	0/1
		5 min		5 min			
4	High Temperature Storage	85C				1000 hrs	0/1
5	Low Temperature Storage	-35C				1000 hrs	0/1
6	DC Operating Life	I_F 700mA				1000 hrs	0/1
7	High Temperature/High Humidity	Ta 60C R.H 90 .				1000 hrs	0/1

Judgment Criteria		
	Forward Voltage Vf	Vf_{max} Increase <1.2x
	Reverse Current I_R	$I_{R\ max}$ Increase <2x
	Luminous Intensity Flux	Iv Decay < 50%

Note Measurement shall be taken after the tested samples have been returned to normal ambient conditions.

Soldering heat reliability DIP Please refer to the following figure



ROITHNER LASERTECHNIK, A-1040 Vienna, Austria, Wiedner Hauptstrasse 76, Top 9/1

Tel.: +43-1-586 52 43 - 0, Fax.: +43-1-586 52 43 44

e-mail: office@roithner-laser.com

http://www.roithner-laser.com