

# SISTEMAS DIGITALES PROGRAMABLES

---

*Guía de realización de las prácticas utilizando la placa DE2 de Altera.*

Ricardo J. Colom Palero, Rafael Gadea Girones y Vicente Herrero Bosch

Universitat Politècnica de València

---



### *Histórico de Revisiones*

Versión	Fecha	Cambios
1.0	30/11/2011	Versión Preliminar
1.1	16/02/2012	Se ha añadido el apartado de cómo archivar un proyecto en Quartus.
1.2	16/02/2012	Se ha añadido el desarrollo de Testbench para simulación empleando Modelsim
1.3	21/02/2012	Se ha añadido como se hace la configuración inicial de Modelsim.
1.4	02/03/2012	Se ha modificado la Figura 5.11. Esquema de entradas y salidas del bloque de generación de datos y sincronismo de la pantalla LCD.
1.5	18/04/2012	Modificación de algunos detalles en la redacción de las Tareas 3.1 y 3.2.

# Contenido

---

1	Descripción del Hardware Utilizado en el Laboratorio .....	5
1.1	Introducción .....	5
1.2	Descripción de la placa DE2 .....	6
1.2.1	Las entradas de reloj .....	8
1.2.2	Uso de los LEDs, interruptores y pulsadores .....	8
1.2.3	Los visualizadores 7-segmentos .....	9
1.3	Verificación física de un diseño .....	10
1.4	¿Cómo archivar un proyecto en Quartus II? .....	13
2	Desarrollo de Testbench para Simulación Empleando Modelsim .....	15
2.1	El Testbench .....	15
2.2	El Simulador Modelsim-Altera .....	16
3	Diseño de Sistemas Secuenciales .....	23
3.1	Objetivos .....	23
3.2	Repaso conceptos teóricos .....	23
3.2.1	Procesos secuenciales .....	23
3.2.2	Diseño jerárquico en Verilog HDL .....	25
3.2.3	La reutilización de los diseños .....	26
3.3	Estructura de la práctica .....	28
3.4	Desarrollo de la práctica .....	28
3.4.1	Tarea 1.1 – Realización de un Contador con Verilog HDL .....	28
3.4.2	Tarea 1.2 – Contador parametrizable .....	28
3.4.3	Tarea 1.3 – Registro de desplazamiento .....	29
3.4.4	Tarea 1.4 – Diseño del juego de luces del coche fantástico .....	29
4	Diseño de Maquinas de Estado Finitos .....	31
4.1	Objetivos .....	31
4.2	Introducción teórica .....	31
4.2.1	Máquinas de estados finitos en Quartus II .....	31
4.3	Estructura de la práctica .....	33
4.4	Desarrollo de la práctica .....	34
4.4.1	Tarea 2.1 – Diseño de un controlador para un motor paso a paso .....	34
4.4.2	Tarea 2.2 – Verificación del controlador para motor paso a paso .....	36
4.4.3	Tarea 2.3 – Diseño de una máquina de estados para el juego de luces del coche fantástico .....	38

5	Diseño de un Controlador para Pantalla Táctil.....	39
5.1	Objetivos .....	39
5.2	Introducción teórica .....	39
5.3	Descripción del hardware específico.....	41
5.4	Utilización del Módulo LTM.....	45
5.4.1	El interface serie de la pantalla LCD .....	45
5.4.2	Temporización de las señales para la visualización en la pantalla LCD. ....	47
5.4.3	El interface serie del convertidor ADC .....	50
5.5	Estructura de la práctica .....	51
5.6	Desarrollo de la práctica .....	51
5.6.1	Tarea 3.1 – Generación de las señales de sincronismo y datos de la pantalla LCD.....	52
5.6.2	Tarea 3.2 – Cambio de color de la pantalla .....	55
5.6.3	Tarea 3.3 – Generación de barras de colores en la pantalla.....	55
5.6.4	Tarea 3.4 – Visualización de caracteres en pantalla.....	55
5.6.5	Tarea 3.5 – Generación de una figura simple en un punto determinado de la pantalla .....	58
5.6.6	Tarea 3.6 – Generación de una frase que aparezca en pantalla.....	58
5.6.7	Tarea 3.7 – Escribir desde el teclado y visualizar en pantalla.....	58
5.6.8	Tarea 4.1 – Interface serie del convertidor ADC .....	60
5.6.9	Tarea 4.2 – Visualización del punto de contacto de la pantalla.....	62
5.6.10	Tarea 5 – Diseño libre .....	62
5.6.11	Tarea 6 – Interface serie de la pantalla LCD.....	62

## 1 Descripción del Hardware Utilizado en el Laboratorio

Los métodos de aprendizaje deben ser estructurados, gráficos e ilustrativos. Para aprender el diseño de dispositivos programables, hay que utilizar una herramienta. En el mercado existe una gran diversidad de ellas. Altera tiene un programa universitario con un software muy gráfico e intuitivo complementado con un hardware de verificación.

### 1.1 Introducción

A finales de los años 90, el fabricante de dispositivos programables Altera creó el Programa Universitario; el cual fue diseñado para cubrir las necesidades de los profesores de universidad, que impartían clases sobre diseño digital basado en dispositivos lógicos programables. El Programa Universitario disponía de todo lo necesario para crear e implementar diseños digitales, incluyendo las siguientes características:

- El software de desarrollo MAX+plus II, en su versión 7.21 estudiantil.
- La Placa educativa UP1. Con los dispositivos:
  - El dispositivo EPM7128S de la familia MAX7000S.
  - El dispositivo EPF10K20 de la familia FLEX10K.
- Cable de programación por puerto paralelo denominado ByteBlaster.

La electrónica es una de las ciencias más cambiantes que existen, lo cual implica que actualmente el programa universitario de Altera haya evolucionado hacia otras versiones más actuales de software y de hardware.

El software de desarrollo MAX+plus II, incluía un completo flujo de diseño y un intuitivo entorno gráfico. La entrada del diseño se podía realizar mediante esquemáticos o mediante fichero de textos con descripciones hardware con VHDL. También disponía de programación, compilación y verificación de diseños para los dispositivos EPM7128S y EPF10K20. Para maximizar el aprendizaje, el software de MAX+plus II incluía una completa e instantánea ayuda. Manteniendo esta filosofía MAX+plus II evolucionó en el tiempo a través de diferentes versiones hasta alcanzar la versión 10.2.

Tras la versión 10.2 de MAX+plus II, Altera cambió el Programa Universitario hacia una herramienta más profesional, pero manteniendo la compatibilidad con el entorno gráfico de la anterior. El nuevo software de desarrollo para el Programa Universitario se ha denominado Quartus II, que actualmente se encuentra en su versión 11.1.

La placa UP1 fue un experimento basado en dos dispositivos de Altera de las familias MAX7000S y FLEX10K. Tiene un diseño simple, que usado con el software de MAX+plus II o Quartus II, proporciona una plataforma para el aprendizaje del diseño digital utilizando herramientas de desarrollo de alto nivel y PLDs.

La placa educativa UP1 fue diseñada para cubrir las necesidades de los educadores y del laboratorio de diseño. La placa UP1 soporta tanto arquitecturas basadas en productos terminados como arquitecturas basadas en tablas de look-up (LUTs) e incluye dos PLDs. El dispositivo EPM7128S puede ser programado en la propia placa por

medio de un cable usando la técnica del tipo ByteBlaster. El dispositivo EPF10K20 puede ser configurado utilizando también el cable ByteBlaster o por medio de una memoria EPROM.

Con el tiempo, la placa educativa UP1 dio paso a la placa UP2, cuya principal diferencia está en que se substituyó el dispositivo EPF10K20 por un EPF10K70, el cual tiene el mismo encapsulado y patillaje. Por tanto, a partir de aquí se hablará de la placa educativa UP, cuando se refiera a aspectos comunes a las dos.

A principios de 2004, Altera sacó al mercado una nueva placa para educación denominada UP3, que supuso un cambio bastante radical en las características de la misma. La UP3 únicamente incorpora un dispositivo el EP1C6 de la familia Cyclone, lo cual impone la necesidad de utilizar Quartus II como herramienta de diseño, ya que MAX+plus II no dispone de este dispositivo. La placa UP3 tiene un gran número de elementos adicionales como son un LCD, memorias, interface USB, conexión PS/2, VGA, etc., que la hacen más versátil. Todo ello hace posible que la placa UP3 también se pueda utilizar para realizar diseños con Nios. Nios es lo que se denomina en inglés un “softcore processor”; es decir un cuerpo de un microprocesador, que ha sido diseñado por Altera y que se puede utilizar como un elemento más del diseño.

La política actual de Altera consiste en reducir al mínimo el número de placas educativas que dispone, de tal manera que se utilice la misma placa para realizar diseños digitales con dispositivos programables y diseños basados en Nios. Por todo ello, a principios de 2006 salió al mercado la nueva placa denominada DE2 (Development and Education), por tratarse de un diseño para educación y para desarrollos.

Los diseños pueden ser fácilmente cargados en la placa educativa, por medio del cable de programación, el cual es una simple interface de comunicación entre el PC y la placa. Este cable es un canal de comunicación para transferir datos entre el software (MAX+plus II o Quartus II) y la placa educativa. Debido a que los diseños son cargados directamente en el dispositivo de la placa, resulta relativamente sencillo probar un prototipo y cualquier cambio sobre el diseño puede ser testado de inmediato.

## 1.2 Descripción de la placa DE2

El elemento principal de la placa DE2 es la FPGA, es este caso una Cyclone II. Alrededor de dicho dispositivo se emplazan un gran número de elementos que hacen de la placa DE2 una herramienta muy versátil para docencia, permitiendo implementar un gran número de diseños, desde sencillos circuitos digitales hasta complejos proyectos multimedia. En la Figura 1.1 se puede ver una imagen de la placa DE2 en la que se muestran todos los elementos que la constituyen y su ubicación, los cuales se enumeran a continuación:

- Dispositivo FPGA Altera Cyclone® II EP2C35.
- Dispositivo Configuración serie EPCS16.
- USB Blaster (en placa) para programación y uso de las APIs de control. Acepta la programación por JTAG y AS (Active Serial).
- 512 kbyte SRAM.

- 8 Mbyte SDRAM.
- 4 Mbyte memoria flash.
- Conector para tarjetas SD.
- 4 pulsadores.
- 18 interruptores.
- 18 LEDs rojos.
- 9 LEDs verdes.
- 8 visualizadores 7-segmentos.
- Módulo de visualización LCD.
- Un oscilador de 50 MHz y otro de 27 MHz, para actuar como reloj.
- CODEC de audio de 24-bit calidad CD con conectores tipo jacks para línea de entrada, salida y micrófono.
- Convertidor digital-analógico para VGA (Triple DAC de 10-bit alta velocidad) con conector de salida VGA.
- Decodificador de TV (NTSC/PAL) y conector de entrada de TV.
- Controlador Ethernet 10/100 con conector RJ45.
- Controlador USB host-device con conectores USB del tipo A y del tipo B.
- Transceiver RS-232 y conector de 9 pines.
- Conector PS/2 para ratón o teclado.
- Transceiver IrDA.
- Dos conectores de expansión de 40 pines con diodos de protección.

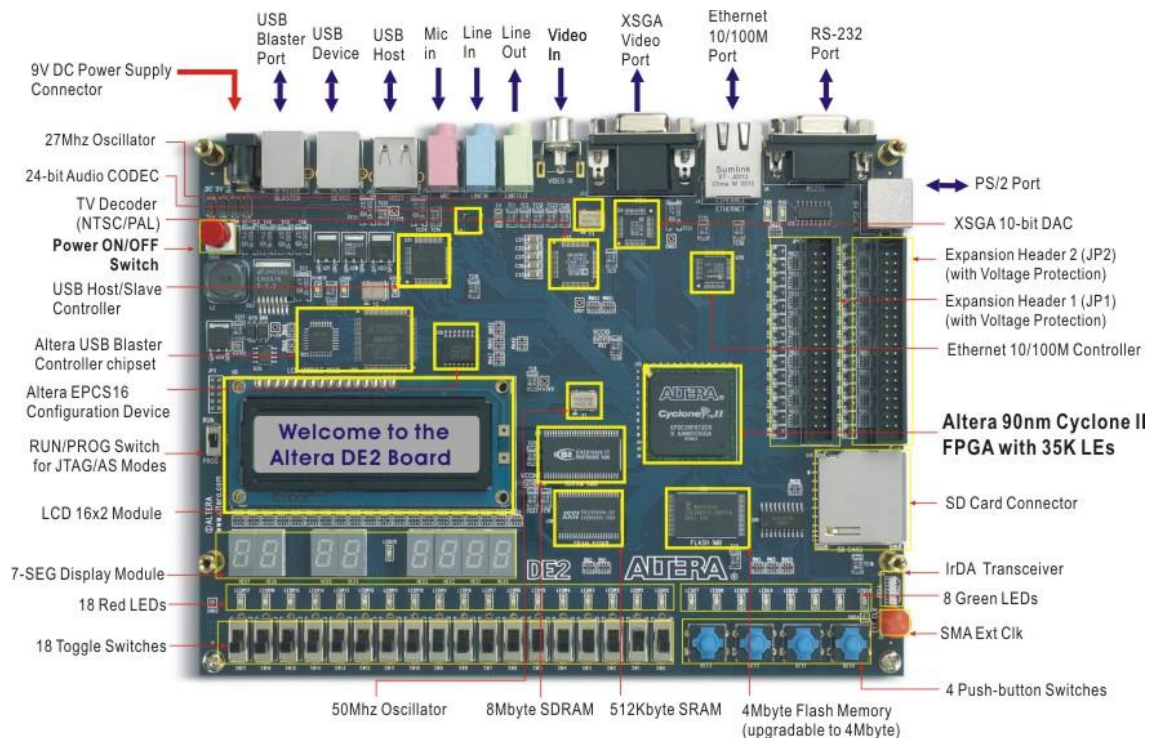


Figura 1.1. La placa de Altera DE2.

Todos estos elementos que constituyen la placa DE2, se conectan física e inamoviblemente, por medio de las pistas de la placa de circuito impreso, a los pines



del dispositivo Cyclone II. Cada componente se conecta a uno o varios de las 672 patillas que tiene la FPGA. Para poder utilizar desde el dispositivo Cyclone II uno o varios de los elementos enumerados, habrá que saber a cuál o a cuales de los pines de la FPGA se conecta dicho componente. Esta información la facilita el fabricante de la placa, que es el que ha establecido las conexiones y que a continuación se detalla, por medio de unas tablas, para aquellos elementos que se ha considerado que se van a utilizar en el desarrollo de la asignatura.

La placa DE2 dispone de una configuración precargada de fábrica, que permite rápidamente ver sus características. En el momento que se conecta la alimentación de la placa, se carga la configuración de fábrica y se puede ver:

- Parpadear todos los LEDs.
- Todos los displays 7-segmentos, cuentan de 0 a F cíclicamente.
- En el LCD se muestra el mensaje “Welcome to the Altera DE2 Board”.

### 1.2.1 Las entradas de reloj

La placa DE2 de Altera incorpora dos osciladores que generan señales de 27 MHz y 50 MHz. La placa también tiene un conector del tipo SMA, que permite la conexión de una fuente de reloj externa. Es importante remarcar que para utilizar el reloj de 27 MHz hay que poner a nivel lógico alto el pin denominado “TD\_RESET” que corresponde con el número C4 del dispositivo Cyclone II. En la Tabla 1.1, se muestra la lista de los pines del dispositivo Cyclone II que se conectan con las diferentes señales de reloj comentadas.

**Tabla 1.1. Asignación de pines de la señales de reloj.**

Nombre	Número Pin FPGA	Comentarios
CLOCK_27	D13	Señal de reloj de 27 MHz
CLOCK_50	N2	Señal de reloj de 50 MHz
EXT_CLOCK	P26	Señal de reloj de externa.

### 1.2.2 Uso de los LEDs, interruptores y pulsadores

La placa DE2 tiene cuatro pulsadores, cada uno de los cuales tiene un circuito “Trigger Schmitt”, para eliminar los rebotes al pulsar. Los cuatro pulsadores denominados de KEY0 a KEY3, se conectan directamente al dispositivo Cyclone II. Cada pulsador proporciona un nivel lógico alto (‘1’) en reposo (cuando no está pulsado) y un nivel lógico bajo (‘0’) mientras está pulsado.

Hay 18 interruptores en la placa DE2, que no dispone de circuitería eliminadora de rebotes. Cada interruptor se conecta directamente a un pin del dispositivo Cyclone II. Se encuentran en la parte inferior de la placa y se han denominado de SW0 a SW17. Cuando el interruptor se encuentra hacia abajo (hacia el borde de la placa) genera un nivel lógico bajo (‘0’), mientras que cuando se encuentra hacia arriba genera un nivel lógico alto (‘1’).

La placa DE2 tiene 27 diodos LED, de los cuales hay 18 LEDs rojos situados cerca de los 18 interruptores, 8 diodos LED verdes junto a los pulsadores y un diodo LED verde adicional situado entre los displays 7-segmentos. Cada diodo LED se conecta



directamente a un pin del dispositivo Cyclone II, cuando se pone un '1' lógico en el pin el LED se enciende, mientras que un '0' lógico mantiene el LED apagado.

En la Tabla 1.2, Tabla 1.3 y Tabla 1.4 se muestra la lista de los pines del dispositivo Cyclone II que se conectan con los interruptores, pulsadores y diodos LED respectivamente.

**Tabla 1.2. Asignación de pines de los interruptores.**

Nombre	Número Pin FPGA	Nombre	Número Pin FPGA	Nombre	Número Pin FPGA
SW[0]	N25	SW[6]	AC13	SW[12]	P2
SW[1]	N26	SW[7]	C13	SW[13]	T7
SW[2]	P25	SW[8]	B13	SW[14]	U3
SW[3]	AE14	SW[9]	A13	SW[15]	U4
SW[4]	AF14	SW[10]	N1	SW[16]	V1
SW[5]	AD13	SW[11]	P1	SW[17]	V2

**Tabla 1.3. Asignación de pines de los pulsadores.**

Nombre	Número Pin FPGA	Nombre	Número Pin FPGA
KEY[0]	G26	KEY[2]	P23
KEY[1]	N23	KEY[3]	W26

**Tabla 1.4. Asignación de pines de los LEDs.**

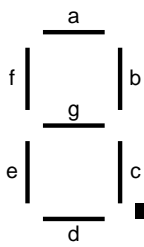
Nombre	Número Pin FPGA	Nombre	Número Pin FPGA	Nombre	Número Pin FPGA
LEDR[0]	AE23	LEDR[9]	Y13	LEDG[0]	AE22
LEDR[1]	AF23	LEDR[10]	AA13	LEDG[1]	AF22
LEDR[2]	AB21	LEDR[11]	AC14	LEDG[2]	W19
LEDR[3]	AC22	LEDR[12]	AD15	LEDG[3]	V18
LEDR[4]	AD22	LEDR[13]	AE15	LEDG[4]	U18
LEDR[5]	AD23	LEDR[14]	AF13	LEDG[5]	U17
LEDR[6]	AD21	LEDR[15]	AE13	LEDG[6]	AA20
LEDR[7]	AC21	LEDR[16]	AE12	LEDG[7]	Y18
LEDR[8]	AA14	LEDR[17]	AD12	LEDG[8]	Y12

### 1.2.3 Los visualizadores 7-segmentos

La placa DE2 dispone de ocho visualizadores de 7-segmentos, los cuales se encuentran agrupados en dos parejas de dos y un bloque de cuatro, permitiendo así la representación de números de diferentes tamaños. Los siete segmentos de los visualizadores se encuentran conectados a los pines del dispositivo Cyclone II. Aplicando un nivel lógico bajo en un segmento este se ilumina y con un nivel lógico alto se apaga.

Cada segmento del visualizador se identifica mediante un índice comprendido entre la letra a y la letra g. Hay que destacar que el punto del visualizador no se encuentra conectado y por tanto no se puede utilizar. En la Tabla 1.5 se muestra la asignación de pines y una imagen con el índice de cada segmento.

Tabla 1.5. Asignación de pines de los visualizadores 7-segmentos.

Nombre	Número Pin FPGA	Nombre	Número Pin FPGA	Nombre	Número Pin FPGA
HEX0 a	AF10	HEX1 a	V20	HEX2 a	AB23
HEX0 b	AB12	HEX1 b	V21	HEX2 b	V22
HEX0 c	AC12	HEX1 c	W21	HEX2 c	AC25
HEX0 d	AD11	HEX1 d	Y22	HEX2 d	AC26
HEX0 e	AE11	HEX1 e	AA24	HEX2 e	AB26
HEX0 f	V14	HEX1 f	AA23	HEX2 f	AB25
HEX0 g	V13	HEX1 g	AB24	HEX2 g	Y24
HEX3 a	Y23			HEX4 a	U9
HEX3 b	AA25			HEX4 b	U1
HEX3 c	AA26			HEX4 c	U2
HEX3 d	Y26			HEX4 d	T4
HEX3 e	Y25			HEX4 e	R7
HEX3 f	U22			HEX4 f	R6
HEX3 g	W24			HEX4 g	T3
HEX5 a	T2	HEX6 a	R2	HEX7 a	L3
HEX5 b	P6	HEX6 b	P4	HEX7 b	L2
HEX5 c	P7	HEX6 c	P3	HEX7 c	L9
HEX5 d	T9	HEX6 d	M2	HEX7 d	L6
HEX5 e	R5	HEX6 e	M3	HEX7 e	L7
HEX5 f	R4	HEX6 f	M5	HEX7 f	P9
HEX5 g	R3	HEX6 g	M4	HEX7 g	N9

### 1.3 Verificación física de un diseño

Una vez diseñado y verificado el funcionamiento de un sistema digital mediante la correspondiente simulación, hay probar su funcionamiento en el hardware para el que ha sido diseñado, lo que se conoce habitualmente como verificación física. Para probar un diseño en la placa DE2 de Altera hay que realizar los siguientes pasos:

1. Asignarle al diseño el dispositivo FPGA que tiene la placa DE2.
2. Realizar la correspondiente asignación de pines.
3. Programar el dispositivo FPGA de la placa DE2.

En primer lugar hay que comprobar que cuando se creó el proyecto del diseño en Quartus II, se le asignó correctamente el dispositivo sobre el que se implementará dicho sistema digital. La comprobación del dispositivo se realiza en el menú **Assignments > Device** de Quartus II, ventana que se muestra en la Figura 1.2. En este caso, la verificación física del diseño se realizará sobre la placa DE2, cuyo dispositivo es una Cyclone II con la referencia: EP2C35F672C6. Si el dispositivo asignado no es el correcto, debe realizarse una compilación del diseño antes de proseguir.

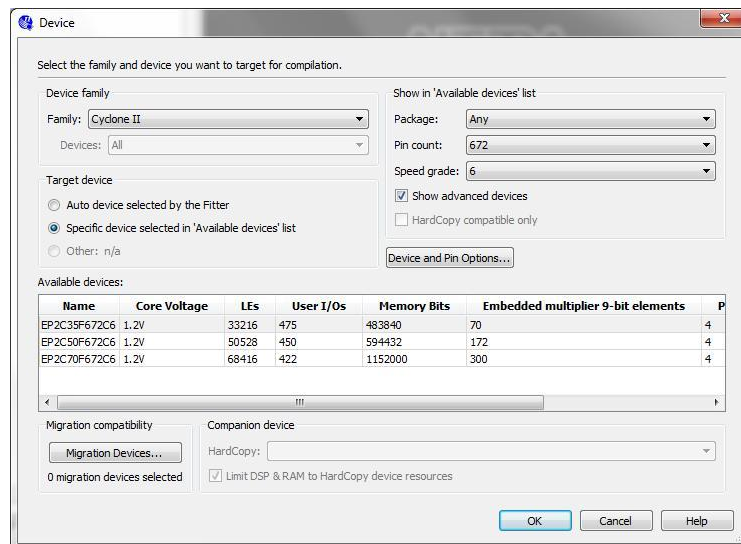


Figura 1.2. Ventana de asignación del dispositivo.

La asignación de pines consiste en indicar a que pin de la FPGA, debe conectarse cada una de las señales de entrada y de salida del diseño que se quiere probar. Esto requiere que en primer lugar se identifiquen las señales de entrada y salida del diseño y a que pines de la FPGA deben de conectarse. En tablas anteriores se muestran cuáles son los pines de la FPGA en los que se conectan algunos elementos de la placa DE2, como los LEDs, los interruptores, los pulsadores o los display 7-seguntos.

La asignación de pines se realiza en el menú **Assignments > Pin Planner** de Quartus II, ventana que se muestra en la Figura 1.3. En la parte superior de la Figura 1.3 se ve una vista inferior del encapsulado del dispositivo. Los 672 pines que tiene la Cyclone II, se distribuyen en una matriz cuadrada en la que se enumeran las filas mediante letras y las columnas mediante números. En la parte inferior de la ventana mostrada en la Figura 1.3, se ve la lista de señales de entrada y salida del diseño, para cada señal se indica el nombre (**Node Name**), la dirección (**Direction**), la conexión (**Location**), y otros datos. Es en la columna de **Location** donde se debe especificar el pin al que se quiere conectar la señal. Por ejemplo, una señal de reloj del sistema (CLOCK\_50) irá conectada al oscilador de la placa, según la Tabla 1.1 el oscilador de 50 MHz se encuentra físicamente conectado al pin N2 de la FPGA, por tanto en la columna **Location** de la señal se debe poner el valor N2.

Existe otra forma de realizar la asignación de pines, la cual es dependiente de la información que nos pueda facilitar el fabricante de la placa. Esta otra forma consiste en importar un fichero que contiene la lista de asignaciones de pines y que nos facilita el fabricante de la placa DE2 (DE2\_pin\_assignments.csv). Este fichero en realidad contiene una lista en la cual se relaciona un nombre de señal prefijado por el fabricante de la placa DE2 (valor que asocia con la columna **Node Name**) con un pin del dispositivo (valor que se asocia con la columna **Location**). Sin embargo, esto implica importar un gran número de asignaciones, todas las de la placa DE2 alrededor de 600, cuando en algunos casos solo se necesitan unas pocas, con lo cual las no necesarias aparecen en el **Pin Planner** remarcadas con interrogantes. Notar que para utilizar este método, es importante haber nombrado las señales de entrada y salida del

diseño del mismo modo que aparece en el fichero de asignaciones. La importación del fichero de asignaciones se realiza mediante: **Assignments > Import Assignments**, dando lugar a la ventana de la Figura 1.4.

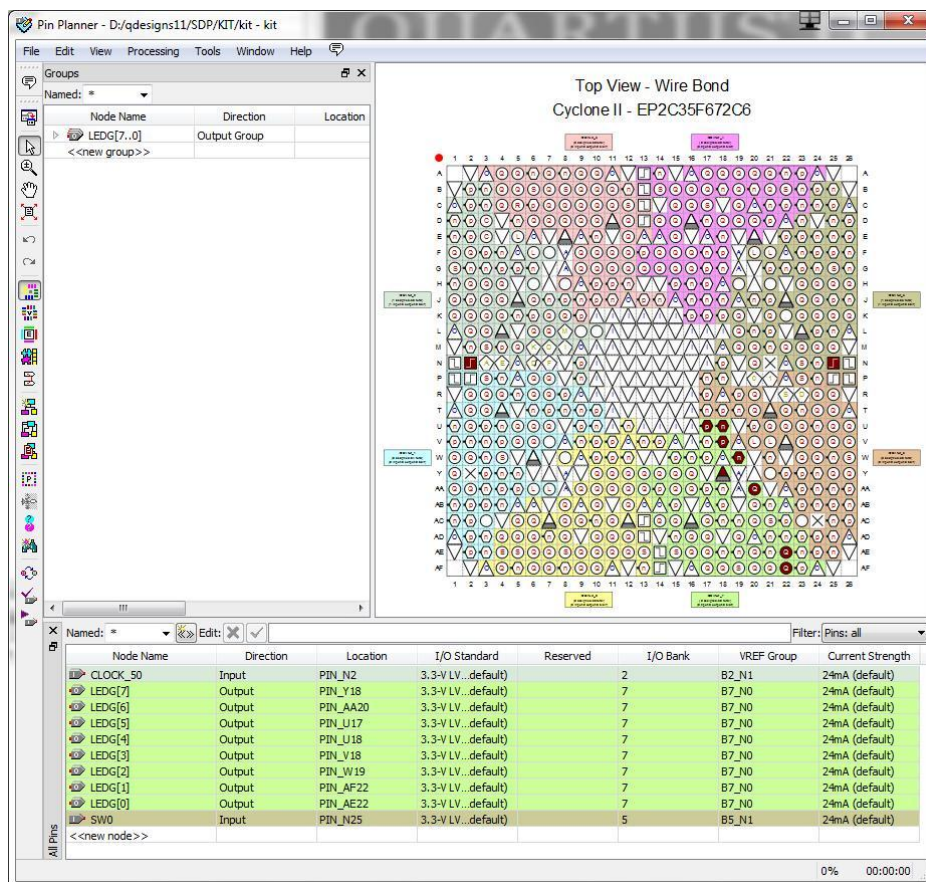


Figura 1.3. Ventana de asignación de pines, Pin Planner.

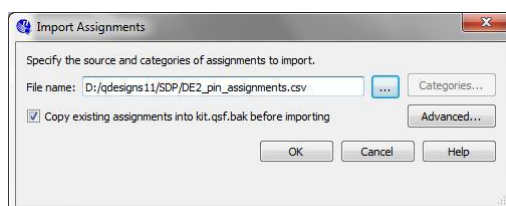


Figura 1.4. Ventana de importación de asignaciones.

Una vez realizada la asignación de pines se debe proceder a una nueva compilación ya que los pines deben incorporarse al diseño. Tras la compilación se programará la FPGA mediante el menú **Tools > Programmer**, que abre la ventana mostrada en la Figura 1.5.

Asegurarse que la placa DE2 está operativa pulsando el botón rojo de la misma, se observa la iluminación de múltiples LEDs de la placa.

En la ventana **Programmer**, lo primero es asegurarse que se ha detectado la placa DE2, si junto al botón **Hardware Setup** aparece la inscripción **No Hardware**, significa que la placa no ha sido detectada, en cuyo caso hay que pulsar el botón **Hardware**

**Setup**, para especificar como medio de comunicación con la placa el **USB Blaster**. Una vez verificado, que hay conexión entre el Quartus II y la placa DE2, hay que añadir (si no aparece ya en la ventana) el fichero que tiene el mismo nombre que el proyecto y con extensión .sof. Este fichero sof (SRAM Object File) contiene los códigos binarios que permiten programar el dispositivo para que el diseño quede emplazado en su interior. Se marca la casilla **Program/Configure** del fichero añadido. Y finalmente se pulsa el botón **Start** para iniciar la programación del dispositivo que será visible en la barra de progreso.

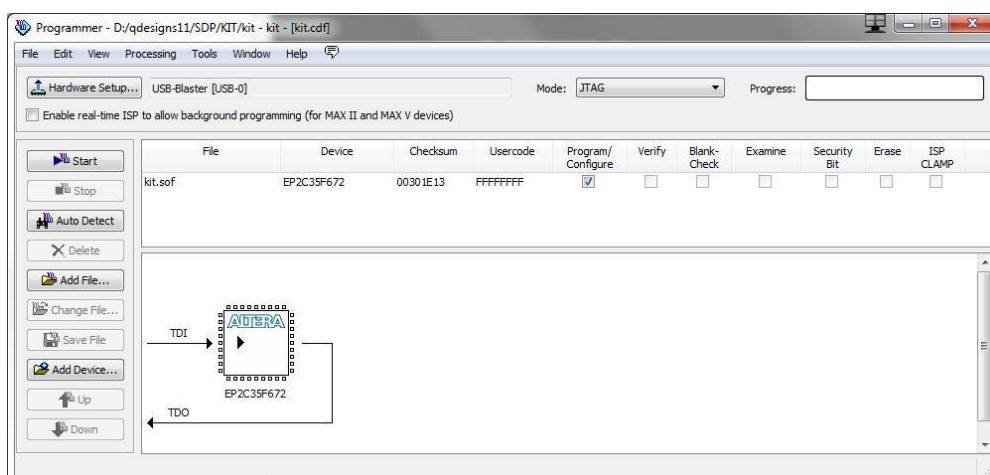


Figura 1.5. Ventana de programación del dispositivo.

#### 1.4 ¿Cómo archivar un proyecto en Quartus II?

Cuando se está trabajando en una herramienta de diseño electrónico como Quartus II, siempre surge la necesidad de copiar el trabajo realizado para poder llevarlo consigo y así poder trabajar en distintas máquinas o equipos. Como Quartus trabaja a nivel de proyectos, lo más inmediato es copiarse todo el subdirectorio del proyecto o empaquetarlo en un archivo mediante una utilidad de compresión de archivos.

Sin embargo, Altera ha dotado a su herramienta Quartus de una utilidad que permite archivar un proyecto, empaquetándolo en un único archivo con la extensión .qar. La ventaja que presenta el uso de esta utilidad, es que automáticamente selecciona los archivos realmente necesarios para su posterior uso, además como la extensión .qar esta enlazada con Quartus, con un simple doble click en el archivo se desempaqueta el proyecto y se abre sobre Quartus.

Para archivar el proyecto que se está actualmente en Quartus se debe proceder del siguiente modo:

1. Seleccionar en el menú de Quartus: **Project > Archive Project**. Esto abre la ventana del **Archive Project**, tal como se ve en la Figura 1.6.
2. En la ventana de la Figura 1.6, hay que establecer un nombre para el fichero en el apartado: **Archive file name**.
3. Pulsando el botón de **Advanced...**, se abre la ventana de opciones tal como muestra la Figura 1.7. Aquí se puede personalizar la selección de ficheros que se deben incluir en el proceso de archivado del proyecto.



4. Seleccionar en el **File set** la preconfiguración **Service request**, tal como se muestra en la Figura 1.7.
5. Pulsando el botón **OK**, se vuelve a la ventana del **Archive Project**.
6. Ahora, al pulsar en el botón **Archive** se creará el fichero donde se ha archivado el proyecto, que tendrá por nombre el que se ha especificado en **Archive file name** y la extensión **.qar**. Este fichero se encontrará en el subdirectorio del proyecto, a no ser que se haya indicado otro lugar.

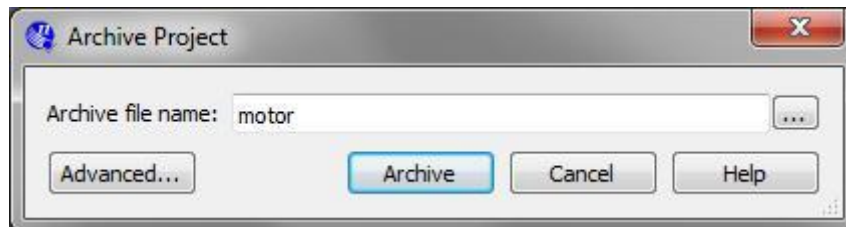


Figura 1.6. Ventana de archivado del proyecto.

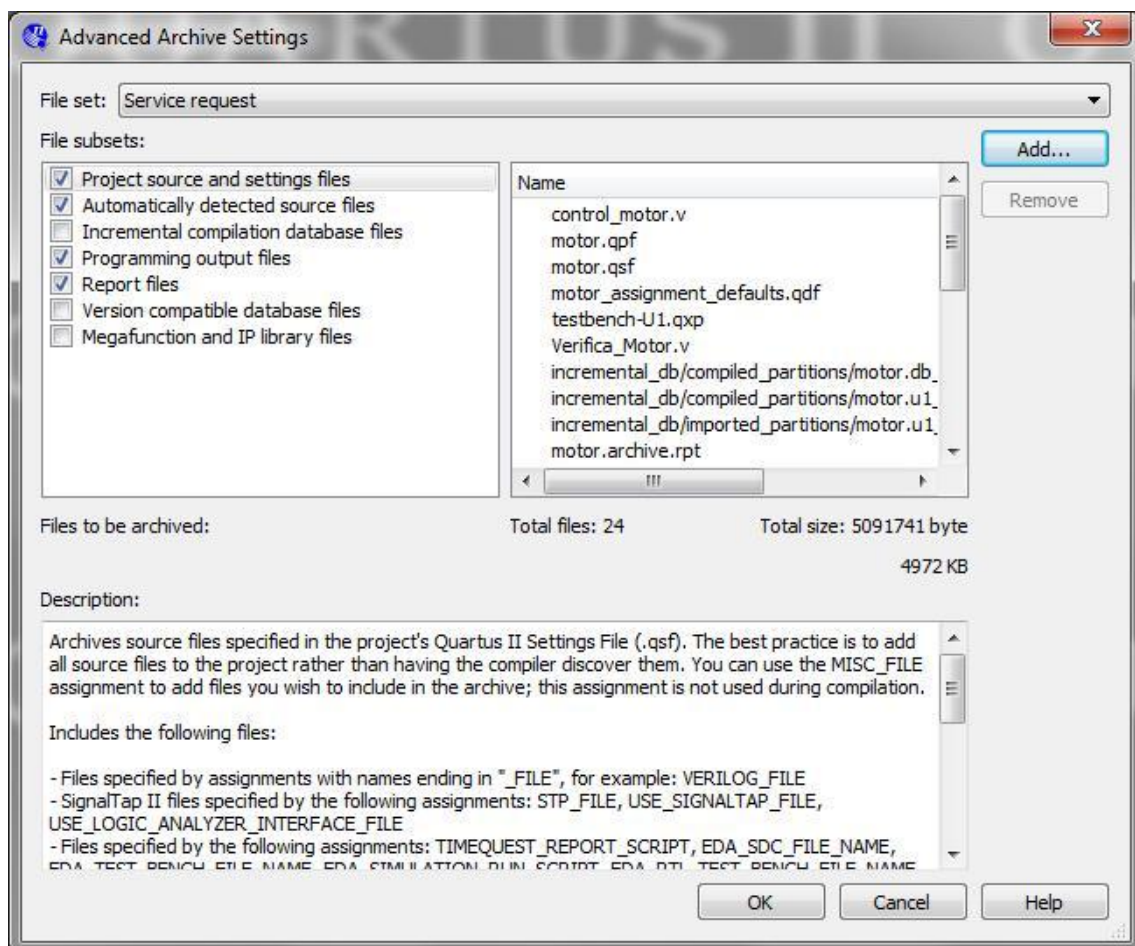


Figura 1.7. Ventana de opciones del archivado del proyecto.

## 2 Desarrollo de Testbench para Simulación Empleando Modelsim

En la actualidad no es extraño encontrar diseños que superen las decenas de millones de puertas equivalentes (SoC / System on Chip). Es evidente que este tipo de chips no se pueden realizar partiendo de cero con las condiciones habituales de mercado y por ello se emplean bloques IP (Intellectual Property) que constituyen en si mismos componentes de una elevada complejidad, listos para ser ensamblados dentro de un sistema mayor. Incluso pudiendo confiar en el correcto funcionamiento de estos IP, las labores de verificación del sistema completo o SoC consumen hoy en día entre el 70% y el 80% del tiempo total de diseño. Es lógico pues que el número de ingenieros que trabajan en la verificación de un diseño sea aproximadamente el doble que los que trabajan en la parte RTL. Otro dato muy importante a tener en cuenta, es que debido a las técnicas de reuso (empleo de IPs), la longitud de los testbenches supera la longitud del código RTL en la mayoría de los casos. Todo esto justifica con creces el desarrollo de una metodología de verificación que sistematice el procedimiento.

### 2.1 El Testbench

La verificación no es únicamente la creación de un testbench. La verificación es un proceso mediante el cual se demuestra la corrección funcional de un diseño. Empleando la verificación podemos asegurarnos que nuestro diseño cumple con las expectativas planteadas en la fase de especificación. Bajo la denominación de procedimientos de verificación se engloban una cierta cantidad de útiles que nos permitirán trabajar a distintos niveles para poder avanzar con seguridad en el flujo de diseño.

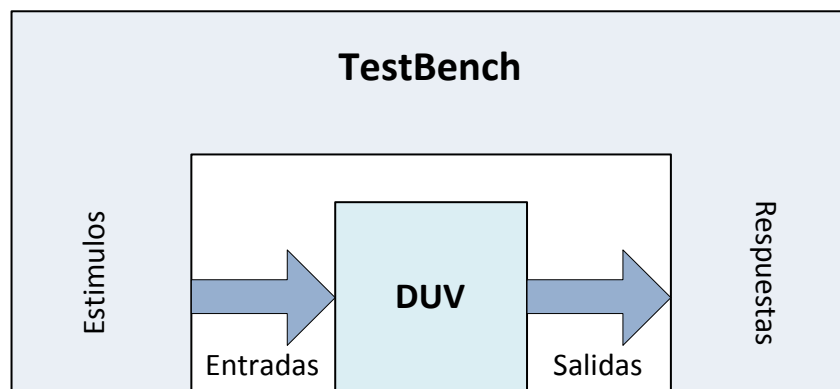


Figura 2.1. Esquema de Verificación.

El testbench será la herramienta principal que nos permita ejecutar la verificación. Un testbench es un conjunto de código generado para permitir la introducción de datos (señales etc) en el diseño bajo verificación (D.U.V / device under verification) y la observación de sus respuestas; engloba al DUV (ver Figura 2.1) y constituye el 'tubo de ensayo' donde se va a desarrollar nuestro trabajo. Dado que el testbench en si mismo no va a ser sintetizado pueden emplearse multitud de lenguajes para su descripción VHDL, Verilog, SystemVerilog, C, etc. Sin embargo, debido al carácter 'hardware' del proceso, en el que se deberán de emplear condiciones temporales etc.



se hace recomendable el uso de lenguajes HDL al menos en las partes que se encuentran en contacto directo con el DUV.

## 2.2 El Simulador Modelsim-Altera

Las labores de verificación de los circuitos precisan de simuladores HDL que permitan la ejecución de código HDL a todos los niveles e incluso en ocasiones la mezcla de distintos lenguajes tanto HDL como software. Modelsim (Mentor Graphics) es uno de los simuladores interpretados más potentes del mercado que se integra con la herramienta de diseño Quartus II en su versión adaptada (y de prestaciones limitadas) Modelsim-Altera. A continuación se describirán los pasos a seguir para configurar la herramienta Quartus II para ejecutar las simulaciones empleando Modelsim-Altera, empleando un ejemplo descrito en la Figura 2.2:

```
module contador_param(CLK,RST_A,ENA,COUNT,TC);
parameter BITS = 4;
parameter cuenta = 4'd14;

input CLK,RST_A,ENA;
output [BITS-1:0] COUNT;
output TC;
reg unsigned [BITS-1:0] COUNT;

always @(posedge CLK or negedge RST_A)
begin
    if (RST_A==0) COUNT <= {BITS{1'b0}};
    else
        if (CLK==1)
            if (ENA==1)
                if (COUNT != cuenta)
                    COUNT <= COUNT + {{BITS-1{1'b0}},1'b1};
                else
                    COUNT <= {BITS{1'b0}};
            end
        end
    assign TC = (COUNT==cuenta) ? 1'b1 : 1'b0;
endmodule
```

Figura 2.2. Fichero contador\_param.v.

1. Durante la creación del proyecto en Quartus II seleccionar ModelSim – Altera como herramienta de simulación y Verilog HDL como formato de entrada.

## EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	<input type="checkbox"/> Run this tool automatically to synthesize the current design
Simulation	ModelSim-Altera	Verilog HDL	<input checked="" type="checkbox"/> Run gate-level simulation automatically after compilation
Timing Analysis	<None>	<None>	<input type="checkbox"/> Run this tool automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figura 2.3. Opciones de Simulación en la creación del proyecto en Quartus II.

En caso de no haber seleccionado dichas opciones en la fase de creación del proyecto, seguir las opciones mostradas en la Figura 2.5.

Verificar que el path de instalación de ModelSim se encuentra bien introducido en Quartus II. Para ello en el menú **Tools > Options > EDA Tool Options** debe de figurar el directorio donde se encuentre modelsim (similar al que aparece en la Figura 2.4)

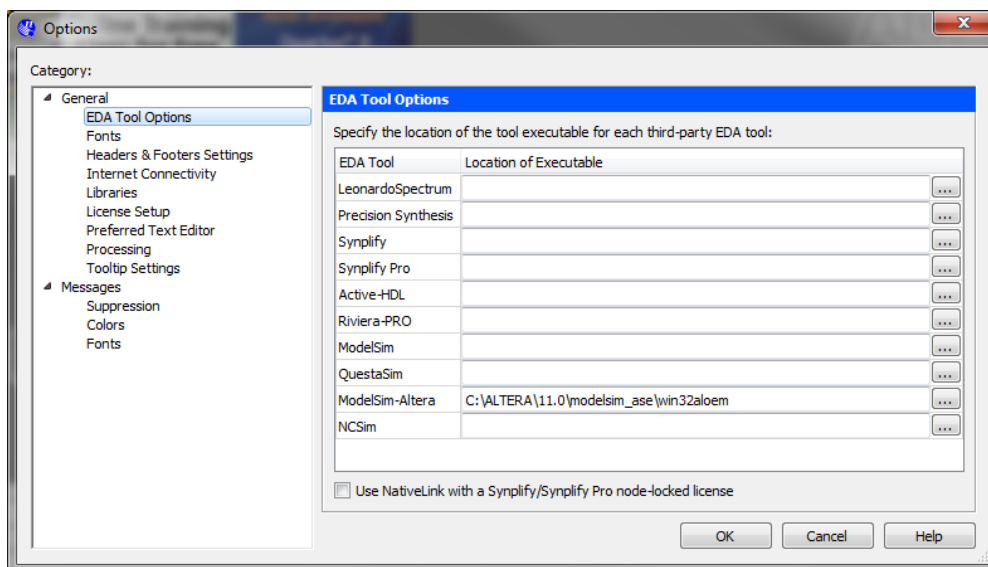


Figura 2.4. Path de ModelSim - Altera.

2. Seleccionar en el menú **Processing > Start > Start Test Bench Template Writer**. Quartus II generará un directorio dentro del proyecto: *simulation/modelsim* con un archivo denominado **contador\_param.vt**.

```
// Verilog Test Bench template for design : contador_param
//
// Simulation tool : ModelSim-Altera (Verilog)
//
`timescale 1 ns / 1 ps
module contador_param_vlg_tst();
// constants
```

```

// general purpose registers
reg eachvec;
// test vector input registers
reg CLK;
reg ENA;
reg RST_A;
// wires
wire [3:0] COUNT;
wire TC;

// assign statements (if any)
contador_param i1 (
// port map - connection between master ports and
// signals/registers
    .CLK(CLK) ,
    .COUNT(COUNT) ,
    .ENA(ENA) ,
    .RST_A(RST_A) ,
    .TC(TC)
);
initial
begin
// code that executes only once
// insert code here --> begin
// --> end
$display("Running testbench");
end
always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
@eachvec;
// --> end
end
endmodule

```

Se trata de una plantilla para desarrollar el *testbench* de nuestro módulo, donde se ha realizado:

- a. Declaración de las señales de conexión de entrada y salida del DUV.
- b. Instanciación del módulo del DUV.
- c. Descripción de un bloque procedural *initial* para la inicialización de las señales que lo requieran.
- d. Descripción de un bloque *always* donde se ejecuta la introducción de Casos de Verificación (CV) en el DUV. El esquema adoptado por Quartus se basa en la declaración de un *reg* denominado *eachvec* que modifica su valor cada vez que se desea pasar a un caso de verificación nuevo. De ahí que después de cada bloque donde se realiza la introducción del CV se introduzca un control de ejecución que espera a un cambio en *eachvec*.

3. Identificar los CV a comprobar en el DUV. En el ejemplo se han desarrollado 3 casos:
  - a. Funcionamiento habitual del contador (reset asíncrono incluido).
  - b. Colisión del *reset* con *enable*.
  - c. Desactivación del *enable* justo en la última cuenta.
4. Editar el testbench para introducir los CV y el resto de sistemas del testbench (generación de reloj y reset, aserciones, utilidades...). La estructura recomendada se basa en la escritura de tareas (*task*) para cada CV. La generación del reloj se puede realizar mediante un bloque *always*.

#### NOTAS:

- ➔ La activación de los estímulos (señales de entrada) debe de respetar la temporización del circuito y evitar problemas de metaestabilidad (condiciones de carrera). Es importante considerar que en futuras simulaciones podremos cambiar la frecuencia de trabajo del reloj para probar las características del diseño en la fase de implementación. Por lo tanto la mejor estrategia es realizar los cambios de señales en el flanco **NO-ACTIVO** del circuito (negedge CLK en este caso).
- ➔ La estructura interna de la tarea puede tener una ejecución de tipo secuencial (**begin...end**) o una ejecución de tipo paralelo (**fork...join**). Hay que tener en cuenta el encadenamiento de los tiempos de retardo en el caso de los bloques secuenciales.
- ➔ Los retardos y el comportamiento de los estímulos a nivel temporal se describirán como múltiplos del periodo de la señal de reloj para permitir una parametrización de la simulación. (ver *localparam*).

*Si la instancia del DUV emplea parámetros, su valor deberá de ser el mismo que el empleado en la Compilación del diseño en Quartus II.*

```
// Verilog Test Bench template for design : contador_param
//
// Simulation tool : ModelSim-Altera (Verilog)
//
`timescale 1 ns/ 1 ps
module contador_param_vlg_tst();

localparam T = 10;

reg CLK;
reg ENA;
reg RST_A;
// wires
wire [3:0] COUNT;
wire TC;

// assign statements (if any)
contador_param #(.BITS(4),.cuenta(4'd14)) i1 (
// port map - connection between master ports and
// signals/registers
.CLK(CLK),
.COUNT(COUNT),
.ENA(ENA),
.RST_A(RST_A),
```

```

        .TC(TC)
    );

initial
    begin
        RST_A=0;
        CLK=0;
        ENA=0;
        $display("SIMULANDO!!!");

        CASO_1(); // Caso de Verificacion 1
        #(T*5);
        CASO_2(); // Caso de Verificacion 2. RST_A con ENABLE
        #(T*5);
        CASO_3(); // Caso de Verificacion 3. ENABLE con TC
    end

task CASO_1;
    begin
        #T RST_A = 1'b1;
        @(negedge CLK) ENA = 1'b1;
        #(T*20);
        @(negedge CLK) ENA = 1'b0;
    end
endtask

task CASO_2;
    begin
        fork
            RST_A = 1'b0;
            #(T*4) RST_A = 1'b1;
            ENA = 1'b1;
            #(T*2) ENA = 1'b0;
        join
    end
endtask

task CASO_3;
    begin
        ENA = 1'b1;
        #(T*14);
        @(negedge CLK) ENA = 1'b0;
        #(T*5);
        @(negedge CLK) ENA = 1'b1;
    end
endtask

always
    begin
        #(T/2) CLK <= ~CLK;
    end

endmodule

```

5. Configuración de Quartus II para la simulación con ModelSim:
  - a. En el menú **Assignments > Settings > EDA Tools Settings > Simulation** emplear las opciones descritas en la Figura 2.5.
  - b. En la opción **Test Benches** será necesario incluir el *testbench* desarrollado en el punto anterior. Para ello seleccionar **Test Benches > New...** y emplear las opciones descritas en la Figura 2.6. En estas opciones se indica:
    - i. El nombre del módulo que implementa el *testbench*.
    - ii. El nombre de la instancia del DUV para aplicar los retardos obtenidos en la fase de extracción.
    - iii. El periodo de tiempo de simulación (el indicado en el propio *testbench* o bien uno que se forzará en este punto).
    - iv. El fichero que contiene el *testbench* mediante la opción **Test bench Files > Add...**
  - c. Si todo se ha realizado correctamente en la opción **Compile test bench** del menú de la Figura 2.5, aparecerá la opción del *testbench* que se ha introducido.

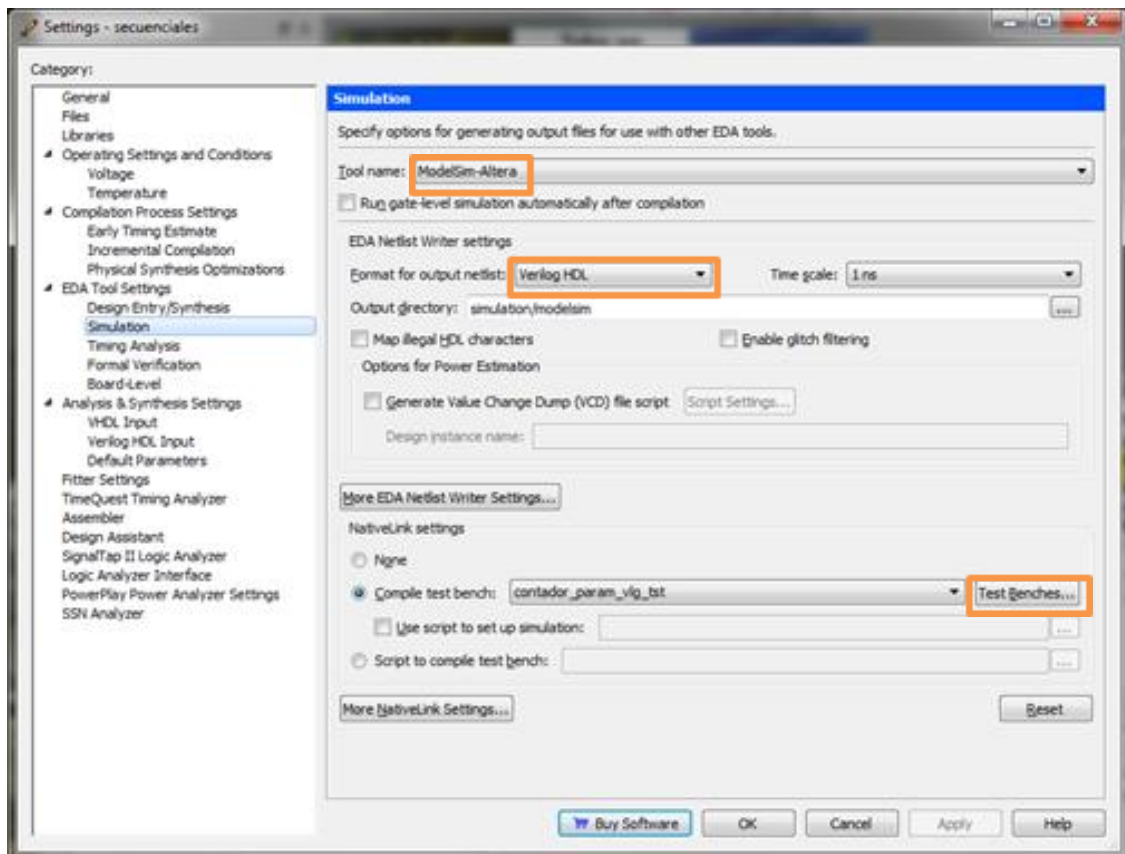


Figura 2.5. Opciones de configuración I.

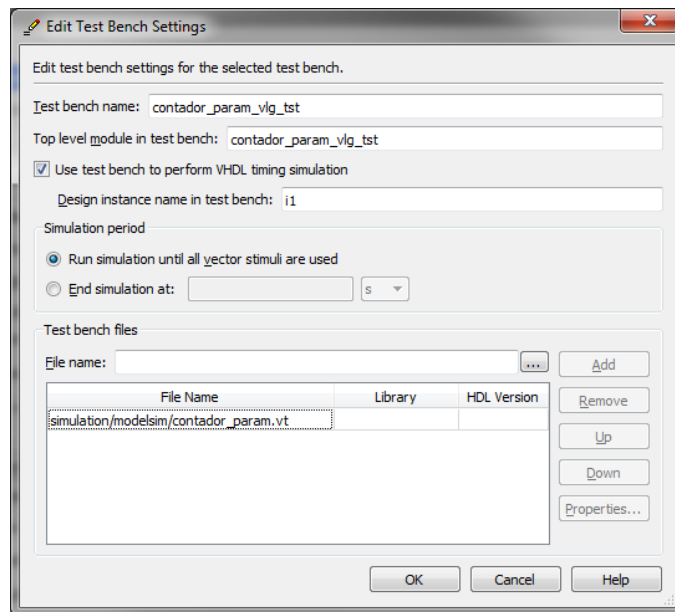


Figura 2.6. Opciones de configuración II.

6. La simulación puede realizarse de dos formas distintas:
  - a. **Tools > Run EDA Simulation Tools > Run RTL Simulation.** Permite realizar una simulación del código RTL previa al proceso de síntesis e implementación. Se emplea esta opción por defecto para descartar problemas de temporización.
  - b. **Tools > Run EDA Simulation Tools > Run Gate Level Simulation.** Permite realizar una simulación de tipo GATE LEVEL, es decir del resultado de los procesos de síntesis e implementación. Se añaden los retardos e información temporal del diseño empleando un modelo de caso peor (*Slow Model*) o un modelo de caso mejor (*Fast Model*).

En cualquiera de las dos opciones, Quartus II llamará a ModelSim ejecutando una compilación y simulación en éste último. Los puertos de entrada y salida del DUV se muestran con los resultados de la simulación (ver Figura 2.7)

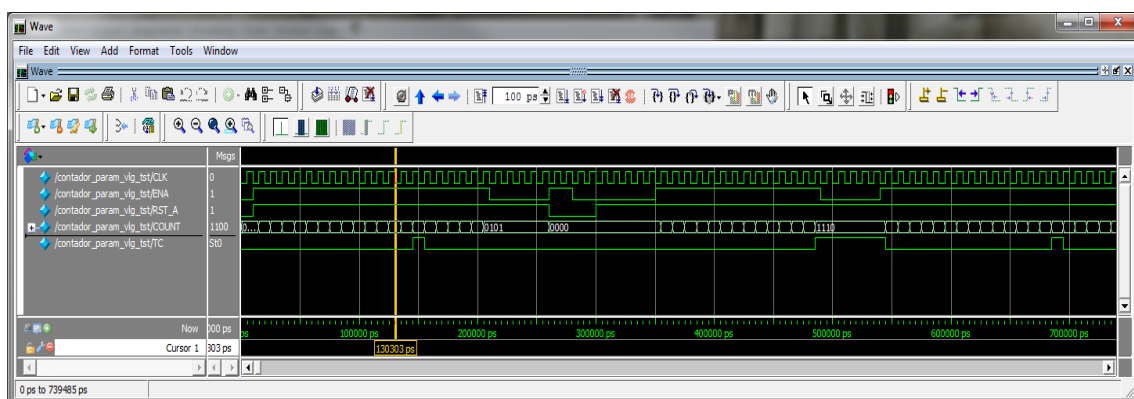


Figura 2.7. Resultado de la simulación.



## 3 Diseño de Sistemas Secuenciales

### 3.1 Objetivos

- Aprender los conceptos básicos del lenguaje Verilog HDL y su uso en la realización de diseños.
- Incidir en el uso de bloques procedurales gobernados por el reloj para la modelización de circuitos secuenciales
- Realización práctica de algunos diseños digitales utilizando el lenguaje Verilog HDL.
- Conocer cómo se realiza la jerarquía en Verilog HDL y el uso de parámetros.

### 3.2 Repaso conceptos teóricos

#### 3.2.1 Procesos secuenciales

Los procesos secuenciales son aquellos de los que hacemos uso para modelizar circuitos secuenciales. Las reglas y recomendaciones básicas que se deben tener en cuenta cuando se modeliza un circuito secuencial mediante un proceso secuencial son las siguientes:

- En un proceso secuencial aparece una (y solo una) señal de reloj. Esto es evidente pensando en el **circuito secuencial síncrono** que se está modelizando. La afirmación anterior implica que en un proceso secuencial debe aparecer una (y solo una) alusión a un flanco de una señal, es decir una construcción del tipo `@(posedge ...) o @(negedge ...)`.
- Un circuito secuencial es sensible a los flancos de la señal de reloj y a las entradas asíncronas. Esto implica que el proceso secuencial que lo modeliza debe tener una lista de sensibilidad que incluya la señal de reloj y las entradas asíncronas si las hubiera (normalmente un reset asíncrono), por ejemplo: `@(posedge clk or negedge reset)`. Algunos sintetizadores limitan el número de señales asíncronas que pueden acompañar a la señal de reloj en la lista de sensibilidad. De todo lo anterior se deduce que si se está modelizando un circuito secuencial absolutamente síncrono en todo su funcionamiento, solo debe aparecer la señal de reloj en la lista de sensibilidad.
- La estructura única y básica de todo proceso secuencial es un “`if else if else`” que debe englobar toda la descripción. Como se sabe esta estructura implica prioridad (según el lugar en el que aparecen). Deberán comprobarse las señales de entradas más prioritarias en primer lugar. Estas señales se sabe que son, en los circuitos secuenciales, las señales asíncronas.
- **Todas las asignaciones no bloqueantes de señales** que se realicen dentro de un proceso secuencial van a inferir flip-flops por cada una de ellas. Si se quiere que una asignación infiera un simple cable sin añadir flip-flops adicionales, debe ser realizada dicha asignación fuera del proceso.
- El orden de las **asignaciones bloqueantes** determina que es sintetizado (es decir si infiere o no flip-flops) mientras que con las asignaciones no

bloqueantes de señales, como se ha indicado en el párrafo anterior, es indiferente: siempre infiere flip-flops. Esto hace que uno deba ser muy cuidadoso con lo que hace cuando se trabaja con las asignaciones bloqueantes de señales, mientras que con las no bloqueantes todo es más previsible y por tanto más cómodo de trabajar.

- **Se recomienda** por lo tanto trabajar exclusivamente con asignaciones no bloqueantes de señales en el interior de los procesos secuenciales. Las plantillas templates y ejemplos de ALTERA no hacen caso de esta importantísima recomendación y os pueden conducir a innumerables problemas en vuestros diseños. Cuidado.

Tabla 3.1. Ejemplo de uso de asignaciones bloqueantes de señales.

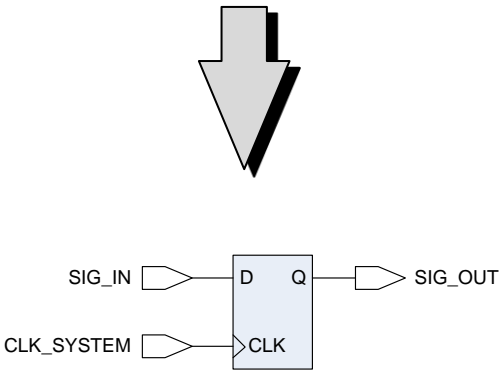
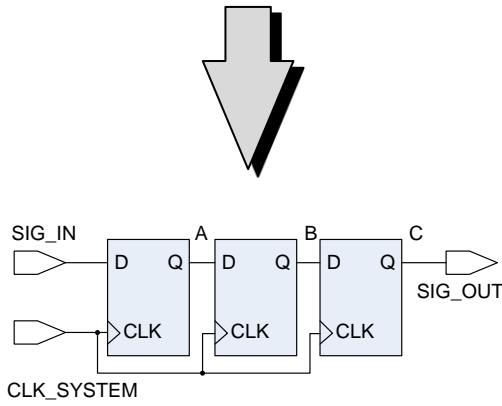
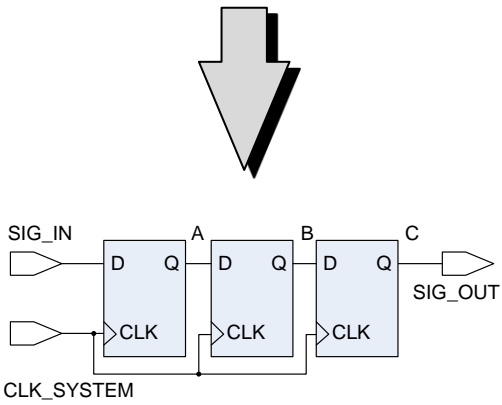
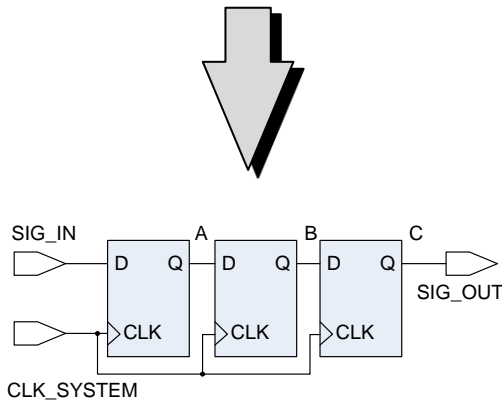
Registro de desplazamiento MAL descrito	Registro de desplazamiento BIEN descrito
<pre> module SHIFT_REGISTER (SIG_IN, CLK_SYSTEM, SIG_OUT); output SIG_OUT; input SIG_IN, CLK_SYSTEM; reg A, B, C; always @(posedge CLK_SYSTEM) begin     A = SIG_IN;     B = A;     C = B; end assign SIG_OUT = C; endmodule </pre>	<pre> module SHIFT_REGISTER (SIG_IN, CLK_SYSTEM, SIG_OUT); output SIG_OUT; input SIG_IN, CLK_SYSTEM; reg A, B, C; always @(posedge CLK_SYSTEM) begin     C = B;     B = A;     A = SIG_IN; end assign SIG_OUT = C; endmodule </pre>
	

Tabla 3.2. Ejemplo de uso de asignaciones no bloqueantes de señales: RECOMENDADO.

Registro de desplazamiento BIEN descrito	Registro de desplazamiento BIEN descrito
<pre> <b>module</b> SHIFT_REGISTER (SIG_IN, CLK_SYSTEM, SIG_OUT); <b>output</b> SIG_OUT; <b>input</b> SIG_IN, CLK_SYSTEM; <b>reg</b> A, B, C; <b>always</b> @(posedge CLK_SYSTEM) <b>begin</b>     A &lt;= SIG_IN;     B &lt;= A;     C &lt;= B; <b>end</b> <b>assign</b> SIG_OUT = C; <b>endmodule</b> </pre>	<pre> <b>module</b> SHIFT_REGISTER (SIG_IN, CLK_SYSTEM, SIG_OUT); <b>output</b> SIG_OUT; <b>input</b> SIG_IN, CLK_SYSTEM; <b>reg</b> A, B, C; <b>always</b> @(posedge CLK_SYSTEM) <b>begin</b>     C &lt;= B;     B &lt;= A;     A &lt;= SIG_IN; <b>end</b> <b>assign</b> SIG_OUT = C; <b>endmodule</b> </pre>
	

### 3.2.2 Diseño jerárquico en Verilog HDL

Es muy sencillo establecer jerarquía en Verilog HDL. Para ello sólo se tiene que instanciar el módulo que se quiere utilizar en la descripción. La instanciación de módulos en Verilog HDL se realiza de la siguiente forma:

```

module AOI_str (y_out, x_in1, x_in2, x_in3, x_in4, x_in5);
    output y_out;
    input x_in1, x_in2, x_in3, x_in4, x_in5;

    //Definición de nodos intermedios
    wire y1, y2;

    //Instanciación de los módulos

```

```

    nor (y_out, y1, y2);
    and (y1, x_in1, x_in2);
    and (y2, x_in3, x_in4, x_in5);
endmodule

```

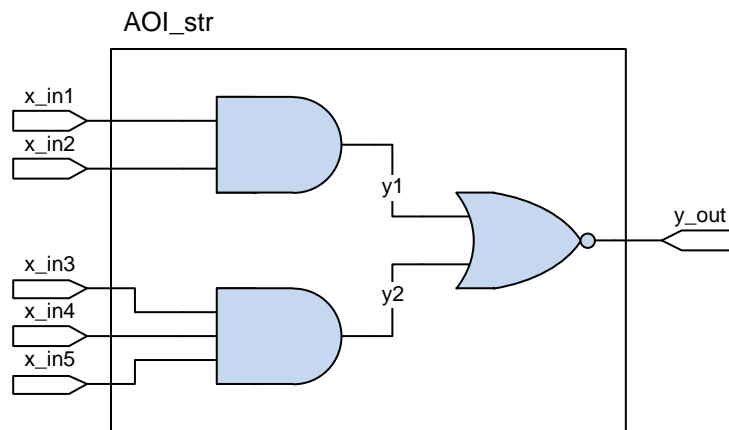


Figura 3.1. Ejemplo de uso de jerarquía en Verilog HDL.

El esquema de la Figura 3.1, muestra la construcción de un bloque denominado *AOI\_str* a partir de instancias de puertas *and* y *nor*. Para la interconexión de los módulos entre sí será necesaria la declaración de nodos internos de tipo *wire*. Por otra parte, una misma descripción en Verilog HDL puede combinar instancias de módulos, primitivas, así como otras estructuras. La utilización que se está haciendo de Verilog HDL en este ejemplo es puramente **estructural**. Es decir, se está creando lo que en las herramientas de diseño asistido por ordenador (CAD) se denomina un *netlist*: un listado de instancias de módulos junto con sus interconexiones.

El verdadero potencial de Verilog HDL reside en su capacidad de describir el comportamiento de circuitos digitales mediante un lenguaje de alto nivel, dejando a la herramienta (en este caso Quartus II) la tarea de trasladar del dominio **comportamental** al dominio **estructural**.

### 3.2.3 La reutilización de los diseños

Un diseñador eficiente debe saber rentabilizar el tiempo que dedica a realizar los diseños. Por ello, es importante saber reutilizar los módulos o bloques anteriormente diseñados. Una cualidad de los lenguajes HDL, que permite incrementar las posibilidades de reutilizar un diseño, es la parametrización de los mismos. Por ejemplo, en muchas ocasiones se requiere el uso de un contador, si previamente se ha realizado ese contador se puede reutilizar. Sin embargo puede ocurrir (y ocurrirá) que el contador realizado sea de 8 bits y el que se necesite de 10 bits. La solución parece sencilla, se toma la descripción del contador de 8 bits y se modifica para hacer uno de 10 bits. Esto tiene el inconveniente de tener que disponer de una colección distinta de archivos con modelizado de contadores para diferentes tamaños.

La mejor solución es disponer de un solo archivo con el modelo del contador, de tal manera que este se haya realizado dependiente de un parámetro de configuración,

que se cambiará en función de las necesidades. En la Figura 3.2 se muestra el ejemplo de un multiplexor de bus de tamaño parametrizable, en ella se pueden ver dos formas distintas de utilizar los parámetros.

<pre> <b>module</b> mux   #(<b>parameter</b> WIDTH = 4)   (<b>output</b> [WIDTH-1:0] out,    <b>input</b> [WIDTH-1:0] A, B,    <b>input</b> sel);    <b>assign</b> out = (sel) ? A: B;  <b>endmodule</b> </pre>	<pre> <b>module</b> mux (out, A, B, sel);   <b>parameter</b> WIDTH = 4;   <b>output</b> [WIDTH-1:0] out;   <b>input</b> [WIDTH-1:0] A, B;   <b>input</b> sel;    <b>assign</b> out = (sel) ? A: B;  <b>endmodule</b> </pre>
---	---

Figura 3.2. Ejemplos de diseño parametrizable.

En la Figura 3.3, se muestra a su vez diferentes formas de instanciar el módulo creado en la Figura 3.2. En primer lugar se instancia un multiplexor de 8 bits de bus. En el segundo caso se parametriza con 10 bits, utilizando una sintaxis de conexión por nombre, en lugar de posición como se utiliza en el primer caso. Tal como se aprecia en un mismo módulo se puede instanciar varias veces el mismo componente con distintos valores en sus parámetros.

```

module Top (...);
...
  mux #(8) u1 (a, b, c, d);
...
  mux #(.WIDTH(10)) u2 (.out(s), .A(x), .B(y), sel(z));
...
endmodule

```

Figura 3.3. Ejemplo de instanciación de módulos parametrizables.

Para completar el uso de parámetros, en ocasiones puede ser necesario realizar operaciones matemáticas con el parámetro antes de utilizarlo dentro del diseño. Por ejemplo, se quiere a partir de un dato obtener el número de bits necesario para poder alcanzarlo. Volviendo al ejemplo del contador, en lugar de fijar como parámetro el número de bits, se establece el valor del fin de cuenta, en este caso ¿Cómo se obtiene el número de bits que debe tener el contador? La solución es recurrir al uso de una función que permita obtener el número de bits a partir del dato de fin de cuenta. En la Figura 3.4 se muestra un ejemplo de cómo hacerlo utilizando la función: `CLogB2`, que calcula el logaritmo en base 2, con redondeo al entero más próximo. Tal como se aprecia la función `CLogB2`, permite obtener el valor de cuantos bits hacen falta para alcanzar el fin de cuenta establecido. La función `CLogB2` se encuentra definida en el fichero de funciones matemáticas denominado `MathFun.vh`, que habrá que incluir previamente en el diseño.

```

module contador
  #(parameter fin_cuenta=20)
  (...);

  `include "MathFun.vh"
  parameter WIDTH=CLogB2(fin_cuenta);

  ...

```

```
endmodule
```

Figura 3.4. Ejemplo de uso de funciones para el cálculo de parámetros.

### 3.3 Estructura de la práctica

Todo diseño de sistema secuencial va gobernado por una señal de reloj, proveniente de un oscilador externo al dispositivo, en el que se implementa el diseño. En la placa DE2 el oscilador que se dispone tiene una frecuencia de 50 MHz, lo cual significa que los cambios de las señales se producirán cada 20 ns. Cuando se realizan diseños para controlar o utilizar elementos de visualización perceptibles por el ojo humano, hay que tener presente que cambios con un tiempo inferior a 50 ms, no son visibles por los humanos. Así que, no se puede utilizar directamente el oscilador de 50 MHz para realizar cambios que deben ser visibles por el ojo humano; por ejemplo, hacer que un LED parpadee.

En esta práctica se ve cómo solucionar este problema de visualización, manteniendo los criterios de un correcto diseño síncrono en que todos los flip-flops del diseño deben estar gobernados por la misma señal de reloj.

La práctica de diseño de sistemas secuenciales con Verilog HDL está estructurada de la siguiente forma:

1. Realización de contadores en Verilog HDL.
2. Realización de registros de desplazamiento.
3. Realización del juego de luces del coche fantástico.

### 3.4 Desarrollo de la práctica

#### 3.4.1 Tarea 1.1 – Realización de un Contador con Verilog HDL

Realizar un contador binario, mediante descripción Verilog HDL que tenga las siguientes características:

- Contador síncrono ascendente-descendente de 8 bits, módulo 256.
- Funcionamiento por flanco de bajada del reloj.
- Reset, asíncrono.
- Carga paralelo síncrona, habilitada mediante una señal activa a nivel bajo.

#### 3.4.2 Tarea 1.2 – Contador parametrizable

Realizar un contador binario, mediante descripción Verilog HDL que tenga las siguientes características:

- Contador síncrono ascendente-descendente de módulo parametrizable `PARAMETER`.
- Funcionamiento por flanco de subida del reloj.
- Reset, síncrono (máxima prioridad).
- Entrada de habilitación de cuenta **enable** que habilita la cuenta.
- Señal de salida **TC** activa cuando el contador llega al fin de cuenta.

### 3.4.3 Tarea 1.3 – Registro de desplazamiento

Realizar un registro de desplazamiento mediante descripción Verilog HDL, cuyas características sean las siguientes:

- Registro de desplazamiento de 7 bits síncrono.
- Reloj activo por flanco de subida.
- Línea de reset síncrona.
- Entrada serie.
- Salida serie.
- Salida paralelo.
- Entrada de habilitación de desplazamiento **enable** que permite el desplazamiento.

### 3.4.4 Tarea 1.4 – Diseño del juego de luces del coche fantástico

Realizar en los LEDs verdes LEDG7 a LEDG0 de la placa DE2 el juego de luces siguiente:

Ciclos	LEDG7	LEDG6	LEDG5	LEDG4	LEDG3	LEDG2	LEDG1	LEDG0
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								

En este diseño debe reutilizar el contador parametrizable de la Tarea 1.2, y el registro de desplazamiento de la Tarea 1.3, con las siguientes consideraciones:

- El contador (que cuenta con la señal de reloj de la placa DE2 de 50 MHz) se empleará para habilitar con su salida **TC** la señal de **enable** del registro de desplazamiento, de forma que el desplazamiento sea cada 0,25 segundos.
- El registro de desplazamiento debe funcionar como un contador Johnson, por lo que generará 14 estados que corresponderán a los 14 ciclos del juego de luces.
- Además deberá crear un **decodificador** que a partir de los estados del contador Johnson genere la salida a los LEDs de LEDG7 a LEDG0.
- El **enable** del contador debe estar manejado por el estado del interruptor 0 SW[0], de la placa DE2.



El tope de la jerarquía del diseño debe ser un fichero Verilog HDL en donde se realice la instanciación del contador y del registro de desplazamiento, y el código asociado al proceso del decodificador. Tome como punto de partida el código mostrado y el diagrama de bloques siguiente de la Figura 3.5 para saber cómo crear el mapeo.

```

module kit ( //Declare los puertos del módulo );

    // Declare las señales necesarias

    contador
    #(.modulo(valor_del_modulo))
    u0(  .clock (CLOCK_50),
        // Complete las demás asociaciones
    );

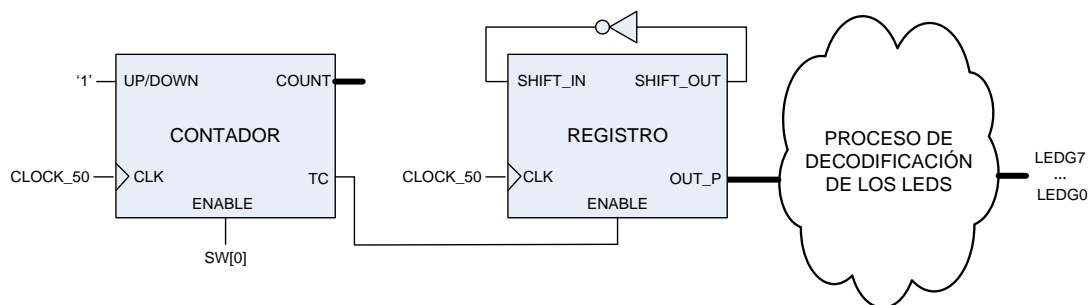
    registro
    u1(  .clock (CLOCK_50),
        // Complete las demás asociaciones
    );

    always @(//lista de sensibilidades)
    begin
        // Código del decodificador
    end

    // Asignaciones de señal necesarias

endmodule

```



**Figura 3.5. Diagrama de bloques de KIT.**

Procure que el nombre de los puertos del módulo coincida con el nombre que tiene la asignación de los pines de la FPGA en la placa DE2. Estas asignaciones están en el fichero **DE2\_pin\_assignments.csv** que está disponible en **Recursos** de la Asignatura en **PoliformaT**. Para utilizar estas asignaciones ejecute el comando **Import Assignments** del menú **Assignments**. De esta manera se ahorrará el tener que realizar las asignaciones de pines una a una tal y como ya hizo en la práctica anterior. Deberá definir los puertos asignados a los pines identificados como bus (por ejemplo **SW[0]** input [0:0] SW y los **LEDG** output [7:0] LEDG, y luego emplearlas como bus en Verilog HDL (por ejemplo SW[0], LEDG<=8'b00100000).

Verifique su correcto funcionamiento en la placa DE2.

## 4 Diseño de Maquinas de Estado Finitos

En sistemas digitales de una cierta complejidad siempre es necesaria la utilización de una unidad de control que gestione el funcionamiento del mismo. Las unidades de control son generalmente máquinas de estados finitos, y por tanto de su correcto funcionamiento depende todo el sistema.

### 4.1 Objetivos

- Aprender los conceptos básicos del lenguaje Verilog HDL y su uso en el diseño de máquinas de estados finitos.
- Aprender las principales formas constructivas para máquinas de estados finitos del lenguaje Verilog HDL.
- Aprender las particularidades del diseño de máquinas de estados finitos en Quartus II.
- Aprender a verificar el funcionamiento del diseño introduciéndolo en un sistema de prueba diseñado expresamente para él.
- Realización práctica de algunos diseños digitales de máquinas de estados utilizando el lenguaje Verilog HDL.

### 4.2 Introducción teórica

#### 4.2.1 Máquinas de estados finitos en Quartus II

En las clases teóricas de la asignatura se explica, de modo general, como se realiza la descripción de una máquina de estados utilizando un HDL como Verilog HDL. Sin embargo, cada herramienta de síntesis HDL, suele tener sus propias peculiaridades que permiten identificar una descripción HDL como la descripción de una máquina de estados finitos. A continuación se mostrarán las particularidades que Altera recomienda para que Quartus II, reconozca que la descripción HDL corresponde con una máquina de estados.

Para asegurar un adecuado reconocimiento de la máquina de estados y obtener unos buenos resultados de implementación, Altera recomienda seguir las siguientes recomendaciones:

- Asignar un valor por defecto a todas las señales de salida que se generen en la máquina de estados, para evitar la inferencia de latches o realimentaciones combinacionales.
- Separa la lógica de la máquina de estados de cualquier función aritmética, *data paths* e incluso de la asignación de valor a las salidas.
- Si el diseño contiene una operación que se va a utilizar en más de un estado, habrá que definir la operación fuera de la máquina de estados y crear las salidas necesarias en la máquina para utilizar dicha operación.
- Utilizar un *reset*, asíncrono o síncrono, para definir un estado para el arranque o *power-up*.
- Representar los estados de la máquina de estados mediante datos del tipo parámetro (*parameter*). La implementación de estos parámetros hace que la máquina de estados sea fácil de leer y reduce el riesgo de errores durante la codificación.

- Altera recomienda utilizar directamente valores enteros en la variable de los estados, como `next_state <= 0`.
- No se infiere una máquina de estados en Quartus II, si para la transición del estado se utilizan operaciones aritméticas, por ejemplo:

```
case (state)
  0: begin
    if (ena) next_state <= state + 2;
    else next_state <= state + 1;
  end
  1: begin
    ...
  end
endcase
```

- No se infiere una máquina de estados en Quartus II, si la variable del estado es una salida.
- No se infiere una máquina de estados en Quartus II para variables con signo.

Si se tiene cualquier duda de cómo se implementa y que se reconozca una máquina de estados en Quartus II, siempre se puede recurrir a las plantillas (**Templates**). En las plantillas se pueden encontrar los esqueletos para la descripción de cuatro tipos de máquinas de estados:

- Máquinas de Moore.
- Máquinas de Mealy.
- Maquinas Seguras. Descripción de una máquina de estados que evita la existencia de estados espurios o no definidos.
- Maquinas con codificación de estados fijada por el diseñador.

A continuación se puede ver el aspecto de la plantilla de Quartus II de una máquina de estados de Moore de 4 estados:

```
// Quartus II Verilog HDL Template
// 4-State Moore state machine

// A Moore machine's outputs are dependent only on the
// current state.
// The output is written only when the state changes. (State
// transitions are synchronous.)

module four_state_moore_state_machine
(
  input clk, in, reset,
  output reg [1:0] out
);

  // Declare state register
  reg [1:0] state;

  // Declare states
  parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;

  // Output depends only on the state
  always @ (state) begin
```

```

        case (state)
            S0:
                out = 2'b01;
            S1:
                out = 2'b10;
            S2:
                out = 2'b11;
            S3:
                out = 2'b00;
            default:
                out = 2'b00;
        endcase
    end

    // Determine the next state
    always @ (posedge clk or posedge reset) begin
        if (reset)
            state <= S0;
        else
            case (state)
                S0:
                    state <= S1;
                S1:
                    if (in)
                        state <= S2;
                    else
                        state <= S1;
                S2:
                    if (in)
                        state <= S3;
                    else
                        state <= S1;
                S3:
                    if (in)
                        state <= S2;
                    else
                        state <= S3;
            endcase
        end
    end
endmodule

```

Figura 4.1. Plantilla de Quartus II de una máquina de Moore de 4 estados.

### 4.3 Estructura de la práctica

Las prácticas de este tema están estructuradas de la siguiente forma:

1. Diseño en Verilog HDL de una máquina de estados finitos para el control de un motor paso a paso.
2. Verificación del funcionamiento de la maquina en un entorno de verificación prediseñado.
3. Diseño en Verilog HDL de una máquina de estados finitos para el diseño del juego de luces del coche fantástico.

## 4.4 Desarrollo de la práctica

### 4.4.1 Tarea 2.1 – Diseño de un controlador para un motor paso a paso

Se desea realizar un controlador para un motor paso a paso, mediante descripción Verilog HDL. El controlador que se debe implementar, tendrá que emular el comportamiento del circuito integrado L297 de SGS THOMSON. Por tanto, el controlador consistirá en una máquina de ocho estados, en la que cada estado determinará la situación en la que se encuentra el bobinado del motor. La evolución de la máquina a través de los diferentes estados, provocará el giro del motor, cada cambio de estado se interpreta como un paso. El controlador tendrá cinco líneas de entrada, cuyo comportamiento deberá ser el siguiente:

- **CLK**, línea de reloj, activa por flanco de subida.
- **RESET**, asíncrono y activo a nivel bajo, posiciona el controlador en el estado inicial, también llamado "HOME".
- **ENABLE**, activa a nivel alto, habilita el funcionamiento del controlador; es decir, mientras esté activa el controlador debe hacer girar el motor (ir cambiando de estado).
- **UP\_DOWN**, esta línea sirve para indicar el sentido de giro del motor. Cuando está a nivel alto el motor gira en el sentido de las agujas del reloj, mientras que a nivel bajo lo hace en sentido contrario.
- **HALF\_FULL**, esta línea permite configurar el modo de funcionamiento. Cuando está a nivel alto el motor está en modo HALF, mientras que a nivel bajo el motor se configura en modo NORMAL o WAVE.

El controlador deberá tener así mismo seis líneas de salida, denominadas **A**, **B**, **C**, **D**, **INH1** e **INH2**, cuyo comportamiento dependerá del modo de funcionamiento tal y como se detalla en la Figura 4.2, Figura 4.3 y Figura 4.4.

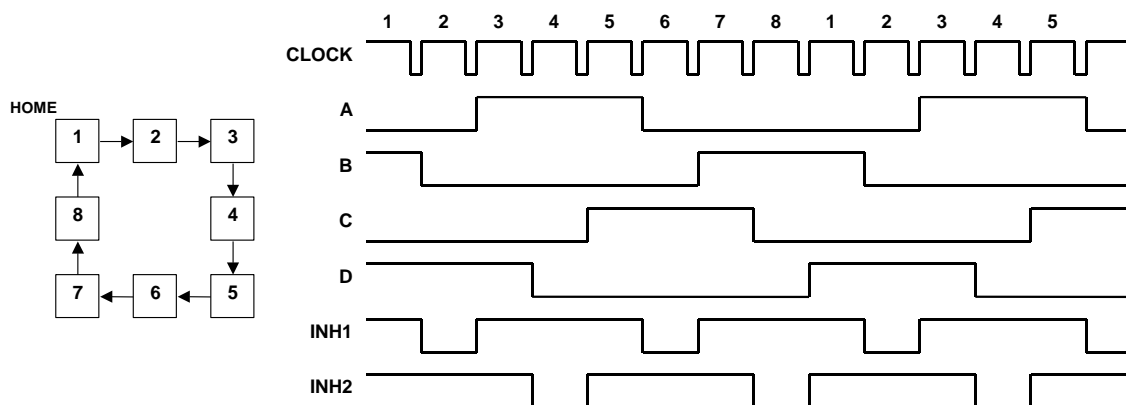


Figura 4.2. Modo de funcionamiento "HALF".

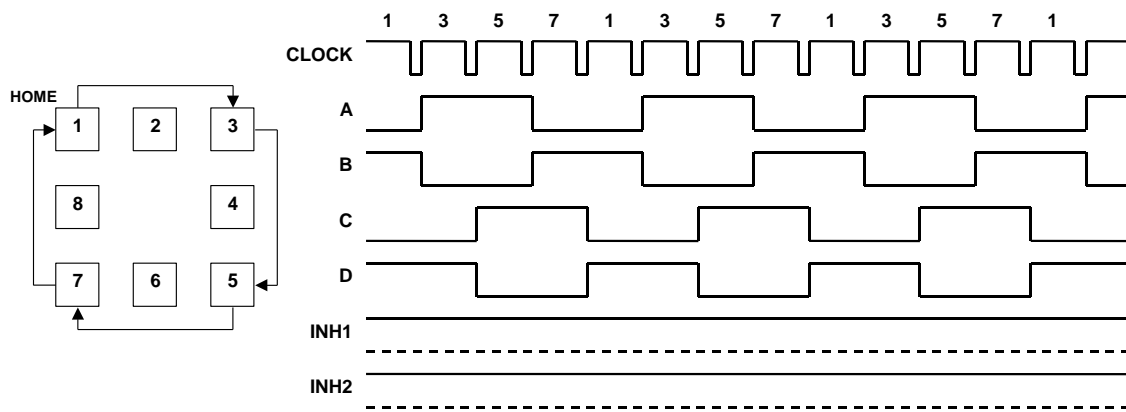


Figura 4.3. Modo de funcionamiento "NORMAL".

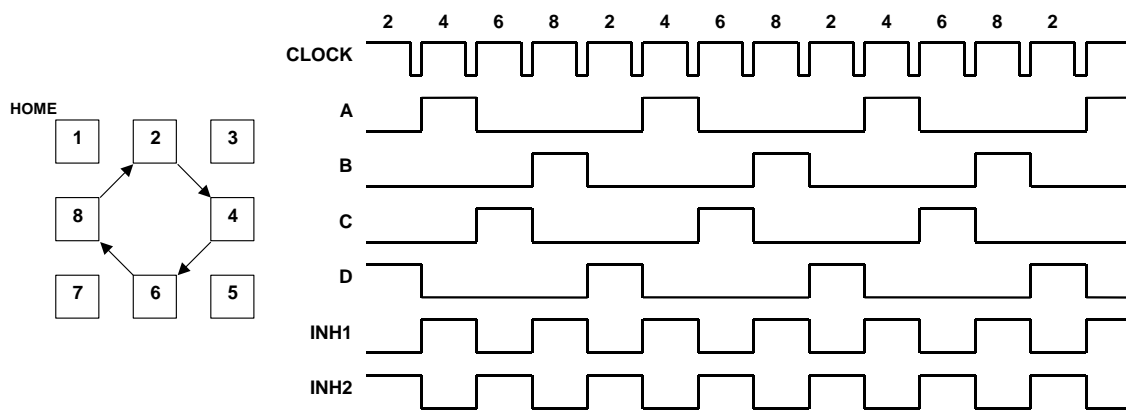


Figura 4.4. Modo de funcionamiento "WAVE".

Como se puede desprender de las figuras anteriores, en función de cómo evolucione la máquina de estados, se tendrán tres modos de funcionamiento. El modo "HALF" o de medio paso en el que la maquina pasa secuencialmente por todos los estados. El modo "NORMAL", en el que el controlador únicamente pasa por los estados impares. Y por último el modo "WAVE" en el que la evolución se realiza a través de los estados pares. En el modo de funcionamiento "HALF" el motor gira a medio paso, mientras que en los otros dos lo hace a paso entero. La selección del modo de funcionamiento se realiza por medio de la línea de entrada HALF\_FULL, si la línea está a nivel alto el modo de funcionamiento es "HALF", sin embargo, si la línea se pone a nivel bajo el modo de funcionamiento será el "NORMAL" o el "WAVE" en función del estado en el que se encuentre.

Diseñar una máquina de estados en Verilog HDL que emule el funcionamiento del controlador anteriormente descrito. El módulo diseñado debe denominarse `control_motor`, así mismo deben respetarse los nombres de los puertos de entrada y salida tal como se ha descrito anteriormente (Figura 4.5). Comprobar el correcto funcionamiento de la máquina de estados mediante la simulación de todas las posibles opciones.

```
module control_motor
(
    input CLK, RESET, UP_DOWN, HALF_FULL, ENABLE,
```

```

        output reg A, B, C, D, INH1, INH2
    );
    ...
endmodule

```

Figura 4.5. Estructura del módulo Verilog HDL del controlador de motor.

#### 4.4.2 Tarea 2.2 – Verificación del controlador para motor paso a paso

La verificación del funcionamiento del controlador para motor paso a paso, requiere de un motor paso a paso. Sin embargo, la placa DE2 no dispone de dicho motor ni de la electrónica necesaria para conectar uno. Entonces ¿Cómo verificar el funcionamiento del controlador diseñado? Mediante un simulador de motor paso a paso realizado por los profesores de la asignatura utilizando la placa DE2 y una pantalla táctil conectada a ella.

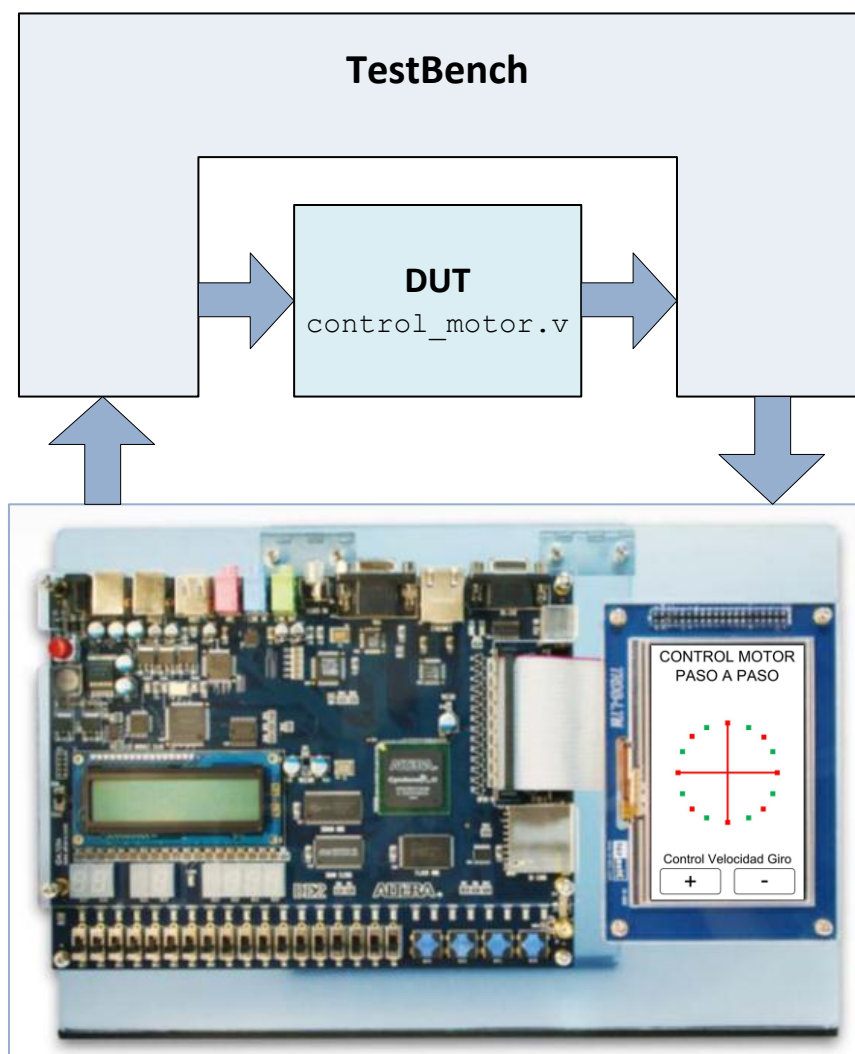


Figura 4.6. Estructura del simulador de motor paso a paso.

En la Figura 4.6 se puede ver la estructura del simulador de motor paso a paso. Está constituido por un diseño de verificación o *TestBench*, dentro del cual se englobará el



diseño bajo test (DUT), en este caso el controlador de motor paso a paso diseñado en la tarea anterior. El *TestBench* se encargará de generar las señales de entrada al DUT, recoger las señales de salida e interactuar con la placa DE2. En la pantalla táctil conectada a la placa DE2 se mostrarán puntos de colores alternativamente rojos y verdes que describen una circunferencia y que representan el giro del motor paso a paso. El simulador no sólo utiliza la pantalla, sino que también emplea otros elementos de la placa DE2, como los pulsadores, los interruptores, los LED y los displays 7-segmentos. A continuación se describe la funcionalidad de los distintos elementos de la placa DE2:

- Pulsador **KEY0**, es el **RESET** del sistema completo.
- Interruptor **SW0**, línea **ENABLE** del controlador para motor paso a paso.
- Interruptor **SW1**, línea **HALF\_FULL** del controlador para motor paso a paso
- Interruptor **SW2**, línea **UP\_DOWN** del controlador para motor paso a paso
- Los **LEDG0** a **LEDG5**, representan las señales **A, B, C, D, INH1** e **INH2** respectivamente del controlador para motor paso a paso.
- Los **LEDR0** a **LEDR15**, representan los 16 puntos o pasos en los que se ha dividido una vuelta completa del motor en la representación mostrada en la pantalla.
- El display **HEX0**, muestra el valor del estado de la máquina de estados que implementa el controlador para motor paso a paso. Es una cifra entre el 1 y el 8, que se obtiene de modo inverso a partir de los valores de las señales **A, B, C, D, INH1** e **INH2**, generados por el controlador teniendo en cuenta que se ajustan a los diagramas de la Figura 4.2, Figura 4.3 y Figura 4.4.
- El display **HEX1**, simplemente muestra la letra **E** de estado.
- El display **HEX3**, muestra la velocidad de giro del motor. Un valor entre el 0 y el 9, donde la velocidad se duplica al incrementar el valor de tal forma que el 0 representa una velocidad de giro de un paso (o medio) por segundo y el 9 de 1,95 ms por paso (o medio paso).
- La **Pantalla Táctil**:
  - En el **centro**, se representa por medio de puntos que siguen una trayectoria circular el giro del motor. En total se representan 16 puntos 8 rojos y 8 verdes alternados, con la finalidad de poder mostrar cuando funciona a medio paso el motor o paso entero. Cuando el motor va a medio paso se van viendo alternativamente puntos rojos y verdes. Cuando el motor funciona a paso entero únicamente se ven puntos rojos o verdes en función del modo “normal” o “wave” del controlador.
  - En la **parte inferior**, hay dos botones, representados por el **signo más** y el **signo menos**, que sirven para incrementar y decrementar respectivamente, la velocidad de giro del motor.

El procedimiento para verificar el funcionamiento del controlador para motor paso a paso utilizando el simulador realizado por los profesores de la asignatura será el siguiente:

1. Descargarse de **PoliformaT** el archivo `motor_alumno.qar`.
2. Picando dos veces sobre el archivo anterior, se desempaqueta el proyecto de Quartus II que contiene el simulador. En el proceso de desempaquetado,

- Quartus II preguntará en qué lugar se quiere desempaquetar y si se quiere abrir el proyecto.
- Una vez abierto el proyecto se puede comprobar que hay un módulo Top denominado `Verifica_motor`, que en su interior instancia dos bloques el `testbench` y el `control_motor`. El primero está oculto y el segundo no existe, ya que es el diseño realizado en la tarea anterior.
  - Copiar el archivo `control_motor.v` realizado en la tarea anterior en el subdirectorío de trabajo donde este el proyecto de Quartus II de verificación que se acaba de desempaquetar.
  - Compilar de proyecto. Si se produce algún error sintáctico debe ser debido a incongruencias con los nombres de los puertos utilizados en el módulo `control_motor` o del propio nombre del módulo. Realizar los cambios pertinentes en el módulo de `control_motor`, nunca en el archivo de `verifica_motor`.
  - Una vez concluida la compilación correctamente, hay que programar el dispositivo de la placa DE2.
  - Probar el funcionamiento en la placa jugando con los diferentes botones y computadores.

#### 4.4.3 Tarea 2.3 – Diseño de una máquina de estados para el juego de luces del coche fantástico

Ahora que se sabe cómo diseñar una máquina de estados finitos con Verilog HDL se puede volver a plantear el ejercicio realizado en la Tarea 1.4, con un enfoque basado en máquinas de estados finitos.

Diseñar una máquina de estados finitos que realice el juego de luces del coche fantástico al igual que se hizo en el diseño de la Tarea 1.4. Tal como se muestra en la Figura 4.7, sigue haciendo falta el contador, para poder mantener la visualización de los LEDs con una temporización de 0,25 segundos.

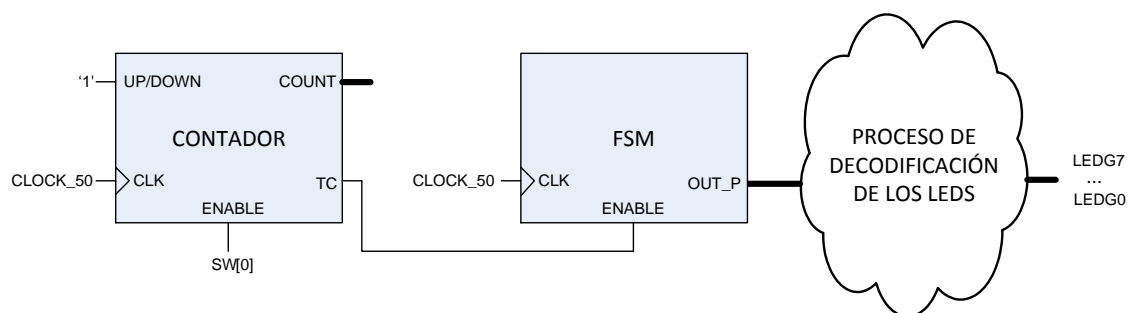


Figura 4.7. Diagrama de bloques de KIT, utilizando FSM.

Verifique su correcto funcionamiento en la placa DE2.

## 5 Diseño de un Controlador para Pantalla Táctil

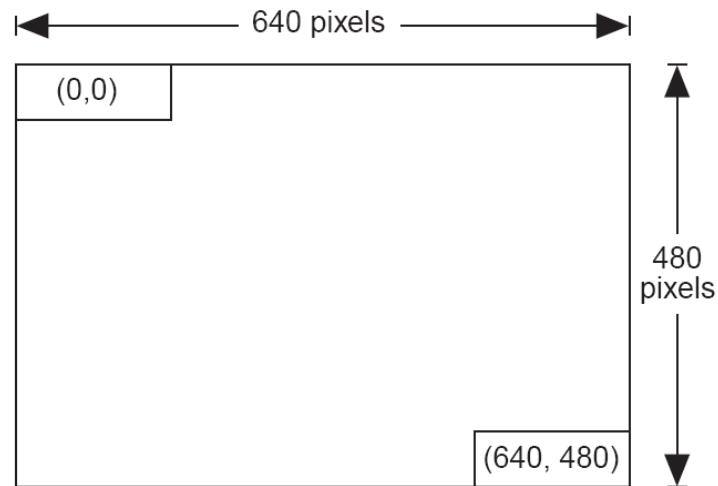
En este apartado se presenta un diseño puramente de control referido al manejo de una pantalla táctil. En la que se podrán distinguir dos partes bien diferenciadas, la visualización con un control similar a la gestión de una VGA y la captura del punto de toque que consiste en una comunicación serie. Conceptualmente el control de la visualización o control VGA, no se va a plantear como una máquina de estados sino como contadores interactuando entre sí y decodificando convenientemente sus salidas. Este tipo de diseños permitirá disponer, de un sistema de visualización muy potente de resultados de otros diseños. Igualmente el diseño de la captura del punto de toque es una comunicación serie y su subsiguiente conversión paralelo que es muy común en protocolos de comunicación, con lo cual, se dispondrá de las ideas fundamentales para transmisiones serie requeridas para algunos trabajos

### 5.1 Objetivos

- Aprender los conceptos básicos del lenguaje Verilog HDL y su uso en el diseño de control.
- Aprender el manejo de memorias ROM y mediante las macros disponibles en Quartus II.
- Aprender el método para probar el diseño en un hardware prediseñado, completando la utilización de la placa educativa DE2 de Altera.
- Aprendizaje y uso de una pantalla táctil.
- Analizar el funcionamiento del teclado.
- Practicar con el envío de las señales del teclado
- Desarrollar un módulo interfaz teclado-pantalla.
- Desarrollar un módulo interfaz de comunicación serie para convertidores analógico-digitales.
- Potenciar la iniciativa e imaginación en la realización de sistemas digitales.

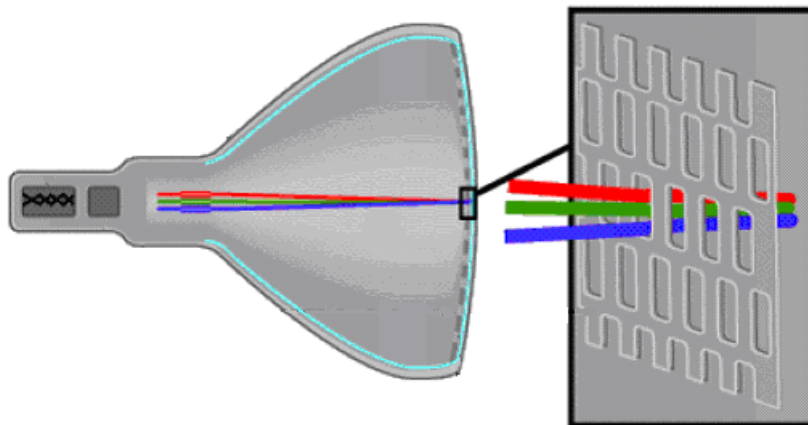
### 5.2 Introducción teórica

Para entender cómo es posible generar una imagen de video usando la placa DE2 de Altera, primero es necesario entender el funcionamiento de un monitor. Un monitor VGA estándar consiste en una cuadrícula de píxeles dividida en filas y columnas. Normalmente un monitor VGA contiene 480 filas, con 640 píxeles por fila, tal y como se muestra en la Figura 5.1.



**Figura 5.1. Estructura de un monitor.**

El color de cada píxel está determinado por el valor de las señales RGB (Red, Green, Blue). Dependiendo del valor de estas señales se obtiene un color diferente para el píxel (ver Figura 5.2).



**Figura 5.2. Configuración del color.**

Cada monitor VGA tiene un reloj interno que determina cuando cada píxel se actualiza (o refresca). Este reloj suele operar a 25 MHz, para una resolución de 640×480 píxeles.

El monitor refresca la pantalla de forma que es parcialmente controlada por las señales de sincronismo vertical y horizontal. El monitor empieza cada ciclo de refresco actualizando el píxel en la esquina superior izquierda de la pantalla, que puede tomarse como el origen de un plano X-Y, tal y como muestra la Figura 5.1, como punto (0,0). Después de que el primer píxel se refresca, el monitor refresca los siguientes píxeles de la fila. Cuando el monitor recibe un pulso en la sincronización horizontal, éste cambia de fila para refresca la siguiente fila de píxeles. Este proceso se repite hasta que el monitor llega a la parte inferior de la pantalla. Cuando el monitor alcanza la parte inferior de la pantalla, recibe un pulso a la sincronización vertical, causando que el monitor empiece a refrescar píxeles en la parte superior de la pantalla.

En un formato estándar de VGA, la pantalla contiene 640×480 píxeles. La señal de video debe redibujar toda la pantalla 60 veces por segundo para dar una imagen fija y reducir el *flicker*. Este periodo se llama el periodo de refresco. El ojo humano puede detectar *flicker* con ciclos de refresco de menos de 30 Hz.

### ¿Por qué la VGA trabaja a 25MHz?

El periodo de refresco de la pantalla es de 60 Hz, por lo que cada píxel se refrescará cada 40 ns; entonces es necesario un reloj de 25 MHz. En realidad es más ajustado 25,2 MHz ya que se tiene: 60 frames × 800 columnas × 525 filas = 25200000 píxeles

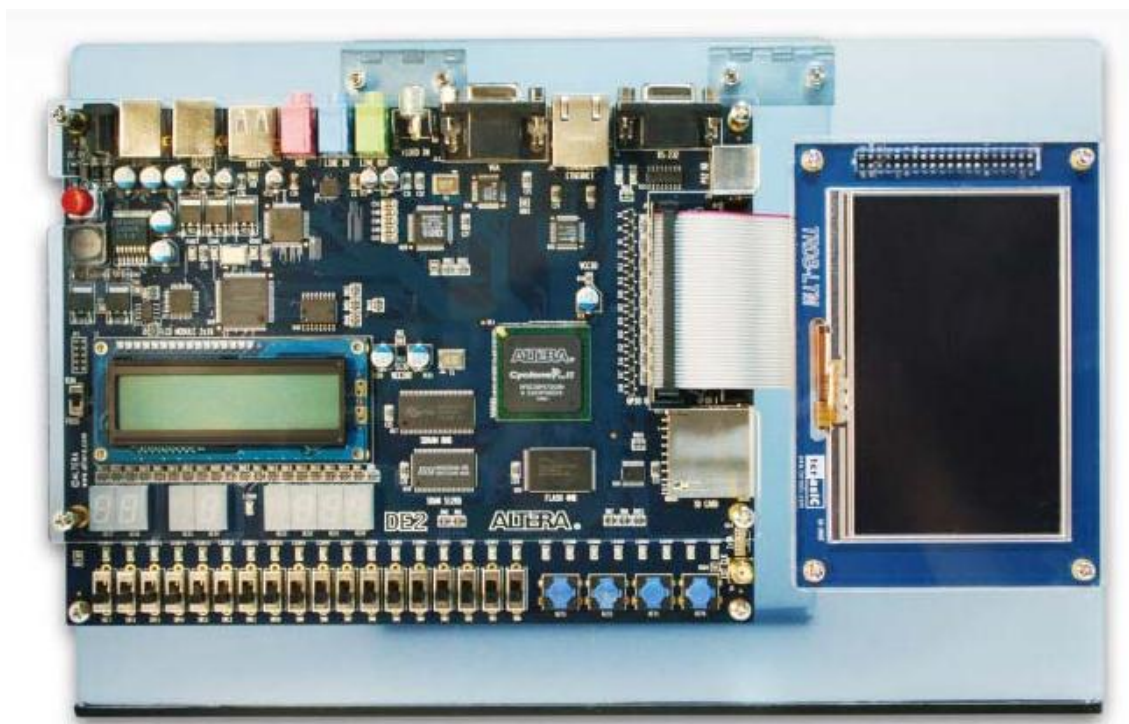
Hay que tener en cuenta que para que la pantalla VGA trabaje apropiadamente, debe recibir los datos en unos instantes determinados mediante pulsos adecuados. En el caso de los pulsos de sincronización vertical y horizontal deben producirse en unos instantes específicos para sincronizar el monitor mientras va recibiendo los datos de los colores.

### Resumen funcionamiento VGA

El monitor escribe en la pantalla enviando las señales del color RGB (rojo, verde, azul) y las señales de sincronización horizontal y vertical cuando la pantalla está en la localización esperada. Una vez que las señales de sincronización son exactas, el monitor necesita mantener la pista de la actual localización, por eso éste envía el color correcto para cada píxel.

## 5.3 Descripción del hardware específico

La empresa Terasic vende un módulo de pantalla táctil digital, adaptado para la conexión directa con las placas de Altera DE2 y DE1. Dicho módulo conocido también por las siglas LTM (**L**CD **T**ouch **P**anel **M**odule), se conecta a las placas mediante un cable paralelo de 40 pines del tipo IDE, utilizando principalmente el conector de expansión JP1 de la placa.



**Figura 5.3. Aspecto final de la conexión entre el módulo LTM y la placa DE2.**

El LTM está formado principalmente por tres componentes:

- Una pantalla LCD de color TD043MTEA1 de la marca Toppoly, la cual dispone de un interface paralelo para visualización de imágenes con soporte RGB de 24 bits, y otro interface serie de tres líneas para el control de la visualización y realización de ajustes, como el contraste, el brillo o la modulación gamma entre otros. En la Tabla 5.1 se pueden ver algunas especificaciones generales de la pantalla:

**Tabla 5.1. Características generales de la pantalla.**

Ítem	Descripción	Unidades
Tamaño de la Pantalla (Diagonal)	4,3	Pulgadas
Relación de Aspecto	15:9	–
Tipo de Pantalla	Retro iluminado (Backlight)	–
Área Activa (H x V)	93,6 x 56,16	mm
Número de Puntos (H x V)	800 x RGB x 480	Puntos
Tamaño del Punto (H x V)	0,039 x 0,117	mm
Distribución de los Colores	En Rayas (stripe)	–
Número de Colores	16 Millones	–

- Un convertidor Analógico Digital AD7843 de Analog Devices, que se encarga de convertir las coordenadas X e Y del punto donde se ha tocado la pantalla, en datos digitales.
- El conector de expansión de 40 pines.

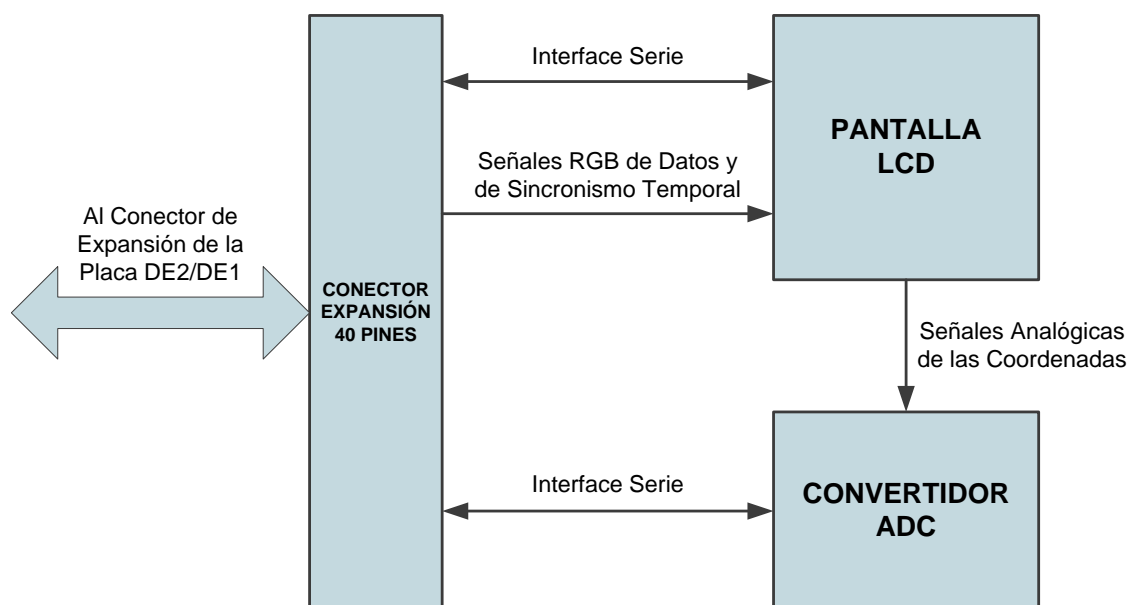


Figura 5.4. Diagrama de bloques del LTM.

ADC_PENIRQ_n	1	2	ADC_DOUT
ADC_BUSY	3	4	ADC_DIN
ADC_DCLK	5	5	B3
B2	7	8	B1
B0	9	10	NCLK
NC	11	12	GND
DEN	13	14	HD
VD	15	16	B4
B5	17	18	B6
B7	19	20	G0
G1	21	22	G2
G3	23	24	G4
G5	25	26	G6
G7	27	28	R0
VCC33	29	30	GND
R1	31	32	R2
R3	33	34	R4
R5	35	36	R6
R7	37	38	GREST
SCEN	39	40	SDA

Figura 5.5. Detalle del patillaje en el conector de expansión de 40 pines.



Tabla 5.2. Descripción de las señales en el conector de expansión de 40 pines.

Número de Pin	Nombre	Dirección	Descripción
1	ADC_PENIRQ_n	Output	Petición de interrupción del ADC
2	ADC_DOUT	Output	Datos de salida serie del ADC
3	ADC_BUSY	Output	Interface serie ADC ocupado
4	ADC_DIN	Input	Datos de entrada serie del ADC
5	ADC_DCLK	Input	Reloj del interface serie del ADC y LCD
6	B3	Input	Bit 3 azul bus de datos del LCD
7	B2	Input	Bit 2 azul bus de datos del LCD
8	B1	Input	Bit 1 azul bus de datos del LCD
9	B0	Input	Bit 0 azul bus de datos del LCD
10	NCLK	Input	Señal de reloj del LCD
11		N/A	
12		N/A	
13	DEN	Input	Señal de habilitación de los datos del LCD
14	HD	Input	Señal de sincronismo horizontal del LCD
15	VD	Input	Señal de sincronismo vertical del LCD
16	B4	Input	Bit 4 azul bus de datos del LCD
17	B5	Input	Bit 5 azul bus de datos del LCD
18	B6	Input	Bit 6 azul bus de datos del LCD
19	B7	Input	Bit 7 azul bus de datos del LCD
20	G0	Input	Bit 0 verde bus de datos del LCD
21	G1	Input	Bit 1 verde bus de datos del LCD
22	G2	Input	Bit 2 verde bus de datos del LCD
23	G3	Input	Bit 3 verde bus de datos del LCD
24	G4	Input	Bit 4 verde bus de datos del LCD
25	G5	Input	Bit 5 verde bus de datos del LCD
26	G6	Input	Bit 6 verde bus de datos del LCD
27	G7	Input	Bit 7 verde bus de datos del LCD
28	R0	Input	Bit 0 rojo bus de datos del LCD
29		N/A	
30		N/A	
31	R1	Input	Bit 1 rojo bus de datos del LCD
32	R2	Input	Bit 2 rojo bus de datos del LCD
33	R3	Input	Bit 3 rojo bus de datos del LCD
34	R4	Input	Bit 4 rojo bus de datos del LCD
35	R5	Input	Bit 5 rojo bus de datos del LCD
36	R6	Input	Bit 6 rojo bus de datos del LCD
37	R7	Input	Bit 7 rojo bus de datos del LCD
38	GREST	Input	Reset global, activo a nivel bajo
39	SCEN	Input	Señal de habilitación del interface serie del LCD a nivel alto y del ADC a nivel bajo
40	SDA	Input/Output	Datos del interface serie del LCD



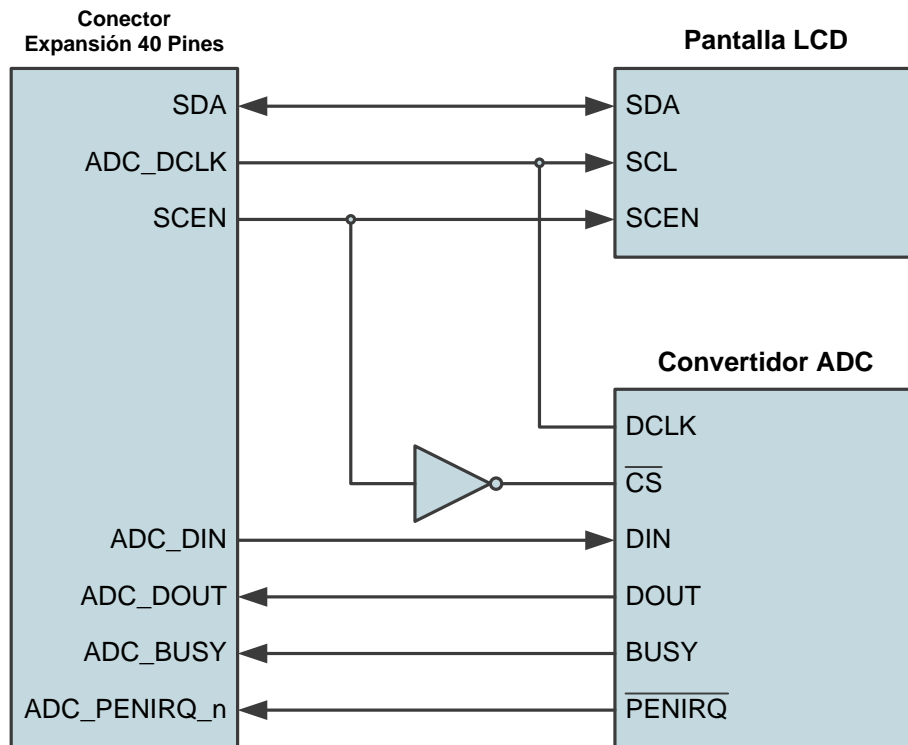
## 5.4 Utilización del Módulo LTM

En este apartado se muestra como se utiliza el módulo LTM, incluyendo el interface de control serie de la pantalla LCD y el del convertidor analógico-digital. También, se mostrarán los requerimientos temporales de las señales de sincronismo y de los datos RGB de la pantalla LCD, necesarios para la representación de imágenes en la misma.

### 5.4.1 El interface serie de la pantalla LCD

La pantalla LCD del módulo LTM dispone de un controlador de LCD que soporta tres resoluciones de visualización de la pantalla y diferentes funciones de ajuste de la imagen, además de disponer de un interface serie de comunicación, un gestor de temporización y un circuito de alimentación. Todas las funciones que tiene el controlador LCD se pueden gestionar accediendo a sus registros internos, por ejemplo desde una FPGA, mediante el interface serie de la pantalla LCD.

El módulo LTM dispone también de un convertidor analógico-digital (ADC), que permite obtener las coordenadas X e Y del punto donde se ha tocado la pantalla, por medio de otro interface serie que tiene el convertidor ADC. Tanto, el interface serie de la pantalla LCD como el del convertidor ADC se encuentran conectados a la FPGA de la placa DE2 por medio del conector de 40 pines que disponen tanto la placa DE2 como el módulo LTM. Debido a la limitación de pines que impone el conector, se han tenido que compartir algunas señales entre los interfaces serie de la pantalla LCD y del convertidor ADC. Ambos interfaces serie utilizan la misma señal de reloj (ADC\_DCLK) y la misma señal de habilitación (SCEN). Para evitar que la comunicación serie de un dispositivo pueda interferir con la del otro, la señal de habilitación se ha negado por medio de un inversor en el convertidor ADC, tal como se muestra en la Figura 5.6. Así cuando la pantalla LCD se encuentra habilitada (SCEN a nivel bajo) el convertidor ADC se encuentra deshabilitado y viceversa.



**Figura 5.6. Esquema de conexionado de los interfaces serie de la pantalla LCD y del convertidor ADC.**

El interface serie de la pantalla LCD, consiste en una comunicación serie síncrona en la que la FPGA actúa de máster de la comunicación generando las señales adecuadas y enviando los comandos de configuración que se requieren. Mientras se esté utilizando el interface serie es necesario que se mantenga la señal de reloj NCLK de la pantalla LCD. En la Figura 5.7 se muestra el diagrama temporal del funcionamiento del interface serie, mientras que en la Tabla 5.3 se pueden obtener los valores temporales. La pantalla LCD reconoce que comienza una nueva transmisión cuando se produce un flanco de bajada de la señal SCEN. A partir de ese momento en cada flanco de subida de la señal de reloj ADC\_DCLK se captura un bit de la línea de datos SDA.

Los primeros 6 bits (A5 ~ A0) indican la dirección del registro interno del controlador de la pantalla LCD. A continuación, hay un bit de Lectura/Escritura, un '0' indica que se va a escribir un dato en el registro y un '1' que el dato del registro indicado se leerá. El siguiente ciclo es un paso de ciclo y seguidamente viene los 8 bits de datos (D7 ~ D0). Los datos y las direcciones se envían secuencialmente desde el bit más significativo al menos significativo. El dato se escribirá en la dirección indicada cuando se detecte el final de la transferencia; es decir, después de los 16 ciclos de la señal de reloj ADC\_DCLK. Los datos no serán aceptados si hay más o menos de 16 flancos de subida, en una transacción.

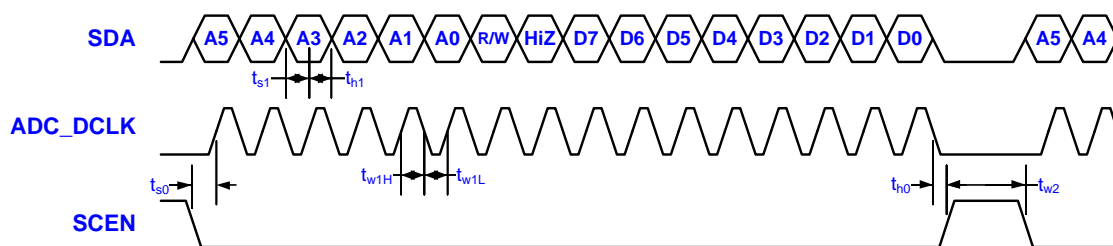


Figura 5.7. Formato de la transmisión y diagrama de temporización para el interface serie con la pantalla LCD.

Tabla 5.3. Parámetros temporales del interface serie con la pantalla LCD.

Ítem	Símbolo	Condiciones	Min.	Max.	Unidades
Tiempo Setup SDA	$t_{s0}$	SCEN a SCL	150		ns
	$t_{s1}$	SDA a SCL	150		ns
Tiempo Hold SDA	$t_{h0}$	SCEN a SCL	150		ns
	$t_{h1}$	SDA a SCL	150		ns
Ancho de Pulso	$t_{w1L}$	Ancho pulso SCL	160		ns
	$t_{w1H}$	Ancho pulso SCL	160		ns
	$t_{w2}$	Ancho pulso SCEN	1,0		ns
Ciclo de Trabajo			40	60	%

#### 5.4.2 Temporización de las señales para la visualización en la pantalla LCD

Para realizar la visualización de imágenes en la pantalla del LCD, hay que generar las señales de sincronismo y de datos adecuadas desde la FPGA al módulo LTM, para lo cual es necesario conocer las especificaciones temporales del mismo. Las señales de datos y sincronismo que necesita la pantalla LCD para visualizar imágenes son los estándares para monitores VGA. Los datos tienen formato RGB, utilizando 8 bits en paralelo para representar cada uno de los colores: Rojo (R[7..0]), Verde (G[7..0]) y Azul (B[7..0]), más una línea de habilitación de los datos (DEN). Para el sincronismo es necesaria una línea de reloj (NCLK), acompañada por las típicas señales de sincronismo horizontal (HD) y vertical (VD).

En la Figura 5.8 se muestran los requerimientos temporales básicos para la visualización de cada fila (horizontal) que se quiere mostrar en la pantalla LCD. Un pulso activo a nivel bajo de duración concreta ( $t_{hpw}$ ), se utiliza como señal de sincronismo horizontal (HD), para indicar el final de una línea y el comienzo de una nueva línea de datos. Después del pulso de sincronismo horizontal, las entradas de datos (RGB), todavía no son válidas durante un periodo de tiempo conocido como “back porch” ( $t_{hbp}$ ). Tras este tiempo, vendrá el tiempo de visualización de los datos en pantalla ( $t_{hd}$ ), en el que los datos se aplican a los píxeles, mediante un recorrido horizontal por la fila seleccionada. Durante el tiempo de visualización de los datos en la pantalla ( $t_{hd}$ ), la señal de habilitación de la pantalla LCD (DEN) debe permanecer a nivel alto. Finalmente hay un nuevo periodo temporal denominado “front porch” ( $t_{hfp}$ ), en el que las entradas de datos (RGB) vuelven a no ser válidas, y que tras concluir se produce un nuevo pulso de sincronismo horizontal.

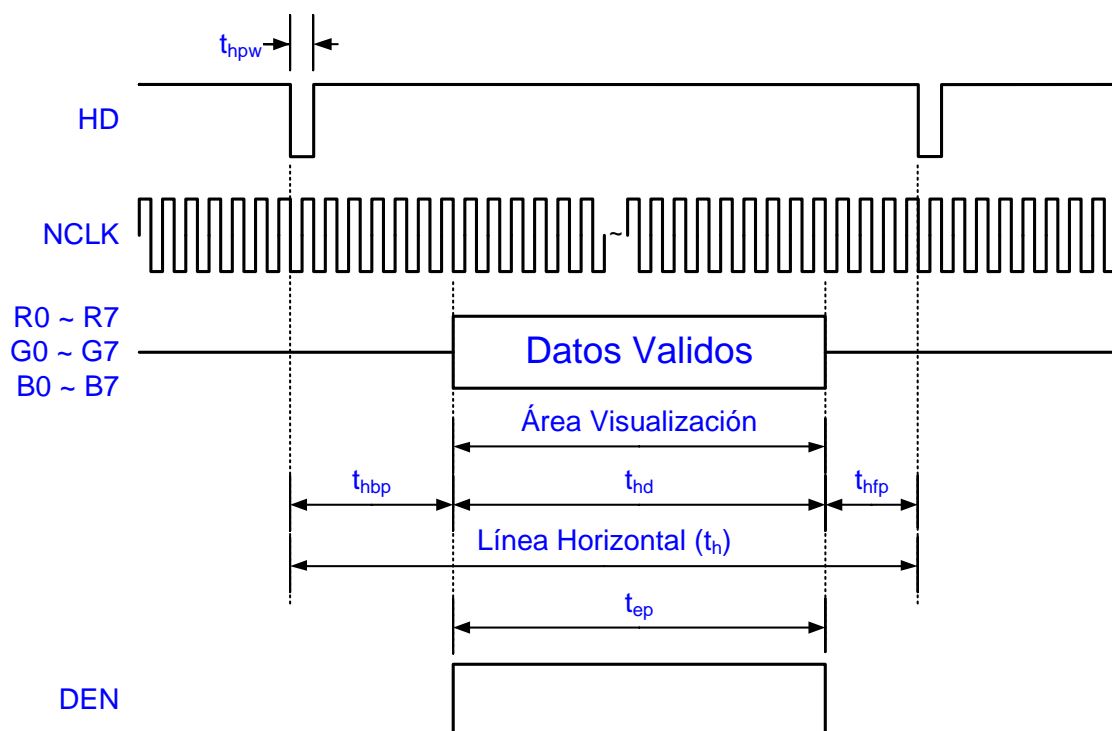


Figura 5.8. Diagrama de la temporización horizontal de la pantalla LCD.

La temporización del sincronismo vertical (VD), es similar a la anterior, tal como se muestra en la Figura 5.9, a excepción que un pulso de sincronismo vertical significa que ha terminado un “frame” o refresco completo de la pantalla y comienza el siguiente. Además, los datos se refieren al conjunto de todas las filas de un “frame”. La Tabla 5.4 y la Tabla 5.5 muestran los distintos valores temporales de la temporización horizontal y vertical para las diferentes resoluciones soportadas por la pantalla LCD. Finalmente la Tabla 5.6 muestra los valores temporales de las señales de sincronismo horizontal y vertical.

Tabla 5.4. Parámetros temporales horizontales de la pantalla LCD.

Parámetro		Símbolo	Resolución			Unidades
			800xRGBx480	480xRGBx272	400xRGBx240	
Frecuencia de reloj		NCLK	33,2	9	8,3	MHz
Datos Horizontales		t <sub>hd</sub>	800	480	400	NCLK
Total Línea Horizontal		t <sub>h</sub>	1056	525	528	NCLK
Ancho de Pulso HS	Min.	t <sub>hpw</sub>	1			NCLK
	Tip.		-			
	Max.		-			
Horizontal “Back Porch”		t <sub>hbp</sub>	216	43	108	NCLK
Horizontal “Front Porch”		t <sub>hfp</sub>	40	2	20	NCLK
Tiempo Habilitación DEN		t <sub>ep</sub>	800	480	400	NCLK

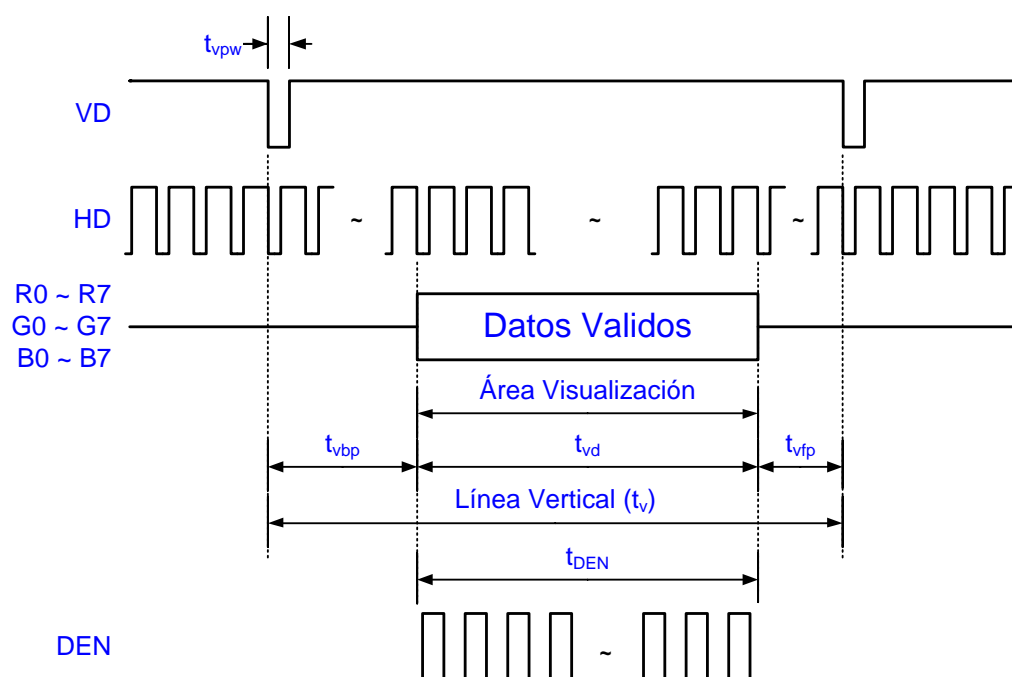


Figura 5.9. Diagrama de la temporización vertical de la pantalla LCD.

Tabla 5.5. Parámetros temporales verticales de la pantalla LCD.

Parámetro		Símbolo	Resolución			Unidades
			800xRGBx480	480xRGBx272	400xRGBx240	
Datos Verticales		t <sub>vd</sub>	480	272	240	H
Total Periodo Vertical		t <sub>v</sub>	525	286	262	H
Ancho de Pulso VS	Min.	t <sub>vpw</sub>	1			H
	Tip.		-			
	Max.		-			
Vertical “Back Porch”		t <sub>vbp</sub>	35	12	20	H
Vertical “Front Porch”		t <sub>vfp</sub>	10	2	2	H
Vertical “blanking”		t <sub>vb</sub>	45	14	22	H
Tiempo Habilidad DEN		t <sub>DEN</sub>	480	272	240	H

Tabla 5.6. Parámetros temporales de las señales de sincronización de la pantalla LCD.

Parámetro	Símbolo	Min.	Unidades
Periodo NCLK	$PW_{CLK*1}$	25	ns
Ancho de pulso a nivel alto NCLK	$PWH_{*1}$	10	ns
Ancho de pulso a nivel bajo NCLK	$PWL_{*1}$	10	ns

Tiempo de setup de los datos, HD, VD y DEN	$t_{ds}$	5	ns
Tiempo de hold de los datos, HD, VD y DEN	$t_{dh}$	5	ns

#### 5.4.3 El interface serie del convertidor ADC

El módulo LTM se encuentra equipado con el circuito integrado de Analog Devices AD7843, para el uso de pantallas táctiles. El AD7843 es un convertidor ADC de 12 bits, que proporciona la posición X e Y del punto de contacto que se ha tocado en la pantalla. La información de las coordenadas X e Y del punto de contacto, capturadas por el convertidor AD7843 se pueden obtener por medio del interface serie que dispone dicho convertidor.

Para obtener las coordenadas del convertidor ADC, en primer lugar hay que monitorizar la señal de interrupción ADC\_PENIRQ\_n. Esta señal permanece normalmente a nivel alto. Cuando se toca la pantalla por medio de un lápiz o del dedo, la señal ADC\_PENIRQ\_n pasa a nivel bajo. En consecuencia, la señal ADC\_PENIRQ\_n se puede utilizar para inicial o desencadenar, desde una FPGA, un proceso que genere la palabra de control necesaria para obtener del convertidor ADC las coordenadas X e Y del punto de la pantalla LCD que ha sido tocado.

La palabra de control que se debe enviar al convertidor ADC, por medio de la línea de datos ADC\_DIN de su interface serie, se muestra en la Tabla 5.7. En ella se proporciona la información sobre el comienzo de la conversión, el canal a direccionar, la resolución de la conversión, la configuración y el modo de alimentación del convertidor ADC.

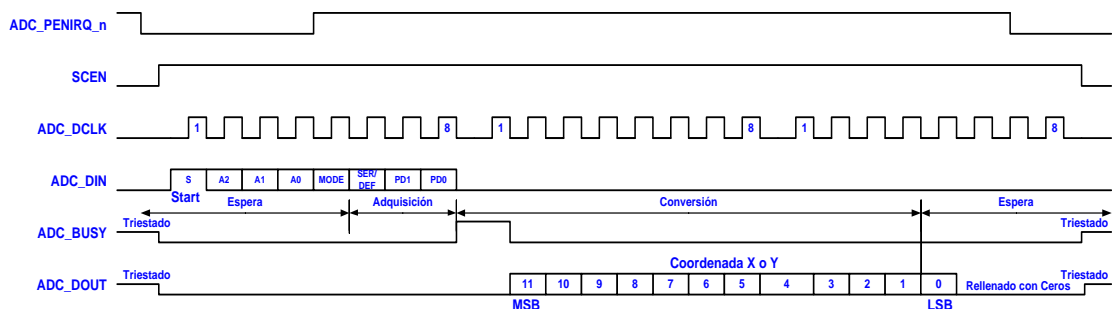
**Tabla 5.7. Descripción del funcionamiento de los bits del registro de configuración del AD7843.**

MSB							LSB
S	A2	A1	A0	MODE	SER/DEF	PD1	PD0

Bit	Nemotécnico	Comentario
7	S	Bit de Comienzo. La palabra de control comienza con un bit a nivel alto en la línea de datos ADC_DIN. Cada 15 ciclos de reloj (ADC_DCLK), puede comenzar una nueva palabra de control para el modo de funcionamiento de 12 bits de conversión, mientras que en el modo de conversión con 8 bits puede empezar cada 11 ciclos de reloj.
6-4	A2-A0	Bits de Selección del Canal. Estos tres bits de dirección, junto con el bit SER/DEF, controlan la selección de las señales de entrada que se van a utilizar para la conversión.
3	MODE	Control de resolución. Este bit permite seleccionar si la conversión analógico digital se realiza utilizando 12 bits (cero lógico) u 8 bits (uno lógico).
2	SER/DEF	Este bit permite seleccionar entre entrada simple o

		diferencial. Funciona en conjunción con las líneas de selección del canal A2–A0.
1, 0	PD1, PD0	Bits de control de alimentación. Estos dos bits permiten seleccionar el modo de consumo del convertidor AD7843.

En la Figura 5.10 se muestra una operación típica, por medio del interface serie del convertidor ADC. El reloj serie ADC\_DCLK es el encargado de gestionar todo el proceso de transferencia de información hacia y desde el convertidor ADC. Una conversión de datos completa dura 24 ciclos de reloj. Recordar que la línea de habilitación SCEN se encuentra compartida con el interface serie de control de la pantalla LCD.



**Figura 5.10. Diagrama de la temporización de la conversión en el interface serie del convertidor ADC.**

## 5.5 Estructura de la práctica

Las prácticas de este apartado están estructuradas de la siguiente forma:

1. Diseño de un sistema que genere señales de vídeo para el control de la visualización de la pantalla táctil.
2. Realización de ejercicios sobre el diseño realizado anteriormente para su comprobación.
  - a. Diseño para cambiar el color de la pantalla.
  - b. Diseño para que aparezcan en pantalla rayas de diferentes colores en la pantalla.
  - c. Diseño para que dada una posición determinada colocar una letra o figura.
3. Diseño de un sistema de comunicación del teclado con la pantalla.
4. Diseño de un sistema de comunicación serie con un ADC, para la gestión de la parte táctil de la pantalla.
5. Diseño libre de un sistema digital donde se fusionen la visualización en la pantalla y el uso de la función táctil.
6. Verificación de los módulos anteriores mediante la configuración de la FPGA Cyclone II de la placa DE2.

## 5.6 Desarrollo de la práctica

En este apartado se pretende plantear el diseño hardware en una FPGA de un bloque que permita utilizar el módulo LTM en la placa DE2 de Altera. Aunque inicialmente

puede parecer una labor ardua, con una adecuada estructuración de las tareas a realizar terminará convirtiéndose en un reto personal de resultados altamente satisfactorios, para los amantes de la electrónica digital.

Tal como se describe en la introducción del capítulo y se ve reflejado en la Figura 5.4, el módulo LTM está formado básicamente por dos bloques: La pantalla LCD y el convertidor ADC. La pantalla LCD se puede descomponer a su vez en dos elementos, el interface serie de la pantalla y las señales de datos y sincronismo. Por tanto, a la hora de realizar el diseño del bloque de control del módulo LTM, se planteará como la realización de tres bloques de diseño: El interface serie de la pantalla LCD, la generación de las señales de sincronismo y datos de la pantalla LCD y el interface serie del convertidor ADC.

#### 5.6.1 Tarea 3.1 – Generación de las señales de sincronismo y datos de la pantalla LCD

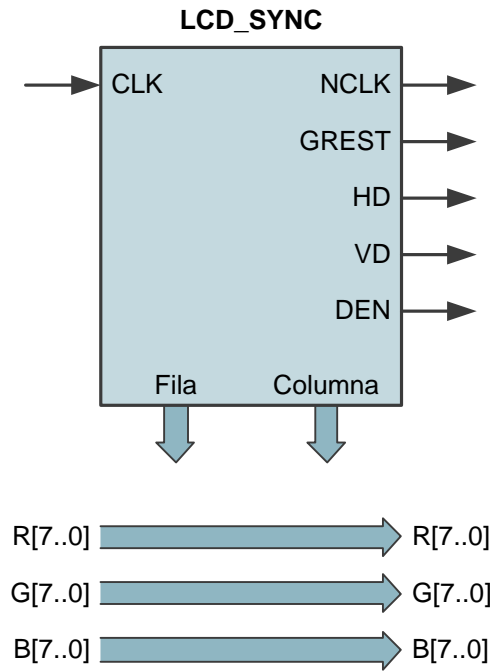
En primer lugar se aborda el diseño del bloque encargado de generar las señales de datos y sincronismo de la pantalla LCD. Este bloque permitirá la representación de imágenes en la pantalla. Es importante destacar que la configuración inicial de la pantalla permite la visualización de imágenes con una resolución de 800×480 pixeles; lo cual significa que no es necesario realizar el diseño del interface serie de la pantalla, para poder ver imágenes en la pantalla. El interface serie de la pantalla sólo hace falta en el caso de querer cambiar la resolución de la pantalla u otros parámetros de configuración o de la visualización de la misma.

Se quiere diseñar el módulo que genere las señales de sincronismo y datos de la pantalla LCD. Tal como se observa en la Figura 5.11, este módulo tendrá las siguientes señales de entrada y salida:

- Señales de entrada:
  - CLK, señal de reloj.
- Señales de salida:
  - HD y VD, señales de sincronismo horizontal y vertical respectivamente.
  - GREST, reset global de la pantalla.
  - NCLK, señal de reloj de salida hacia la pantalla.
  - DEN, señal de habilitación de la visualización en la pantalla.
  - Fila y Columna, salidas que indican la posición en la que se encuentra el refresco de la pantalla.

Para generar las señales de sincronismo, teniendo en cuenta que la resolución es 800×480 pixeles, se utilizarán contadores de forma que ajusten a lo representado en la Figura 5.8 y en la Tabla 5.4 para el sincronismo horizontal y en la Figura 5.9 y en la Tabla 5.5 para el sincronismo vertical. Por tanto para generar el sincronismo horizontal, simplemente hace falta un contador que cuente 800 ciclos de la señal de reloj NCLK. Mientras que para generar el sincronismo vertical, el contador debe contar 480 ciclos del sincronismo horizontal.





**Figura 5.11. Esquema de entradas y salidas del bloque de generación de datos y sincronismo de la pantalla LCD.**

Las señales de sincronismo y de color serán registradas a su salida ya que un pequeño retraso de unos pocos nanosegundos genera que las señales de color RGB provoquen una imagen borrosa.

Es importante tener en cuenta que la señal DEN de salida solo debe de estar activa en la región visible (800×480) de la pantalla. En caso contrario no se observará nada en la pantalla.

También hay que hacer notar la existencia de una señal denominada GREST, cuya función es realizar un Reset del módulo LTM, debiendo permanecer a nivel alto cuando se quiere hacer funcionar el módulo.

Es relevante destacar que aunque la señal de reloj NCLK, se establezca en 33,2 MHz en la Tabla 5.4, las especificaciones de la pantalla acepta frecuencias hasta 40 MHz, siendo perfectamente válida la opción de utilizar una señal de 25 MHz, al igual que la de una VGA, manteniendo en este caso los mismo valores de conteo que se especifican en la Tabla 5.4 y en la Tabla 5.5. En este caso lo más adecuado es que todo el diseño funcione con una señal de reloj de 25 MHz, en lugar de los 50 MHz del oscilador de la placa DE2.

**¿Cómo se pueden obtener los 25 MHz, teniendo en cuenta que el oscilador externo es de 50 MHz y hay que mantener los criterios del correcto diseño síncrono?** La respuesta es: mediante un PLL de los que disponen internamente las FPGAs. Este elemento permite generar un reloj, de distinta frecuencia y fase, a partir de otro. Los PLLs internos del dispositivo se pueden utilizar generándolos y caracterizándolos mediante la herramienta **MegaWizard Plug-In Manager** de Quartus II. El **MegaWizard Plug-In Manager** dispone de una función denominada **ALTPLL** que permite crea un nuevo módulo con los parámetros adecuados para utilizar las PLL

internas de la FPGA. Por medio de las diferentes ventanas de configuración de la función **ALTPLL** se establecen las características del PLL que se quiere utilizar. Dicha función genera los archivos necesarios para utilizar el PLL, entre ellos una plantilla para la instanciación del mismo en el diseño donde se vaya a utilizar. En la Figura 5.12 se muestra el código Verilog HDL, para instanciar y utilizar un PLL llamado `p11_25`, generado mediante la función **ALTPLL** del **MegaWizard Plug-In Manager** de Quartus II.

```
p11_25      p11_25_inst (
    .inclclk0 ( inclclk0_sig ),
    .c0 ( c0_sig )
);
```

Figura 5.12. Ejemplo de instanciación de un PLL generado con el MegaWizard Plug-In Manager.

Importante destacar que la pantalla se encuentra físicamente atornillada a la placa con una rotación de 90°, tal como se aprecia en la Figura 5.3. Esto significa que aunque la resolución sea de 800 pixeles en horizontal y 480 pixeles en vertical, hay que tener en cuenta que físicamente está girada esos 90°.

En la Tabla 5.8, se puede ver la asignación de pines que debe realizarse a las señales de la FPGA de la placa DE2, para utilizar el módulo LTM conectado al conector JP1.

Tabla 5.8. Descripción de las señales del módulo LTM en relación con la asignación de pines en la FPGA de la placa DE2.

Asignación FPGA	Nombre	Dirección	Descripción
D25	ADC_PENIRQ_n	Input	Petición de interrupción del ADC
J22	ADC_DOUT	Input	Datos de salida serie del ADC
E26	ADC_BUSY	Input	Interface serie ADC ocupado
E25	ADC_DIN	Output	Datos de entrada serie del ADC
F24	ADC_DCLK	Output	Reloj del interface serie del ADC y LCD
F23	B3	Output	Bit 3 azul bus de datos del LCD
J21	B2	Output	Bit 2 azul bus de datos del LCD
J20	B1	Output	Bit 1 azul bus de datos del LCD
F25	B0	Output	Bit 0 azul bus de datos del LCD
F26	NCLK	Output	Señal de reloj del LCD
N18	DEN	Output	Señal de habilitación de los datos del LCD
P18	HD	Output	Señal de sincronismo horizontal del LCD
G23	VD	Output	Señal de sincronismo vertical del LCD
G24	B4	Output	Bit 4 azul bus de datos del LCD
K22	B5	Output	Bit 5 azul bus de datos del LCD
G25	B6	Output	Bit 6 azul bus de datos del LCD
H23	B7	Output	Bit 7 azul bus de datos del LCD
H24	G0	Output	Bit 0 verde bus de datos del LCD
J23	G1	Output	Bit 1 verde bus de datos del LCD
J24	G2	Output	Bit 2 verde bus de datos del LCD
H25	G3	Output	Bit 3 verde bus de datos del LCD
H26	G4	Output	Bit 4 verde bus de datos del LCD

H19	G5	Output	Bit 5 verde bus de datos del LCD
K18	G6	Output	Bit 6 verde bus de datos del LCD
K19	G7	Output	Bit 7 verde bus de datos del LCD
K21	R0	Output	Bit 0 rojo bus de datos del LCD
K23	R1	Output	Bit 1 rojo bus de datos del LCD
K24	R2	Output	Bit 2 rojo bus de datos del LCD
L21	R3	Output	Bit 3 rojo bus de datos del LCD
L20	R4	Output	Bit 4 rojo bus de datos del LCD
J25	R5	Output	Bit 5 rojo bus de datos del LCD
J26	R6	Output	Bit 6 rojo bus de datos del LCD
L23	R7	Output	Bit 7 rojo bus de datos del LCD
L24	GREST	Output	Reset global, activo a nivel bajo
L25	SCEN	Output	Señal de habilitación del interface serie del LCD a nivel alto y del ADC a nivel bajo
L19	SDA	Input/Output	Datos del interface serie del LCD

### 5.6.2 Tarea 3.2 – Cambio de color de la pantalla

Tras diseñar el módulo anterior ahora se requiere verificarlo, de tal forma que en este apartado únicamente se requiere utilizar el módulo anterior y asignar un valor a las señales correspondientes de los colores (RGB), para que en la pantalla entera se vea de un color uniforme en función del valor de dichas señales.

### 5.6.3 Tarea 3.3 – Generación de barras de colores en la pantalla

En este apartado las imágenes que se quieren obtener en pantalla son barras verticales u horizontales de diferentes colores. Para ello habrá que ‘jugar’ con las diferentes señales que forman el módulo LCD\_SYNC.

### 5.6.4 Tarea 3.4 – Visualización de caracteres en pantalla

En este apartado se quiere poder visualizar en pantalla caracteres (por ejemplo: 0, A, B...). La pantalla está constantemente actualizándose (refrescándose), por ello es necesario el uso de una memoria para mantener los caracteres y que puedan ser mostrados en pantalla. Para ello se realizará el diseño de un módulo que vaya leyendo de memoria cada carácter y se lo vaya pasando al módulo LCD\_SYNC para que lo saque por pantalla.

La memoria a utilizar en el diseño será una memoria interna del dispositivo. Estas memorias se pueden crear mediante la herramienta **MegaWizard Plug-In Manager** de Quartus II, la cual dispone de un compilador de memorias que permite generar memorias de varios tipos como una ROM, que hace falta en este caso.

Esta memoria guardará los caracteres que se quieran visualizar en pantalla. Estos caracteres serán de dimensión 8x8. La ROM habrá que inicializarla con estos caracteres, para ello se utilizará un archivo tipo MIF.

El archivo MIF (Memory Initialization File) se utiliza para inicializar la memoria. El archivo puede definirse usando señales binarias, hexadecimales, decimales u octales. El siguiente ejemplo define una memoria de 512 celdas con anchura de 8 bits. Las direcciones son definidas en octal y los datos de las celdas en binario, para una mejor comprensión.

El patrón [00000000 .. 11111111] : 0; inicializa todas las celdas de la memoria a 0 antes de dar valores a unas determinadas direcciones. De esta forma no es necesario definir toda la memoria debido a que la primera asignación que se hace es poner a 0 todas las celdas.

```
Depth = 512;
Width = 8
Address_radix = oct
Data_radix=bin
% Instruction Pattern %
Content
Begin
    [00000000 .. 11111111] : 0;

%%  DATOS PARA LA MEMORIA%%

End;
```

Figura 5.13. Ejemplo de archivo MIF.

Cada letra, número o símbolo es una imagen de 8x8 píxeles de forma que su implementación se realizará de la siguiente manera teniendo en cuenta que la primera dirección de la memoria es 000001; por ejemplo, la implementación de la letra A se muestra en la Figura 5.14. Observar, que en el ejemplo, las direcciones que forman parte del carácter. Para el carácter A la dirección es 01 y de 0 a 7 se va recorriendo cada fila. Es por ello que para el 2º carácter ('B' p.e.) la dirección será 02. De esta forma direccionar el carácter se simplifica mucho.

%Dirección	Datos del carácter	Carácter visualizado%
010 :	00011000 ;	% ** %
011 :	00111100 ;	% ***** %
012 :	01100110 ;	% ** ** %
013 :	01111110 ;	% ***** %
014 :	01100110 ;	% ** ** %
015 :	01100110 ;	% ** ** %
016 :	01100110 ;	% ** ** %
017 :	00000000 ;	% %

Figura 5.14. Ejemplo de implementación de la letra A.

Tal y como se ha indicado anteriormente el módulo que se tiene que diseñar utilizará una memoria ROM de forma que irá leyendo cada carácter y se lo vaya pasando al módulo LCD\_SYNC para que lo saque por pantalla.

Para poder ir leyendo cada carácter de la memoria se utilizará contadores de columna y fila. El contador de columna se utiliza para seleccionar cada bit de izquierda a derecha en cada letra de la memoria al igual que la señal de video se mueve a lo largo de cada fila. El valor leído será 0 ó 1, indicando un color diferente para cada valor.

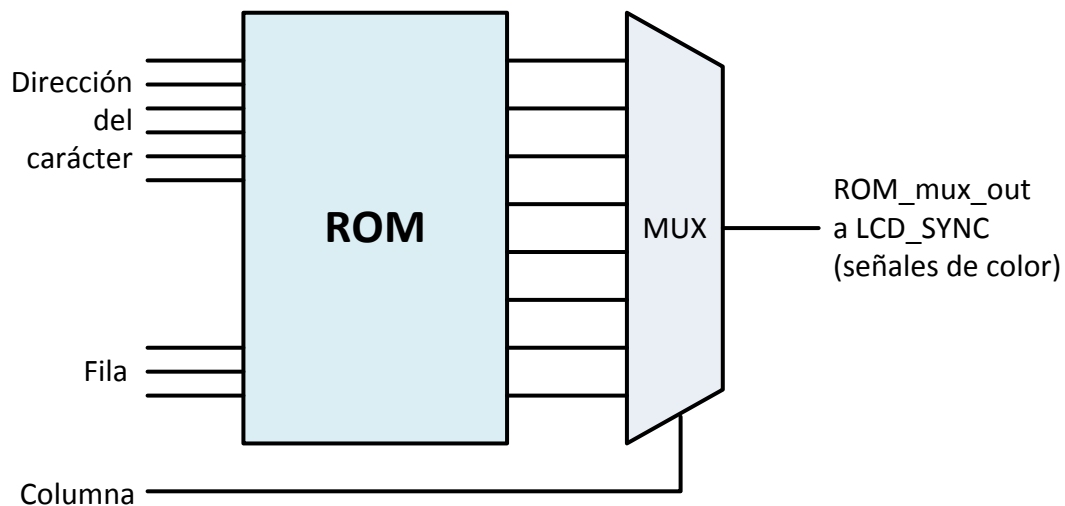
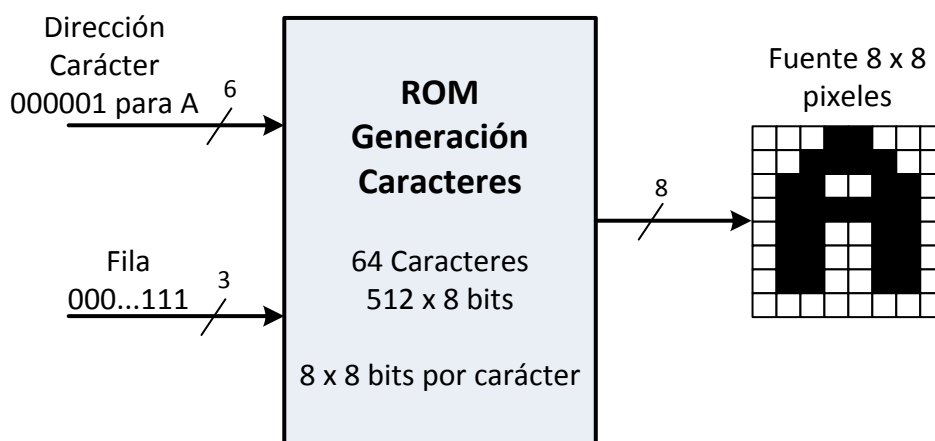


Figura 5.15. Esquema del diseño (CHAR\_ROM) a realizar.

La Figura 5.15 muestra el módulo (CHAR\_ROM) que hay que realizar. Como se observa tiene las siguientes señales externas:

- **Dirección del carácter**, entrada que nos indica la dirección del carácter que se quiere visualizar en pantalla. Como se ha explicado anteriormente será 01, en el caso del carácter A.
- **Fila**, entrada que indica la fila que hay que leer en memoria, es decir, es el contador de fila. Este contador no habrá que implementarlo si no que se utilizará el dado por el módulo LCD\_SYNC que ya recorre las filas.
- **Columna**, entrada que indica la columna que hay que leer en memoria, es decir, es el contador columna, que también se obtendrá de la salida del módulo LCD\_SYNC. En nuestro caso se necesitan para los 8 datos a multiplexar, tres señales del contador de columnas.
- **ROM\_mux\_output**, salida que va dando los datos de cada columna para pasárselo a la pantalla. Por ello se conectará a las entradas R, G, B del LCD\_SYNC. Se ha dibujado una sola salida porque seguramente se manipulará únicamente una de las tres señales RGB para dejar las otras dos restantes fijas.

En dicho esquema, el módulo ROM deberá ser implementado mediante una memoria interna ROM (como se había indicado anteriormente) con un fichero de inicialización adecuado para representar hasta 64 caracteres diferentes (ver Figura 5.16). Dicho fichero de inicialización lo se encuentra disponible en **PoliformaT**.



**Figura 5.16. Organización de la memoria ROM.**

Como se ha elegido un tamaño de carácter de 8x8 y la resolución de la pantalla es de 800x480 píxeles, el número máximo de caracteres que puede soportar el módulo será 100x16.

Se quiere que toda la pantalla se llene de caracteres diferentes o iguales. Se quiere obtener este tipo de representación porque es mucho más fácil de implementar y resultará más fácil de comprobar el módulo diseñado ya que no se tiene en cuenta la posición en pantalla sino que sólo se introduce información por las señales R, G, B.

#### 5.6.5 Tarea 3.5 – Generación de una figura simple en un punto determinado de la pantalla

En este apartado se quiere que el carácter creado en el apartado anterior y que se ha podido visualizar aparezca en pantalla una única vez y en una posición determinada lo más centrada posible en pantalla.

#### 5.6.6 Tarea 3.6 – Generación de una frase que aparezca en pantalla

En este apartado se quiere visualizar los nombres de los componentes del grupo de prácticas en pantalla (en una línea bastante centrada). Como idea se debe partir del esquema observado en la Figura 5.15 con la diferencia de que la dirección de carácter no es una información fija sino que va a ser unas direcciones cambiantes según el posicionado en pantalla (controlado de nuevo por los contadores de filas y columnas).

Una sugerencia importante sería almacenar en otra memoria ROM la frase a visualizar (los contenidos de esa memoria serían las direcciones a seleccionar en la CHAR\_ROM) y se direccionaría esa nueva memoria mediante los contadores de filas y columnas que se tiene accesibles. El correcto direccionamiento y la correcta generación del archivo tipo MIF de inicialización marcarán el éxito o no que se pueda tener en este ejercicio.

#### 5.6.7 Tarea 3.7 – Escribir desde el teclado y visualizar en pantalla

Tras realizar las prácticas de la pantalla donde se manejaba ésta y se podían visualizar caracteres en una posición determinada, se quiere implementar un módulo que se utilice para visualizar en la pantalla las teclas pulsadas en un teclado PS/2 conectado a la placa DE2, tal y como se muestra la Figura 5.17.

Para que no haya que realizar el diseño de control del teclado, en **PoliformaT** se encuentra disponible una versión de dicho bloque, realizada por los profesores de la asignatura. El módulo de control de teclado facilitado proporciona el valor de la tecla pulsada en cada momento. Dicho dato es lo que se conoce como el “scan code” de la tecla, el cual es único y universal para todos los teclados.

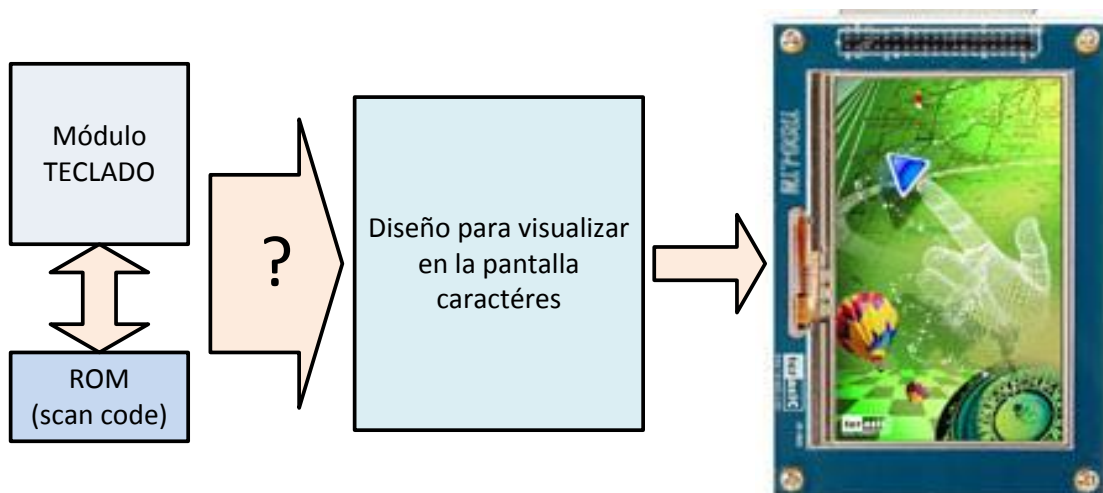


Figura 5.17. Idea del diseño a realizar.

Diseñar un módulo que visualice por pantalla en una posición dada cada una de las letras pulsadas en el teclado. Tal y como se muestra en la Figura 5.17 el diseño a realizar será la interfaz entre el módulo de teclado y el módulo utilizado para visualizar caracteres. Por ello habrá que realizar una transformación del “scan code” a la ROM de los caracteres. En la Figura 5.18 se muestra un esquema de cómo se puede hacer dicha transformación.

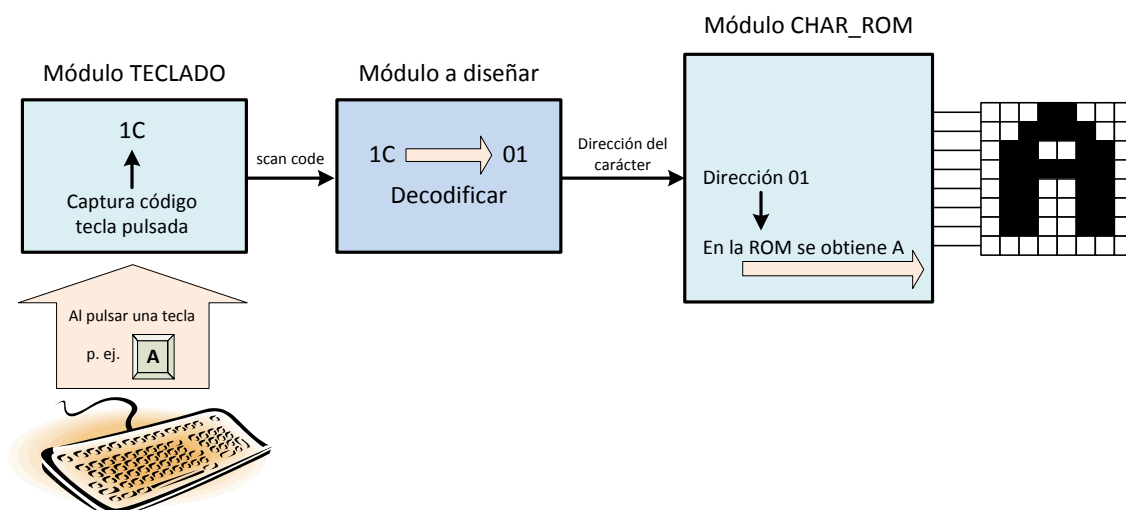


Figura 5.18. Esquema de la solución a realizar.

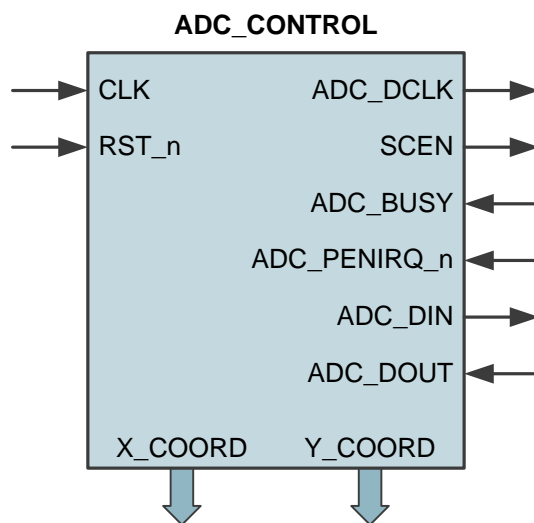
Por ejemplo: si se pulsa la tecla ‘A’ del teclado, el código que se transmite por la línea o “scan code” es ‘1C’. Cuando se reciba este código el módulo a implementar debe actuar de decodificador y pasar este código al código correspondiente a la letra A que

está contenido en la archivo MIF del módulo CHAR\_ROM, es decir se traducirá por la dirección 01, para recordarlo ver la Figura 5.14.

#### 5.6.8 Tarea 4.1 – Interface serie del convertidor ADC

El interface serie del convertidor ADC, permite obtener las coordenadas X e Y del punto de la pantalla que ha sido tocado por un dedo o por un lápiz. Para ello, hay que realizar un bloque que establezca una comunicación serie con el convertidor AD7843 de Analog Devices, ajustándose a los requerimientos temporales y de funcionamiento descritos en la Figura 5.10. El bloque de interface serie con el convertidor ADC, es un diseño independiente del de la pantalla, con la salvedad que por restricciones de conexión se comparten dos señales, tal como se muestra en la Figura 5.6: la señal de reloj ADC\_DCLK y la de selección SCEN.

En este apartado se sentarán las bases para realizar el diseño del interface serie del convertidor ADC, considerando su total independencia; es decir, que hay que generar todas las señales necesarias para la comunicación. En la Figura 5.19, se muestra el diagrama del bloque objeto de diseño con las entradas y salidas que debe tener. Además de las señales propias del interface serie para controlar el convertidor ADC, se debe obtener como salida el valor de las coordenadas X e Y del punto de contacto, que en definitiva es la finalidad del diseño.



**Figura 5.19. Diagrama de entradas y salidas del interface serie del convertidor ADC.**

Tal como se ha descrito anteriormente, cuando un dedo toca la pantalla del módulo LTM, el convertidor ADC pasa a nivel bajo la señal ADC\_PENIRQ\_n, momento que se debe de utilizar para inicial la comunicación con el convertidor ADC, con la finalidad de obtener el valor de las coordenadas del punto que se ha tocado. Para ello, en primer lugar, hay que enviar al convertidor ADC, por la línea ADC\_DIN un byte de control cuyo contenido se describe en la Tabla 5.7. Para obtener el valor concreto del byte, se establecerán las siguientes consideraciones: conversión de 12 bits, modo de entrada diferencial y con alimentación activada y ADC\_PENIRQ\_n habilitada. A partir de estos parámetros se establecen los valores que debe tener el byte de control, según se quiera leer la coordenada X o la Y, tal como son mostrados en la Tabla 5.9.



Tabla 5.9. Valores del byte de control en función de la coordenada.

	MSB						LSB	
	S	A2	A1	A0	MODE	SER/DEF	PD1	PD0
X	1	0	0	1	0	0	1	0
Y	1	1	0	1	0	0	1	0

El hecho que haya palabras de control distintas para obtener la coordenada X y la Y, significa que hay que realizar dos operaciones de lectura del convertidor para conseguir el valor exacto del punto de contacto. Por fortuna, el proceso de lectura en ambos casos es idéntico al mostrado en la Figura 5.10, comienza con la puesta a nivel bajo de la señal ADC\_PENIRQ\_n, tras lo cual se envía la palabra de control para la lectura de la coordenada X o la Y, una vez recibido el dato, tras los 24 ciclos de la señal ADC\_DCLK, concluye la comunicación. Esto significa que tras obtener en primer lugar la coordenada X, la señal ADC\_PENIRQ\_n vuelve a nivel bajo iniciando un nuevo proceso que permite la lectura de la coordenada Y. Todo esto es posible, porque el dedo permanece mucho tiempo tocando la pantalla, en comparación con el tiempo de comunicación, provocando así que se puedan generar varios flancos descendentes de la señal ADC\_PENIRQ\_n, durante el tiempo de contacto.

Los datos de catálogo del convertidor AD7843, establecen que la frecuencia de la señal de reloj ADC\_DCLK, que fija la velocidad de comunicación, puede establecerse en un valor mínimo de 10 kHz hasta un valor máximo de 2 MHz.

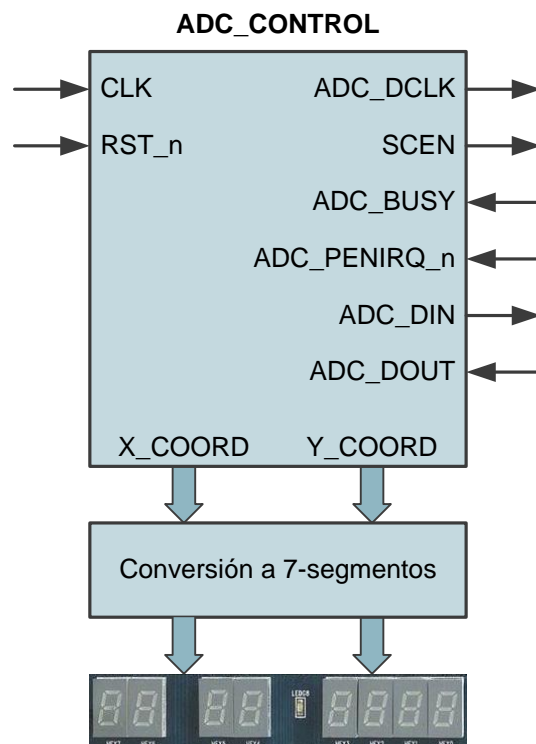


Figura 5.20. Diagrama del ejercicio de visualización del punto de “toque”.

### 5.6.9 Tarea 4.2 – Visualización del punto de contacto de la pantalla

Una vez se realizado el bloque de control del interface serie del convertidor ADC, un buen ejercicio es mostrar el valor del punto de contacto en los visualizadores 7-segmentos de la placa DE2, tal como se sugiere en la Figura 5.20. Dado que la conversión es de 12 bits, se necesitan tres visualizadores para cada coordenada. Representando en hexadecimal se obtendrán los valores que se muestran en la Figura 5.21 para cada una de las esquinas del módulo LTM. En la Tabla 5.8, se puede ver la asignación de pines que debe realizarse a las señales de la FPGA de la placa DE2, para utilizar el módulo LTM conectado al conector JP1.

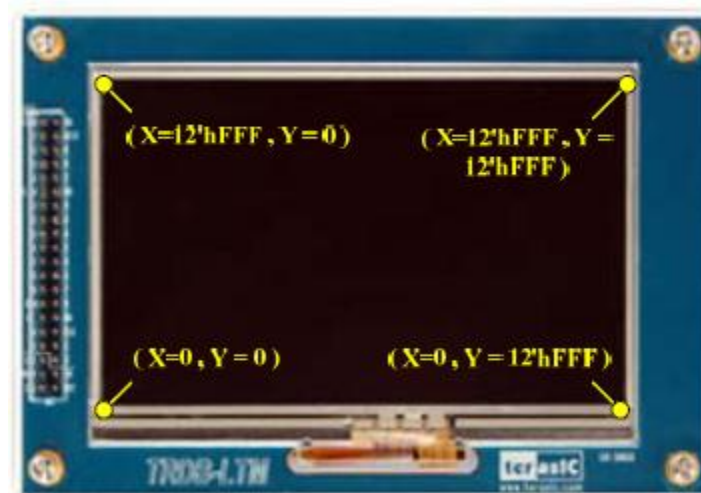


Figura 5.21. Valor de las coordenadas X e Y en las esquinas del módulo LTM.

### 5.6.10 Tarea 5 – Diseño libre

Un buen final de las prácticas de la asignatura es realizar un diseño de un sistema digital donde se demuestre todo lo aprendido a lo largo de las diferentes sesiones de prácticas. Como última tarea se propone realizar un sistema que donde al menos se fusionen la visualización en la pantalla y el uso de la función táctil.

### 5.6.11 Tarea 6 – Interface serie de la pantalla LCD

El interface serie de la pantalla LCD, permite ajustar diferentes aspectos de la visualización en el módulo LTM, entre ellos la elección de la resolución de la pantalla. Inicialmente cuando se conecta la alimentación, el módulo LTM funciona con unos parámetros de visualización por defecto, como por ejemplo una resolución de 800x480 pixeles, que nos ha permitido trabajar hasta este momento. Sin embargo, si se desea cambiar, entre otras cosas, la resolución de la pantalla hay que realizar el diseño de un bloque que permita comunicarse con la pantalla LCD por medio de su interface serie.

El interface serie de la pantalla LCD, consiste en una comunicación serie síncrona, la cual ha sido descrita anteriormente y que se muestra esquemáticamente en el diagrama temporal de la Figura 5.7. A la hora de realizar el diseño del interface serie, hay que recordar algunos aspectos importantes: en primer lugar el interface serie de la pantalla LCD comparte dos señales con el interface serie del convertidor ADC, la señal de reloj (ADC\_DCLK) y la señal de habilitación (SCEN), en segundo lugar la línea de datos (SDA) es bidireccional y por último mientras se esté utilizando el interface serie

es necesario que se mantenga la señal de reloj NCLK de la visualización de la pantalla LCD.

El interface serie permite acceder a un total de 35 registros internos de configuración de la pantalla LCD. En la Tabla 5.10 se describe cada uno de esos registros.

**Tabla 5.10. Registros internos de configuración de la pantalla LCD.**

Dirección	Valor Defecto	Lectura Escritura	Descripción
0x00	0x00	R/W	[7:0]: Registro de Test
0x01	0xC1	R	[7:4]: Identificación Chip, [3:0]: Versión Chip
0x02	0x07	R/W	[7:6]: Selección del método de inversión del punto. [5]: Polaridad pulso sincronismo vertical. Por defecto, activo a nivel bajo. [4]: Polaridad pulso sincronismo horizontal. Por defecto, activo a nivel bajo. [3]: Selección del flanco activo del reloj NCLK. Por defecto, flanco de bajada. [2:0]: Selección de la resolución. Ver descripción en la Tabla 5.11.
0x03	0x5F	R/W	[7]: Selección de la resolución y “standby” por hardware o software. [6]: On/off precarga. [5:4]: Ajuste de la capacidad de salida. [3]: On/off salida PWM. [2]: On/off salida VGL. [1]: On/off salida CP_CLK. [0]: Modo de funcionamiento (normal o espera).
0x04	0x17	R/W	[5:4]: Ajuste frecuencia VGL. [3:2]: Ajuste frecuencia CP_CLK. [1]: Modo vertical invertido. [0]: Modo horizontal invertido.
0x05	0x20	R/W	[5:0]: Desplazamiento horizontal del display para el modo SYNC. Por defecto, centrado.
0x06	0x08	R/W	[3:0]: Desplazamiento vertical del display para el modo SYNC. Por defecto, centrado.
0x07	0x20	R/W	[5:0]: Ajuste de la anchura de pulso a nivel alto de CKH.
0x08	0x20	R/W	[5:0]: Ajuste de la anchura de no coincidencia de CKH.
0x09	0x20	R/W	[5:0]: Ajuste de la anchura de no coincidencia entre el flanco de subida de ENB y de CKH.
0x0A	0x20	R/W	[5:0]: Ajuste de la anchura de pulso a nivel bajo de ENB.
0x0B	0x20	R/W	[5:0]: Ajuste de la ganancia del contraste del color rojo. Ajustable entre 0 y 1,96875 (0x3F), por defecto a 1 (0x20).
0x0C	0x20	R/W	[5:0]: Ajuste de la ganancia del contraste del color verde. Ajustable entre 0 y 1,96875 (0x3F), por defecto a 1 (0x20).
0x0D	0x20	R/W	[5:0]: Ajuste de la ganancia del contraste del color azul. Ajustable entre 0 y 1,96875 (0x3F), por

			defecto a 1 (0x20).
0x0E	0x10	R/W	[5:0]: Ajuste del nivel de brillo en el color rojo. Ajustable entre -16 (0x00) y 47 (0x3F), por defecto a 0 (0x20).
0x0F	0x10	R/W	[5:0]: Ajuste del nivel de brillo en el color verde. Ajustable entre -16 (0x00) y 47 (0x3F), por defecto a 0 (0x20).
0x10	0x10	R/W	[5:0]: Ajuste del nivel de brillo en el color azul. Ajustable entre -16 (0x00) y 47 (0x3F), por defecto a 0 (0x20).
0x11	0x00	R/W	Corrección Gamma (ver diagrama en Figura 5.22): [7:6]: Bits [9:8] del segmento GAMMA 0. [5:4]: Bits [9:8] del segmento GAMMA 8. [3:2]: Bits [9:8] del segmento GAMMA 16. [1:0]: Bits [9:8] del segmento GAMMA 32.
0x12	0x5B	R/W	Corrección Gamma (ver diagrama en Figura 5.22): [7:6]: Bits [9:8] del segmento GAMMA 64. [5:4]: Bits [9:8] del segmento GAMMA 96. [3:2]: Bits [9:8] del segmento GAMMA 128. [1:0]: Bits [9:8] del segmento GAMMA 192.
0x13	0xFF	R/W	Corrección Gamma (ver diagrama en Figura 5.22): [7:6]: Bits [9:8] del segmento GAMMA 224. [5:4]: Bits [9:8] del segmento GAMMA 240. [3:2]: Bits [9:8] del segmento GAMMA 248. [1:0]: Bits [9:8] del segmento GAMMA 256.
0x14	0x00	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 0. Ver diagrama en Figura 5.22.
0x15	0x20	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 8. Ver diagrama en Figura 5.22.
0x16	0x40	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 16. Ver diagrama en Figura 5.22.
0x17	0x80	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 32. Ver diagrama en Figura 5.22.
0x18	0x00	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 64. Ver diagrama en Figura 5.22.
0x19	0x80	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 96. Ver diagrama en Figura 5.22.
0x1A	0x00	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 128. Ver diagrama en Figura 5.22.
0x1B	0x00	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 192. Ver diagrama en Figura 5.22.
0x1C	0x80	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 224. Ver diagrama en Figura 5.22.
0x1D	0xC0	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 240. Ver diagrama en Figura 5.22.
0x1E	0xE0	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 248. Ver diagrama en Figura 5.22.
0x1F	0xFF	R/W	[7:0]: Corrección Gamma, bits [7:0] del segmento GAMMA 256. Ver diagrama en Figura 5.22.
0x20	0xD2	R/W	[7:4]: Define la tensión positiva de referencia del

			DAC para el código FFH. [3:0]: Define la tensión positiva de referencia del DAC para el código 00H.
0x21	0xD2	R/W	[7:4]: Define la tensión negativa de referencia del DAC para el código FFH. [3:0]: Define la tensión negativa de referencia del DAC para el código 00H.
0x22	0x05	R/W	[3:0]: Ajuste del nivel de continua de VCOM.

Tabla 5.11. Descripción del registro de configuración de la resolución de la pantalla LCD.

R02[2:0]			Secuencia Entrada	Secuencia Salida	Observación
0	0	0	400×RGB×240	800×RGB×480	Dual Scan
0	0	1	480×RGB×272	800×RGB×480	Dual Scan
0	1	0	Valor prohibido		
0	1	1	Valor prohibido		
1	0	0	Valor prohibido		
1	0	1	480×RGB×272	480×RGB×272	
1	1	0	Valor prohibido		
1	1	1	800×RGB×480	800×RGB×480	Valor por Defecto

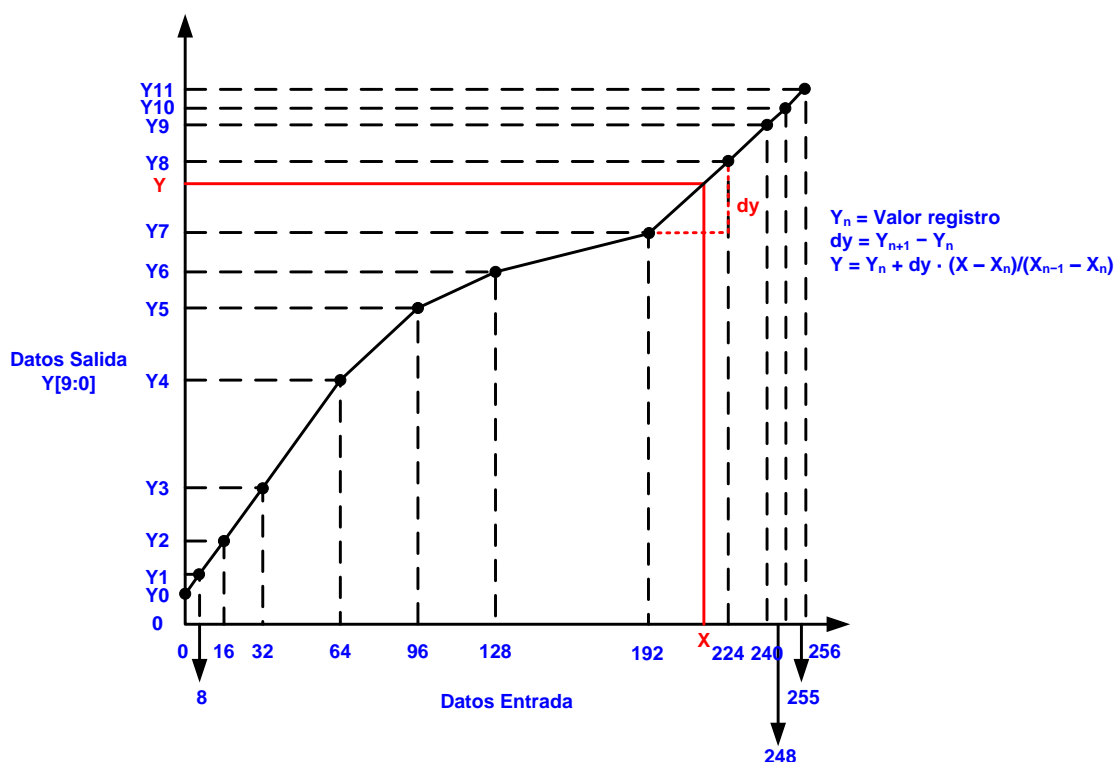


Figura 5.22. Diagrama de la corrección Gamma.