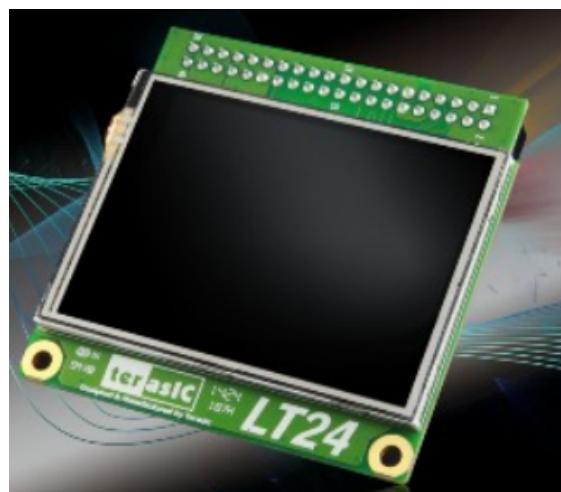
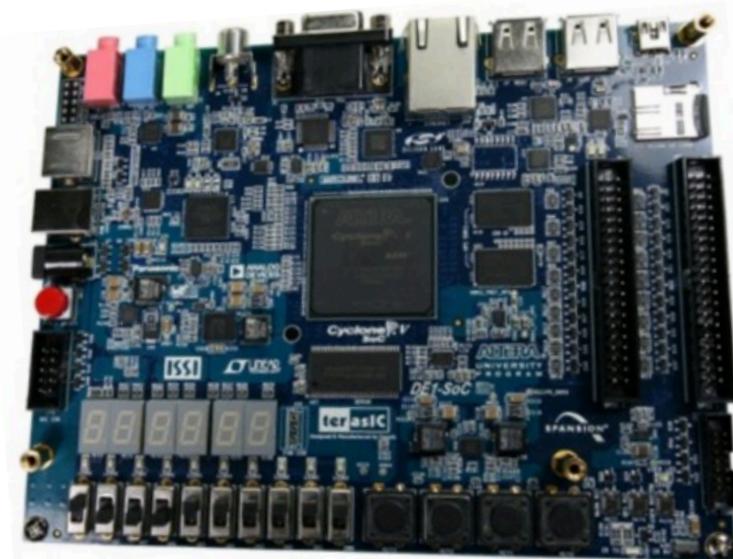


MEMORIA

DE1-SoC: Manejo de LCD

LT24



Autor: Gorka Dabó
Fecha: 03/11/2023

RESUMEN

El objetivo del proyecto es el de aprender a manejar, diseñar y construir un sistema digital programando la FPGA DE1-SoC y la pantalla LCD LT24. En este proyecto, hemos implementado 4 funcionalidades: Dibujar línea, borrar pantalla, dibujar imagen y video. La primera funcionalidad consiste en dibujar una línea diagonal desde la parte superior izquierda de la pantalla del color escogido por el usuario. La funcionalidad de borrar pantalla consiste en dibujar la pantalla completa de un mismo color escogido por el usuario. La funcionalidad de dibujar imagen, consiste en mandar una imagen desde el ordenador a la FPGA mediante el UART, y que esta imagen se pueda ver en la pantalla LT24. La última funcionalidad llamada ‘video’, consiste en mostrar una especie de video en la pantalla del LT24; para ello, primero se muestra la mitad de un cuadrado en la parte izquierda de la pantalla, luego en el centro y luego en la parte derecha de la pantalla a una velocidad de 24 FPS para crear la ilusión de un video.

La implementación de este proyecto la hemos dividido en 4 módulos: LCD_CTRL, LCD_DRAWING, UART y REC_PIXEL. Por cada uno de los módulos, hemos diseñado tanto las unidades de control como las unidades de proceso, y hemos utilizado ModelSim para poder modelar nuestros diseños. Por otro lado, también hemos utilizado ModelSim para simular los diseños implementados y así poder comprobar su funcionamiento antes de programarlo sobre la FPGA. Una vez estábamos seguros de que nuestros diseños estaban bien modelados, utilizando el software Quartus de Intel, hemos programado la FPGA para poder observar el funcionamiento de nuestros diseños, y si no eran correctos, hemos analizado el problema simulando los diseños otra vez en modelSim hasta que hemos logrado que todo funcione correctamente. En este informe, encontrarás toda la información necesaria para entender de qué trata el proyecto, qué métodos hemos utilizado para trabajar, las herramientas que hemos utilizado, los diseños de los módulos al detalle y los problemas que nos han surgido a lo largo del camino.

ÍNDICE

1. INTRODUCCIÓN	4
2. OBJETIVO DEL PROYECTO	7
3. METODOLOGÍA DE DISEÑO	8
3.1 Unidad de Control:	8
3.2 Unidad de proceso:	8
3.3 Diseño VHDL:	8
3.4 Simulación:	9
3.5 Implementación Hardware:	9
3.6 Test del prototipo:	9
3.7 Herramientas de Diseño, simulación y validación:	10
4. DISEÑO DEL SISTEMA	11
4.1 Descripción general del sistema:	11
4.2 División en módulos:	11
4.3 DESCRIPCIÓN DE CADA MÓDULO:	12
4.3.1 LT24_Init:	12
4.3.2 LCD CTRL:	13
4.3.3 LCD DRAWING:	17
4.3.4 UART:	24
4.3.5 REC_PIXEL:	28
5. TEST Y VALIDACIÓN	32
6. MANUAL DE USUARIO	35
7. CONCLUSIONES Y PROPUESTAS DE POSIBLES MEJORAS	36
8. BIBLIOGRAFÍA	37
9. APÉNDICES	37
10. TABLA DE DEDICACIÓN SEMANAL	38

1. INTRODUCCIÓN

El departamento de Arquitectura y Tecnología de Computadores de la Facultad de Informática de la UPV/EHU ha presentado su concurso anual de Diseño y Construcción de Sistemas Digitales. El concurso trata sobre implementar 3 funcionalidades mínimas y una extra sobre la FPGA De1-SoC con la pantalla LT24: Dibujar una línea diagonal en la pantalla, borrar la pantalla y dibujar imagen en la pantalla. La funcionalidad extra que hemos implementado consiste en mostrar un video en la pantalla LT24 de la FPGA De1-SoC. La FPGA De1-SoC y la pantalla LT24, cuando se usan conjuntamente, pueden ser programadas para controlar la pantalla LT24 y esto nos permite crear la interfaz de usuario necesaria para llevar a cabo las implementaciones de las funcionalidades previamente explicadas.

Tal y como se ve en la imagen de la FPGA, podemos observar que tiene muchas opciones para interactuar con el usuario, pero nosotros solamente usamos el botón de Power ON/OFF para encender y apagar la pantalla, los switch para escoger el color y para la funcionalidad extra y los 4 botones de la parte inferior de la FPGA. Por otro lado, también utilizaremos los puertos USB y UART para poder programar la placa y mandar las imágenes deseadas.

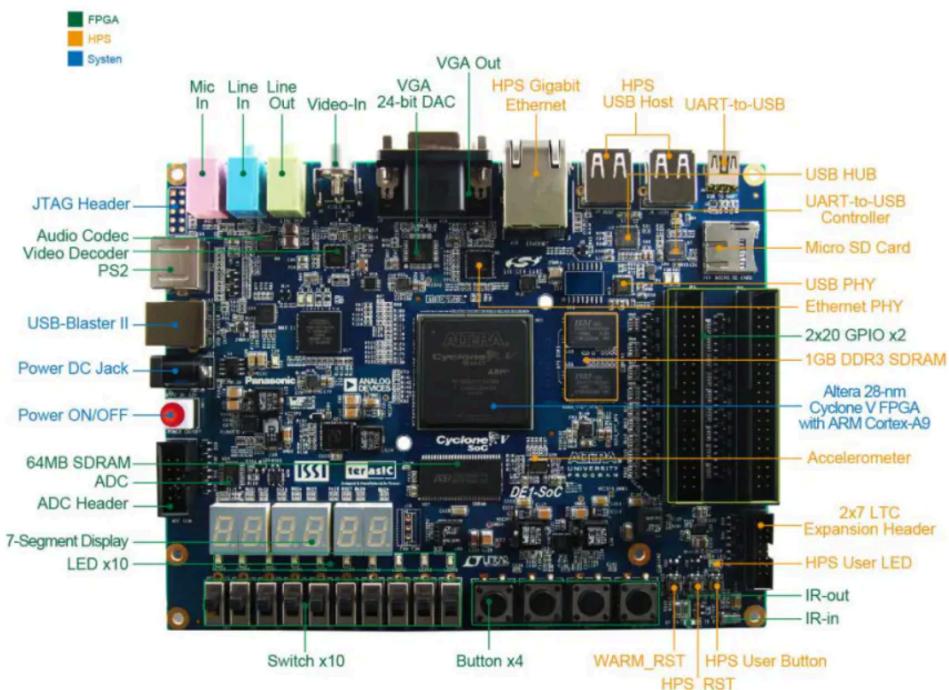


Figura 1: FPGA De1-SoC

Por otro lado, tal y como se muestra en la imagen posterior, la pantalla LT24 irá conectada a la FPGA mediante la GPIO (General Purpose Input/Output), y la conexión con la UART, se hará en la otra GPIO ya que la FPGA cuenta con dos disponibles.

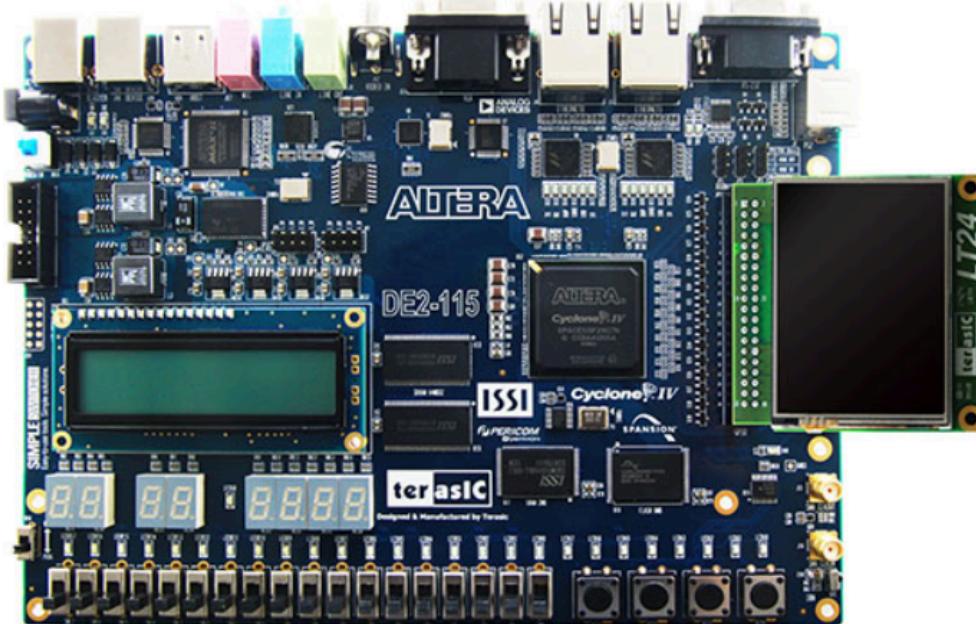


Figura 2: FPGA De1-SoC con Pantalla LT24

Una vez que ya hemos entendido qué componentes usaremos y cómo los conectaremos, el siguiente paso será implementar las funcionalidades que se piden. Para ello, hemos diseñado la unidad de proceso y la unidad de control de 4 módulos que trabajando conjuntamente realizan las 4 funcionalidades. Estos módulos son los siguientes: LCD_CTRL, LCD_DRAWING, UART y REC_PIXEL.

El módulo LCD_CTRL, es el módulo encargado de controlar los comandos que hacen que se mueva el cursor en la pantalla y que se dibujen los pixeles del color escogido. Para ello, está constantemente comprobando a ver si ha recibido la señal para mover el píxel o para pintar la pantalla. En el caso de que reciba una señal para mover el píxel, se guarda la posición de X e Y mediante uno registros, y para dibujar el pixel, se guarda el número de pixeles a dibujar en un registro. Además, utilizamos un sistema de un contador, un descodificador, un registro y dos multiplexores para generar los comandos necesarios para dibujar el píxel o mover el cursor.

El módulo LDC_DRAWING es el que se encarga de preparar y ordenar las funcionalidades en la FPGA, y para ello, hace uso de otros dos módulos: LCD_CTRL y REC_PIXEL. El módulo LCD_CTRL es utilizado para hacer la ‘acción’ de pintar y mover el cursor de la forma en la que se quiere en cada funcionalidad, por lo que el módulo LDC_DRAWING será el encargado de preparar los datos necesarios para el LCD_CTRL y el módulo REC_PIXEL es el encargado de agrupar un pixel entero, avisar al LDC_DRAWING de que el pixel está disponible y así poder mandárselo al LDC_DRAWING mediante un handshake para que éste pueda especificar al módulo LCD_CTRL de que color es el pixel que se quiere dibujar. Para ello, el módulo LDC_DRAWING hace uso de dos contadores para modificar el valor de X e Y, comparadores para generar las señales necesarias para saber si se ha llegado a alguna posición concreta, un registro para guardar el número de pixeles a dibujar y multiplexores tanto para escoger el color del pixel como para escoger distintos valores iniciales para cargar tanto en los contadores como en los registros.

El módulo UART es el encargado de recibir los bits mandados por el UART y almacenarlos de la manera correcta. También se encarga de mandar las señales necesarias al módulo REC_PIXEL mediante handshakes para mandar y avisar de que ha recibido los 8 bits mandados por el UART. Para ello, este módulo utiliza un registro de desplazamiento para guardar los bits recibidos por el UART, contadores para llevar el conteo de los bits recibidos y para hacer las esperas generadas por la diferencia de velocidades del reloj y del UART y un multiplexor junto a un restador para que se cargue el valor adecuado en el contador que se encarga de la espera.

El cuarto y último módulo, el REC_PIXEL, es el intermediario entre el módulo UART y LDC_DRAWING. Este módulo espera hasta que el módulo UART le avisa de que ya se tienen los primeros 8 bits y se guardan en un registro. Cada 2 veces que recibe esta señal, es decir, cada 16 bits recibidos avisa al LDC_DRAWING de que ya se tiene el color del pixel. El REC_PIXEL se hace uso de handshakes con los módulos con los que se comunica para que no ocurran problemas de comunicación.

Más adelante, en el apartado de Diseño del Sistema, cada uno de los 4 módulos está explicado detalladamente, y en el apartado de Test y Validación se pueden ver las simulaciones de los módulos para poder comprobar su funcionalidad.

2. OBJETIVO DEL PROYECTO

Primero de todo, al tener que realizar este trabajo en grupos de tres personas, uno de los objetivos es fomentar el trabajo en equipo, y con ello, el reparto de trabajo y la gestión de un proyecto entre varias personas, algo básico para nuestro futuro en el ámbito laboral. Además, trabajar en equipo nos ha dado la posibilidad de dividirnos el trabajo entre los compañeros por lo que el gran paralelismo que nos brindaba trabajar así nos ha sido de gran ayuda.

A la hora de programar los diferentes componentes, hemos utilizado el lenguaje VHDL, un lenguaje que hasta ahora era nuevo para nosotros, y que tiene notables diferencias con los entornos en los que hemos programado hasta ahora. Por lo tanto, otro de los objetivos es el de aprender a utilizar este lenguaje de programación.

En cuanto a los objetivos técnicos, tenemos que diseñar un sistema digital que sea capaz de dibujar una línea en diagonal en una pantalla, de mostrar un video sobre la pantalla, de recibir imágenes y mostrarlas en la pantalla, y de borrar dicha pantalla. Para después, poder plasmar y sintetizar este diseño en una FPGA (Field Programmable Gate Array), es decir, en una circuito integrado digital programable.

Finalmente, el objetivo principal es el de aumentar nuestro conocimiento y nuestras capacidades en el campo de los sistemas digitales, haciéndonos capaces de diseñar y crear sistemas complejos, eficaces y con un correcto funcionamiento.

3. METODOLOGÍA DE DISEÑO

A continuación, describiremos la metodología con la que hemos desarrollado nuestro diseño, para los diferentes apartados, que en conjunto forman el proyecto en su totalidad.

3.1 Unidad de Control:

En este proyecto son necesarios varias unidades de control, la unidad de control del LCD_CTRL, la unidad de control del LCD_DRAWING y la del módulo LT24_Init para la primera fase; junto con las mencionadas anteriormente, las de UART y REC_PIXEL para la segunda fase. Cada unidad de control es la encargada de gestionar y generar las señales de su unidad de proceso, además de gestionar el trabajo en conjunto entre los diferentes módulos.

3.2 Unidad de proceso:

La unidad de proceso es la otra parte indispensable en los diseños digitales junto con la unidad de control, si la unidad de control es el cerebro que se encarga de gestionar los componentes y sus respectivas señales, la unidad de proceso es el esqueleto, compuesto por dichos componentes. Entre estos encontramos tanto bloques combinacionales, como bloques secuenciales, además de los buses que se encargan de unir y transportar bits de datos entre los diferentes bloques.

3.3 Diseño VHDL:

Mediante el lenguaje de descripción de hardware VHDL (VHSIC Hardware Description Language) hemos modelado los diseños digitales. Este lenguaje es utilizado para modelar diseños de sistemas digitales en circuitos integrados, como es nuestro caso con la FPGA De1-Soc, porque permite describir el funcionamiento de cada bloque, además de poder simular dicho funcionamiento, obteniendo datos detallados como cronogramas que nos muestran los valores de cada señal en todo momento.

3.4 Simulación:

Las simulaciones y los testbench, son herramientas fundamentales en el proceso de diseñar y verificar las unidades de control, proceso y otros elementos de hardware proyecto. En este proyecto, la manera de proceder que hemos tenido con las simulaciones ha sido la de dar valores a las señales de entrada de los módulos para poder verificar el comportamiento de los diseños. Para poder hacer esto, hemos utilizado un testbench por cada módulo, que no es más que un entorno de prueba utilizado para verificar la funcionalidad de nuestros diseños. Cabe recalcar, que en algunos módulos como el LCD_DRAWING, algunas de las señales de entrada las generan el resto de módulos como bien puede ser el LCD_CTRL, y como nosotros hemos hecho una simulación independiente por cada módulo, hemos generado estas señales manualmente calculando los tiempos de espera para las activaciones de las señales que ocurren cuando todos los módulos interactúan entre sí.

3.5 Implementación Hardware:

Para el apartado de implementación hardware, la metodología que hemos seguido es la de primero diseñar el los módulos de LCD_DRAWING, UART y REC_PIXEL, después modelar el diseño utilizando el lenguaje de descripción VHDL y una vez hecho esto, hemos añadido los archivos VHDL al proyecto ‘top’ de Quartus. En este proyecto, se define la estructura y la interconexión de los componentes y los módulos del proyecto. Después, hemos compilado el código y arreglado los errores de compilación, y para finalizar hemos programado la FPGA para que pueda procesar el proyecto de Quartus y así comprobar que funciona como debería.

3.6 Test del prototipo:

Para hacer los test del prototipo la metodología que utilizamos en un principio era la de prueba y error, pero como nos dimos cuenta de que no era la manera más efectiva de corregir los errores, cambiamos el enfoque para que cada vez que algo no funcionaba como queríamos, analizaremos las simulaciones y el diseño estado a estado para poder encontrar el error. Es por esto que primero nos aseguramos de que las simulaciones mostraran el comportamiento que queríamos en cada módulo, y después, comprobamos si el funcionamiento de la FPGA era correcto para que en

el caso de que no lo fuese, pudiéramos hacer las modificaciones necesarias en los diseños y los archivos vhdl para volver a testearlo sobre la FPGA hasta llegar a una versión final correcta.

3.7 Herramientas de Diseño, simulación y validación:

Para hacer el diseño, al inicio del proyecto no hemos utilizado ninguna herramienta en concreto ya que todo el diseño lo hicimos en lápiz sobre una hoja; pero al finalizar el proyecto, hemos utilizado las herramientas de google drawing y microsoft visio para pasar a ordenador las unidades de proceso y control respectivamente. Además, hemos utilizado modelSim para modelar en el lenguaje de descripción VHD y hacer las simulaciones. Por otro lado, para programar la FPGA con nuestros diseños, y para la validación y el testeо sobre el hardware hemos utilizado un software de Intel llamado Quartus el cual es un software de diseño para circuitos integrados programables, específicamente FPGAs.

4. DISEÑO DEL SISTEMA

4.1 Descripción general del sistema:

En la primera fase, este sistema realizaba las tareas de dibujar una línea en la pantalla LT24 y borrar la pantalla LT24. Para ello, se combinan los módulos ‘LCD Control’ y ‘LCD Drawing’. El primero realiza la tarea de colocar el cursor y de dibujar píxeles en la pantalla, mientras que el segundo se encarga de decidir si pintar toda la pantalla o de trazar tan solo una línea. La sincronización y el trabajo entre estos dos módulos son vitales para el correcto funcionamiento del sistema. Hay un tercer módulo, el LT24Setup que se encarga de inicializar la pantalla y trabaja con los demás módulos.

En la segunda fase, hemos implementado la funcionalidad de dibujar una imagen que se haya enviado desde el ordenador mediante la UART. Para lograrlo, hemos añadido dos nuevos módulos, el módulo UART, que se encarga de transmitir a la FPGA la información mandada desde el ordenador, y el módulo REC_PIXEL con el que se unen dos bytes con la información del color transmitidos mediante la UART para formar un píxel. Además, también hemos hecho cambios en el módulo LCD_DRAWING para implementar esta nueva función.

4.2 División en módulos:

Para facilitar el diseño del sistema, lo hemos organizado en 5 módulos: LT24Setup, LCD_CTRL, LCD_DRAWING, REC_PIXEL y UART. Todos los módulos están relacionados entre sí y muchas de las señales de entrada de un módulo son las entradas de otro y viceversa. También utilizamos handshakes entre los distintos módulos para asegurarnos de una correcta comunicación. Para hacer esto, el primer módulo una vez que tiene el dato que quiere transmitir preparado activa una señal de aviso que recibe el segundo módulo, y el primer módulo se queda a la espera de recibir la confirmación (ACK) del segundo módulo para seguir con su ejecución. El segundo módulo, constantemente a la espera de la señal del primer módulo que avisa de que el dato está preparado, y una vez esta señal se active, procesará el dato recibido y mandará la confirmación al primer módulo.

4.3 DESCRIPCIÓN DE CADA MÓDULO:

Como bien hemos explicado anteriormente, para poder implementar todas las funcionalidades del proyecto, hemos dividido el diseño en varios bloques diferentes, que trabajan en conjunto realizando diferentes funciones. En esta sección encontrarás cada uno de los módulos explicado detalladamente junto a una la imagen correspondiente del diagrama de cada módulo.

4.3.1 LT24 Init:

Funcionalidad: Su tarea es la de inicializar y comunicarse con la pantalla LT24. Al activar la señal ‘Reset’ se inicia automáticamente, y al completar el proceso de inicialización activa la salida ‘LT24_Init_Done’. Una vez hecho esto, el dispositivo estará a disposición de las demás señales de entrada para transmitir la información de los demás módulos a la pantalla.

SEÑALES DE ENTRADA Y SALIDA

Señales de entrada: LT24_CS_N_Int, LT24_WR_N_Int, LT24_RD_N_Int, LT24_RS_Int, LT24_D_Int.

Señales de salida: LT24_LCD_ON, LT24_RESET_N, LT24_CS_N, LT24_WR_N, LT24_RD_N, LT24_RS, LT24_D, LT24_Init_Done.

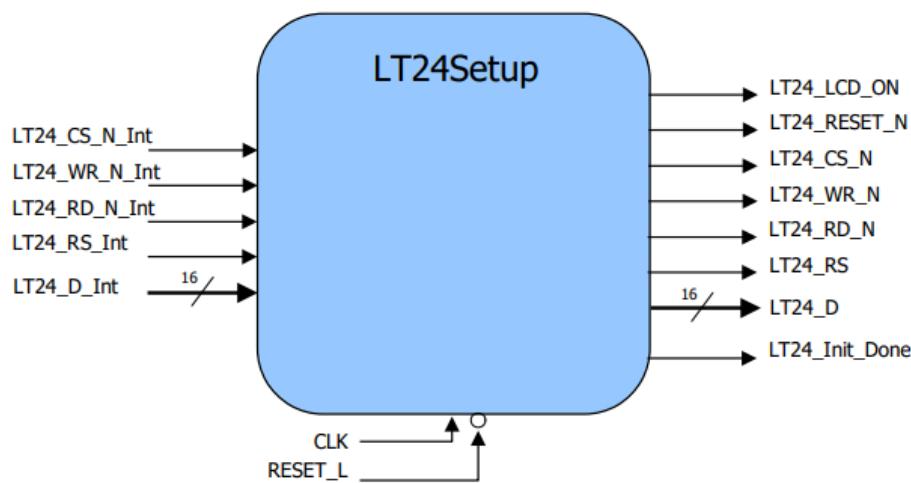


Figura 3: Diagrama del módulo LT24

4.3.2 LCD CTRL:

Funcionalidad: El módulo de control se encarga de las operaciones básicas sobre la pantalla, como colocar el cursor en la posición deseada o dibujar un número concreto de píxeles.

SEÑALES DE ENTRADA Y SALIDA

Señales de entrada: LCD_Init_Done, OP_SETCURSOR, XCOL, YROW, OP_DRAWCOLOUR, RGB, NUM_PIX.

Señales de salida: DONE_CURSOR, LCD_DATA, DONE_COLOUR, LCD_CS_N, LCD_WR_N, LCD_RS.

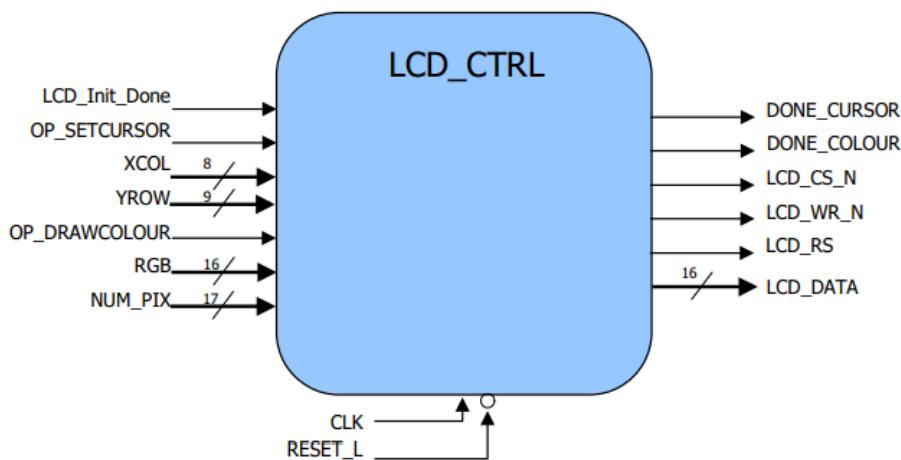


Figura 4: Diagrama del módulo LCD_CTRL

En la unidad de proceso se han utilizado los siguientes componentes:

- 3 registros para guardar las entradas XCOL, YROW y RGB.
- Un contador descendente para guardar la entrada NUMPIX.
- Dos multiplexores y un contador para conseguir el valor adecuado en cada momento en la salida LCD_DATA.
- Un descodificador para controlar el valor del contador.
- Un registro para el valor de RS.

LCD CTRL(UP):

PROCESS UNIT

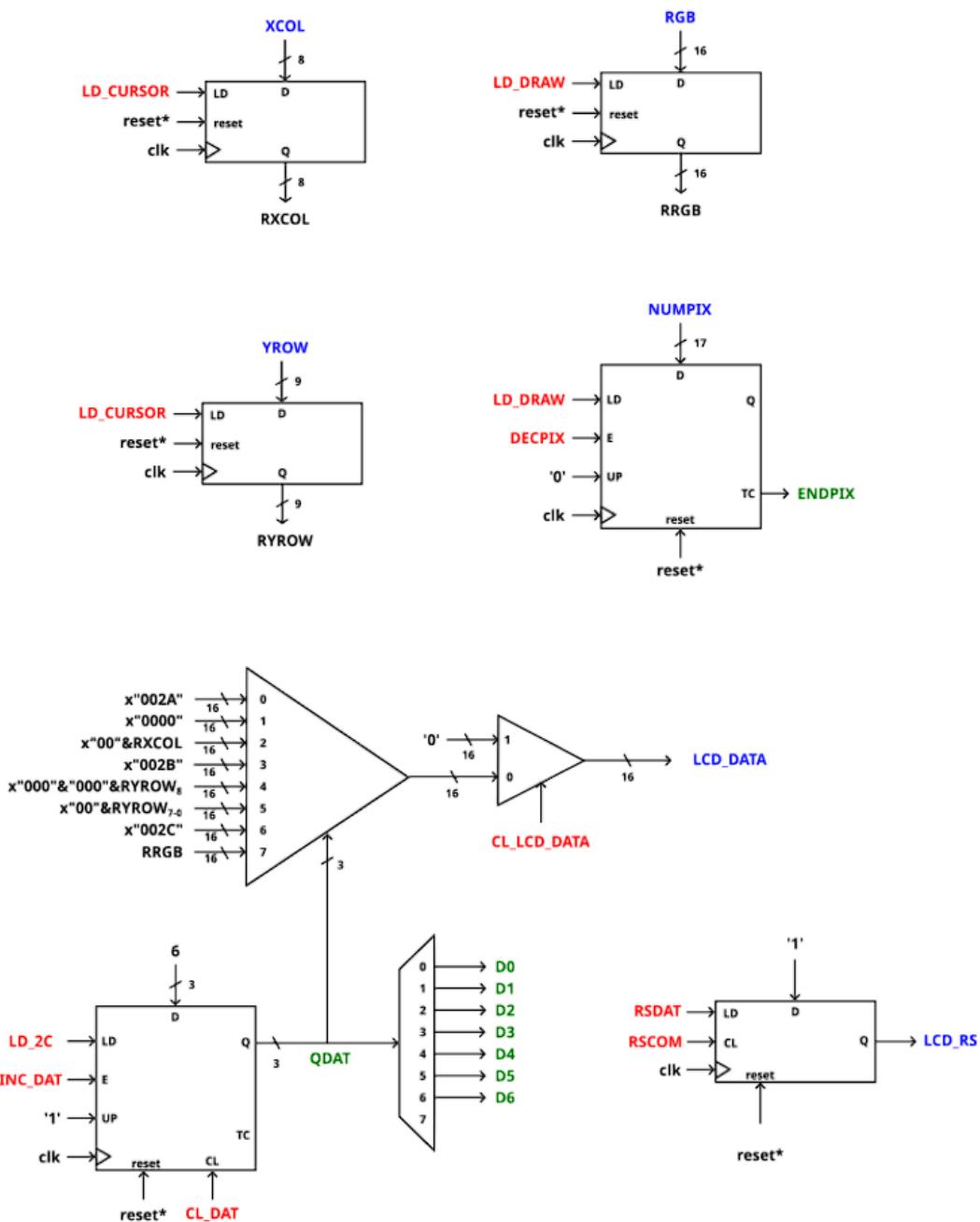


Figura 5: UP de LCD_CTRL

La unidad de control cuenta con 14 estados diferentes. Empieza con el estado E0, en el que se asegura de que LCD_DATA sea igual a 0, activando CL_LCD_DATA y se comprueba que la pantalla esté inicializada. En el caso de estarlo, hay que comprobar que operación se desea realizar entre OP_SETCURSOR que nos llevará al estado E1 y OP_DRAWCOLOUR que nos lleva a E14.

El posicionamiento del cursor se realiza mediante los valores de las salidas LCD_WR_N, LCD_CS_N, LCD_RS y LCD_DATA. En el estado E1 se hacen las inicializaciones, y después mediante los estados E2, E3, E4, E11, E12 y E13 iremos enviando los datos utilizando un bucle. Iremos al estado E11 al finalizar el bucle.

Para dibujar píxeles hay que controlar las mismas salidas que controlamos para el posicionamiento. En el estado E14 se realizan las inicializaciones, y después los estados E2, E3, E4 y E5 hacen que se envíe el comando ‘2C’, para finalmente mediante un bucle con los estados E6, E7, E8 y E9 dibujar el número de píxeles deseado. Cuando el contador descendente llega a 0 finaliza el bucle y va al estado E10 donde se activa DONE_COLOUR.

LCD CTRL(UC):

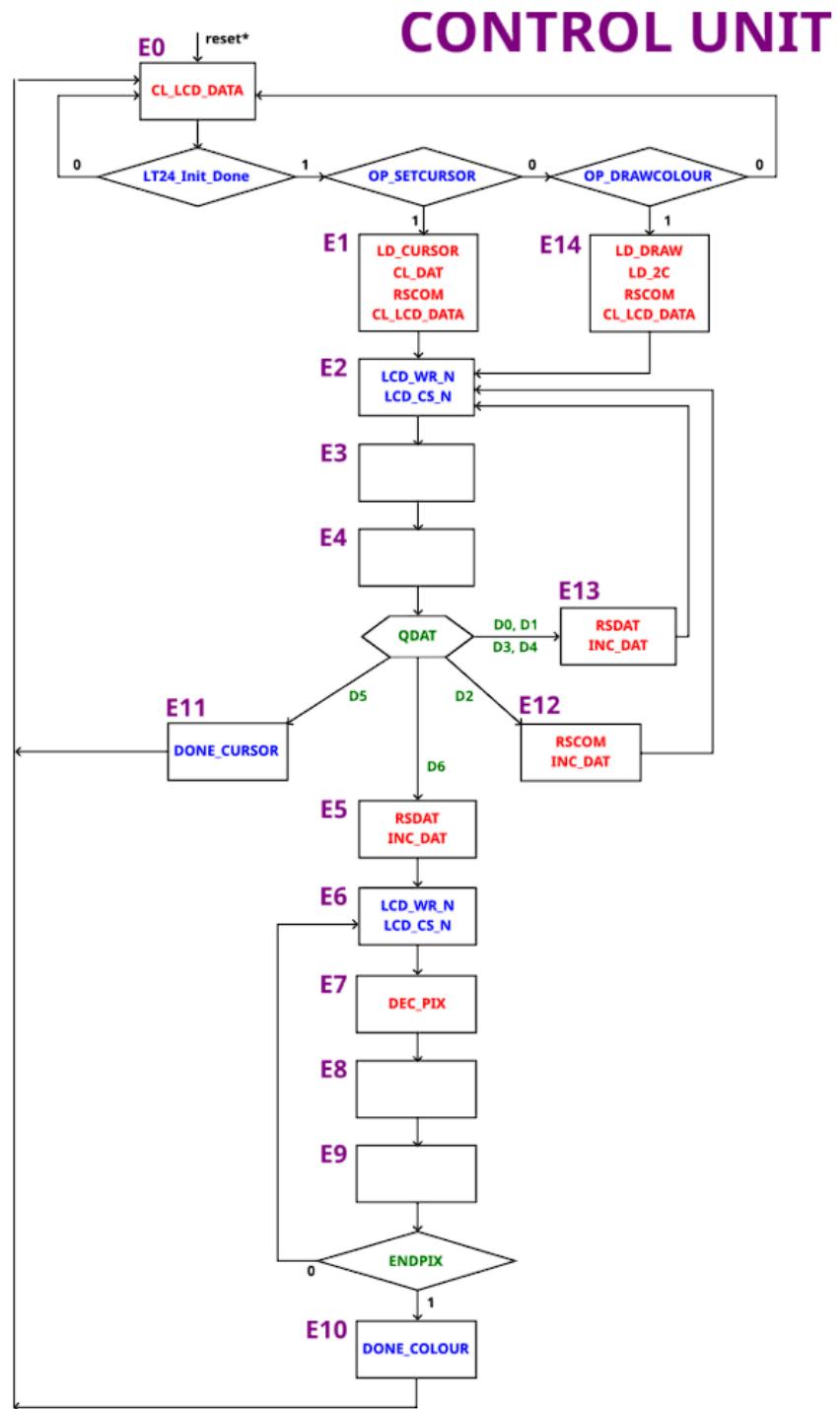


Figura 6: UC de LCD_CTRL

Descripción VHDL: en el VHDL, lo primero ha sido crear la entidad del LCD_CTRL donde se definen los puertos de entrada y salida correspondientes. Después, se define la arquitectura del mismo y se declaran las señales y los estados del LCD_CTRL. A continuación, se implementa la unidad de control para lo cual se crean tanto un proceso para el registro de estados como otro proceso para la lógica de escoger el estado siguiente. Más adelante, se implementa la lógica de generación de señales de control y para finalizar se implementa la unidad de proceso. Para implementar la unidad de proceso, se crea un proceso por cada bloque utilizado en la UP y también se implementa el multiplexor de los comandos.

Simulación: Para la simulación del VHDL, hemos creado un testbench en el que después de generar una instancia de la arquitectura del LCD_CTRL, y relacionar las señales del testbench con las del LCD_CTRL, hemos simulado el funcionamiento del mismo, activando, desactivando y dando valores a las señales que necesitábamos. Después de esto, hemos comprobado en el cronograma que el resultado era el esperado. Más adelante, en la sección de ‘Test y Validación’, encontrarás las simulaciones y los cronogramas de cada módulo explicado al detalle.

4.3.3 LCD DRAWING:

Funcionalidad: El módulo LCD_DRAWING decide si dibujar toda la pantalla, la línea diagonal, la imagen del UART o el video. Para borrar la pantalla, en realidad estamos pintando toda la pantalla del mismo color, y para dibujar una línea tan solo pintamos los píxeles necesarios. Por otro lado, para mostrar la imagen recibida a través del UART en la pantalla tiene que ir de pixel en pixel preparando las señales necesarias para pintarlo del color correcto. Además, para mostrar el video primero borra toda la pantalla (la dibuja enteramente de blanco) y luego pinta los píxeles necesarios para mostrar la mitad o el cuadrado completo en la posición correspondiente para dar esa ilusión de que el cuadrado

se mueve de izquierda a derecha. En definitiva, genera las señales necesarias para el LCD_CTRL

SEÑALES DE ENTRADA Y SALIDA

Señales de entrada: DEL_SCREEN, DRAW FIG, DRAW_IMAGE, VIDEO, COLOUR_CODE, DONE_CURSOR, DONE_COLOUR, Pixel_Rec y Pixel.

Señales de salida: OP_SETCURSOR, XCOL, YROW, OP_DRAWCOLOUR, NUM_PIX, RGB, PixelACK.

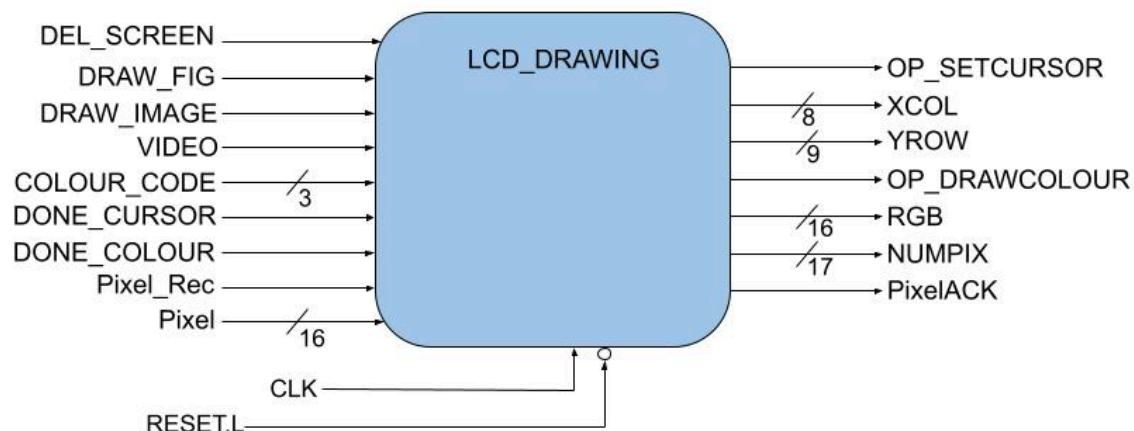


Figura 7: Diagrama del módulo LCD_DRAWING

En la unidad de proceso se han utilizado los siguientes componentes:

- 2 contadores para las coordenadas x e y junto a un multiplexor en la entrada para los casos en los que no se quiera empezar desde la posición (0,0).
- Dos comparadores para saber si se ha llegado a la última columna y para saber si se ha llegado al último píxel (el píxel de la última fila y columna).
- 1 registro para guardar la cantidad de píxeles segundos a dibujar que varía dependiendo de la funcionalidad, y otro registro para guardar el código del color a utilizar para pintar el pixel en algunas funcionalidades.
- Un multiplexor para escoger si el valor de RGB lo determinará el pixel recibido por la uart, el código del color escogido mediante los switch o para escoger los colores blanco y negro directamente (para la funcionalidad del video).
- Además, hay tres contadores que sirven para esperar los ciclos necesarios para que el video vaya a los FPS adecuados, para poder escoger el frame

que se quiere mostrar en la pantalla y para contar las filas pintadas para el cuadrado (el cuadrado será de 80x80 aunque en el primer y último frame solamente se muestra la mitad para dar la impresión de que el cubo sale y entra por la pantalla).

LCD DRAWING(UP):

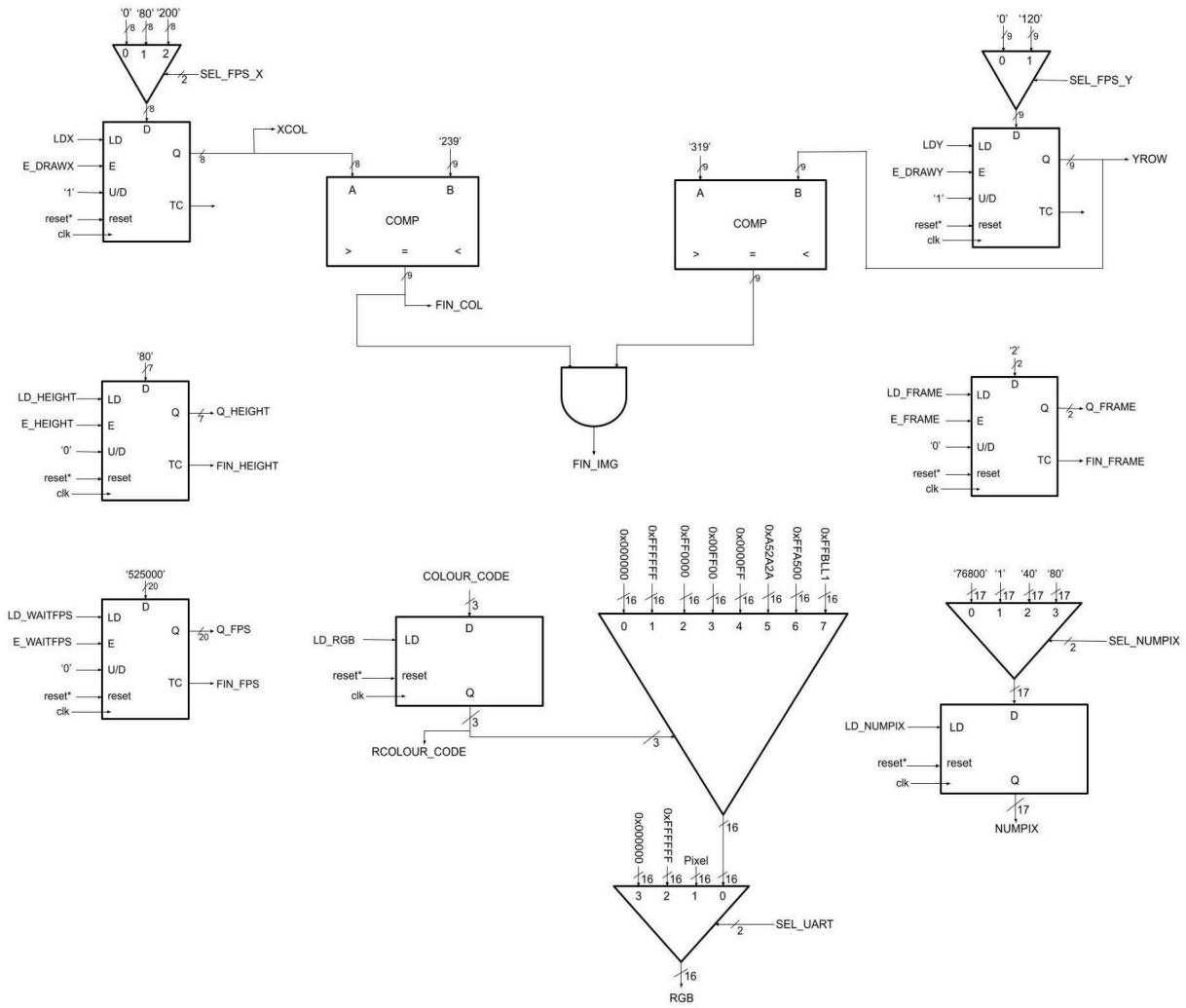


Figura 8: UP LCD_DRAWING

La unidad de control de la figura 9 consta de 28 estados diferentes. Al resetear el sistema, empezaremos en el E0, donde se decide si borrar la pantalla (DEL_SCREEN=1), si dibujar una línea diagonal, (DRAW FIG=1), si recibir y

mostrar la imagen a través del UART (DRAW_IMAGE=1) o si mostrar el video en la pantalla (VIDEO=1).

En el caso de que queramos borrar la pantalla, iremos al estado E1 y cargaremos la posición (0,0) en XCOL e YROW y el número de píxeles a pintar será 76800 (LD_NUMPIX). Después, posicionamos el cursor en la posición que hemos cargado en el estado E2, y una vez esté en su posición, es decir, cuando se active DONE_CURSOR, pasaremos a E3, donde se pintarán los píxeles deseados. Al estar todos los píxeles pintados finalizamos la operación volviendo a E0.

Para dibujar la línea diagonal, comenzamos en E4, donde cargamos el código del color (LD_RGB), la posición del cursor, el número de píxeles seguidos a pintar (LD_NUMPIX), que en este caso será uno (SEL_NUMPIX=1). Pasamos al bucle realizado entre los estados E5, E6 y E7. En el estado E5 posicionamos el cursor en la posición a pintar, después en el E6 pintamos ese píxel y en el E7 sumamos uno a los contadores que cuentan almacenen los valores de X e Y. El bucle finaliza cuando llegamos a la posición 240 de X y regresamos a E0.

Para la funcionalidad del UART, comenzamos en E8 (DRAW_IMAGE=1) donde se cargan los valores (0,0) en XCOL e YROW y además escogemos un solo píxel ya que por cada pixel, como el color puede ser destino, tenemos que hacer las operaciones de mover y dibujar el pixel. Luego, activamos OP_SETCURSOR para avisar al LCD_CTRL de que mueva el pixel y esperamos a que se active la señal DONE_CURSOR para confirmar que el cursor ha sido movido.

Después, nos quedamos a la espera en E10 de recibir la señal (PIXEL_REC)que nos avisa de que ya están los 16 bits del pixel disponibles para que podamos dibujar el píxel de la pantalla del color adecuado. Una vez se activa dicha señal activamos la señal PixelACK en E11 para avisar al resto de módulos de que ya hemos recibido el pixel y en E12 activamos la señal que avisa a LCD_CTRL de que puede dibujar el pixel. Como la carga del valor del píxel se hace un siglo después de que se active OP_DRAWCOLOUR, en E13 activamos la señal SEL_UART=1 para que en el multiplexor RGB tome el valor del Pixel recibido por la UART y nos quedamos a la espera de recibir la señal de que ya se ha dibujado el pixel en la pantalla.

Una vez llegamos a E14 sumamos 1 al valor de X para ir al siguiente píxel; cambio que se efectuará en el siguiente ciclo, es decir, en el caso de que estemos dentro de los límites de la pantalla ya que comprobamos si hemos llegado a la última columna, y en el caso de que si comprobamos si hemos llegado también a la última columna para que en el caso de que si, al haber llegado al final de la pantalla, podamos volver a E0, no obstante, en el caso de que estemos en la última columna pero no en la última fila, sumamos 1 al valor de Y, y cargamos un 0 en X para pasar a la primera columna de la siguiente fila, además, volvemos al estado E9 donde se activa la señal correspondiente para avisar al LCD_CTRL de que mueva el cursor a la posición (XCOL,YROW) y repetir el ciclo.

Para finalizar, tenemos la funcionalidad de video donde empezaremos en el estado E16 (VIDEO=1). En este estado cargamos el valor ‘2’ en el contador de QFRAME. Esto se hace así porque para hacer la ilusión del video, mostramos el cuadrado en 3 posiciones distintas en la pantalla constantemente, primero en la parte izquierda, luego en el centro y para finalizar en la parte derecha de la pantalla. Esta secuencia se repite constantemente para crear esta ilusión, y con este contador controlamos el ‘frame’ en el que estamos que bien podrá ser 2, 1 o 0 ya que hay 3 posiciones posibles.

Luego, desde E17 hasta E19 es lo mismo que en la funcionalidad de borrar pantalla con la diferencia de que en E19, que es cuando se avisa al LCD_CTRL de que dibuje la pantalla, RGB toma el valor Blanco ya que SEL_UART toma el valor 2, esto es así porque hemos decidido que el fondo del video sea siempre blanco. Después, se comprueba el valor que tiene Q_Frame para saber en qué lugar de la pantalla tenemos que dibujar el cuadrado. Si Q_Frame tiene el valor 2, dibujaremos la mitad del cuadrado en la parte izquierda de la pantalla para hacer la ilusión de que el cuadrado está entrando por la parte izquierda de la pantalla. Para lograr esto, cargamos el valor 120 en YROW para que el centro del cuadrado quede en mitad de la pantalla, y 0 en XCOL para dibujar en la parte izquierda de la pantalla.

Además, cargamos 40 en NUMPIX ya que solo queremos dibujar la mitad del cuadrado que es del tamaño 80x80. Después, activamos OP_SETCURSOR en E23 para que el LCD_CTRL mueva el cursor a la posición que le hemos indicado y esperamos a que se complete la operación. En E24 activamos OP_DRAWCOLOUR junto a SEL_UART=3 para que el cuadrado se dibuje de

color negro. Esperamos a que se dibujen los 40 pixeles seguidos, y en E25 restamos 1 en el contador de la altura que empieza en 80 para ir controlando cuantas filas llevamos dibujadas de las 80 que tiene el cuadrado de altura.

También, comprobamos si hemos dibujado ya las 80 filas y si es que sí, nos quedamos a la espera de que pasen los 525000 ciclos para que estas operaciones de dibujar el cubo en posiciones distintas no se hagan demasiado rápido ya que el movimiento del cuadrado sería imperceptible para el ojo humano. Una vez que la espera se acaba, restamos 1 a Q_Frame para que en la siguiente iteración el cuadrado se dibuje en el centro de la pantalla, y volvemos a E17 para que se borre toda la pantalla de blanco como hemos explicado antes. En el caso de que en E25 no hayamos dibujado las 80 filas que tiene de altura el cuadrado, cargamos un 0 en XCOL y sumamos 1 a YROW para volver a E23 y que activando OP_SETCURSOR el cursor se coloque en la primera columna de la siguiente fila. Después de borrar la pantalla entera de blanco, como Q_Frame tiene 1 de valor, llegaremos a E21 donde se cargará el valor 120 en YROW y 80 en XCOL para que al ser el cuadrado de tamaño 80x80, el centro del mismo quede exactamente en el centro de la pantalla.

Asimismo, NUMPIX toma el valor 80 ya que esta vez tenemos que dibujar el cuadrado entero. Después, llegaremos al estado E23 donde se repetirá el mismo proceso anteriormente mencionado hasta que lleguemos a E17 otra vez y se vuelva a borrar toda la pantalla de blanco. Llegados a este punto solamente quedará dibujar la mitad del cuadrado en la parte derecha de la pantalla por lo que al Q_Frame tener 0 de valor, llegaremos a E22 donde cargaremos 120 en YROW y 200 en XCOL además de 40 en NUMPIX. Además de esto, también cargamos el valor 2 en Q_Frame otra vez ya que para hacer la ilusión de que el cuadrado se mueve, el siguiente paso sería dibujar el cuadrado en la parte izquierda de la pantalla por lo que al llegar a E19 en la siguiente iteración, pasaremos a E20 y se volverá a repetir la secuencia entera. Después de E22, iremos a E23 por lo que se volverá a repetir la operación de pintar el cuadrado y volver a E17. Como podrás observar, una vez que empieza la operación de video, no se termina nunca a no ser que el usuario apriete el botón del reset.

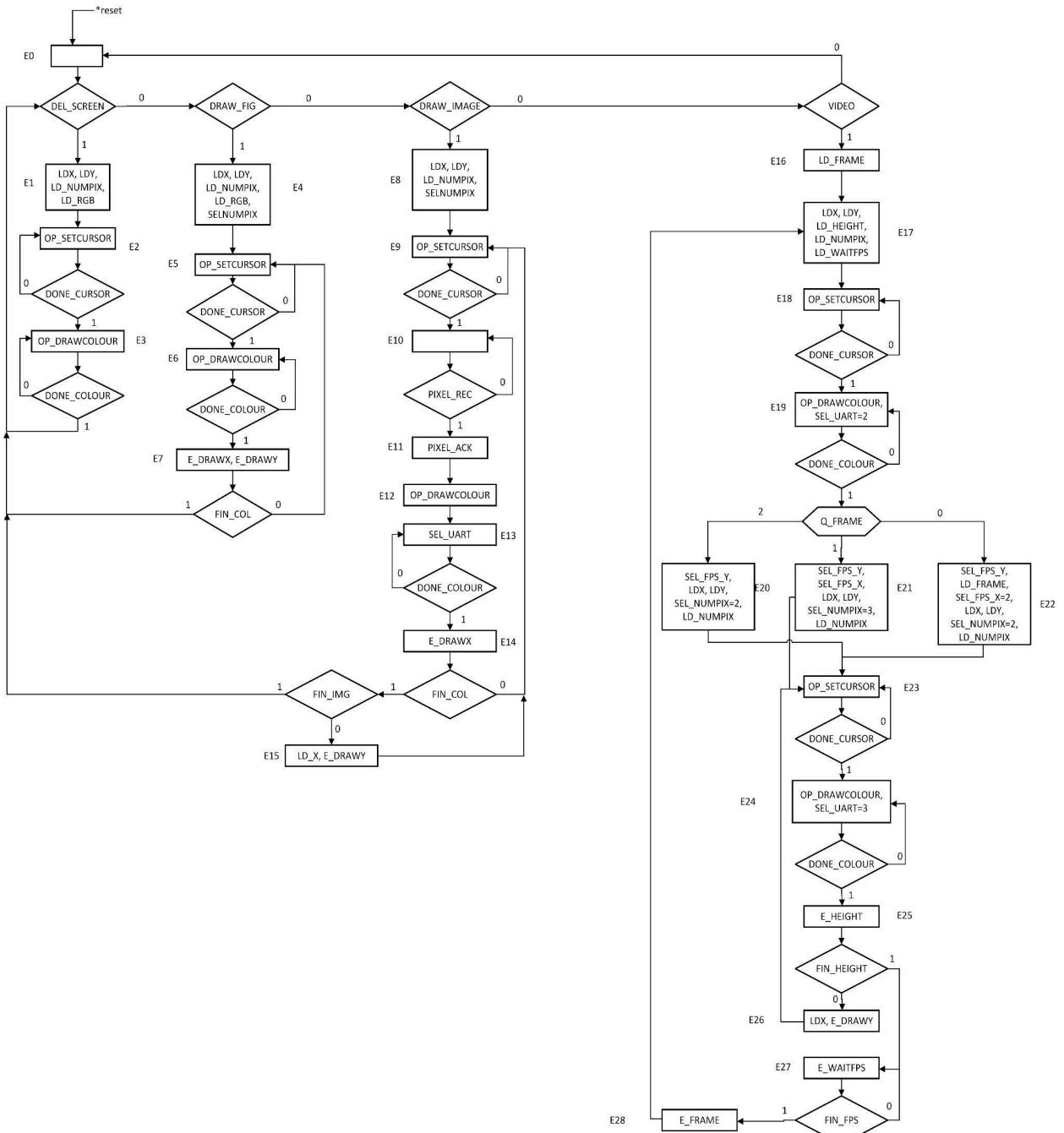


Figura 9: UC LCD_DRAWING

Descripción VHDL: Primero, hemos creado la entidad de LCD_DRAWING con los puertos de entrada y salida correspondientes. Después, hemos definido la arquitectura de la entidad declarando las señales y los estados del LCD_DRAWING. Por otro lado, hemos implementado la unidad de control por

lo que hemos generado un proceso para registrar el estado actual y otro para decidir el estado siguiente. También hemos añadido la lógica de activación de señales. Para finalizar, hemos implementado la unidad de proceso del LCD_DRAWING por lo que hemos creado un proceso por cada bloque de la UP y también está implementado el multiplexor del RGB con los códigos RGB hexadecimales escogidos.

Simulación: Para simular el VHDL del LCD_DRAWING, hemos creado un testbench en el que además de crear una instancia de la arquitectura del mismo, hemos relacionado las señales de la entidad con los que vamos a usar en el testbench para dar valores. Después, hemos simulado la funcionalidad de borrar pantalla, dibujar una línea dando valores a las señales manualmente, la funcionalidad del UART activando y desactivando las señales necesarias con las esperas correspondientes y la funcionalidad de Video activando las señales adecuadas para ello. Para finalizar, hemos comprobado que el resultado de la simulación era el esperado mediante el cronograma. En el apartado de test y validación encontrarás el cronograma y su explicación detallada.

4.3.4 UART:

Funcionalidad: La UART es el módulo encargado de transmitir los datos que se envían desde el ordenador a la FPGA. La velocidad de transmisión es de 115200 baudios por lo que se debe esperar una cantidad determinada de ciclos cada vez que se envía un bit.

SEÑALES DE ENTRADA Y SALIDA

Señales de entrada: RX y RecACK.

Señales de salida: DATA_READY y DATA.

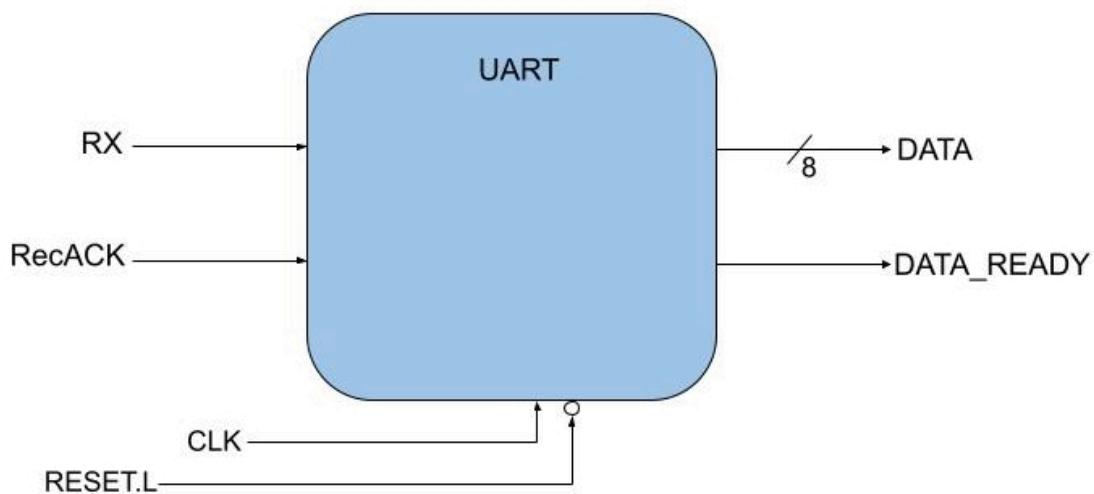


Figura 10: Diagrama del módulo UART

En la unidad de proceso se han utilizado los siguientes componentes:

- Un registro de desplazamiento en el que ir guardando los bits que se reciben desde RX.
- Un contador para saber cuando se han cargado los 8 bits en el registro de desplazamiento.
- Un contador para saber cuando han pasado los ciclos que se debe esperar para poder recibir un nuevo bit (generalmente 434).
- Un contador para contar los ciclos que han pasado en una determinada parte de la ejecución, acompañado de un restador, que restará la cantidad de ciclos que han pasado a 434, obteniendo la nueva cantidad de ciclos a esperar.
- Un multiplexor que decidirá si se deben esperar 434 ciclos, o la cantidad definida mediante la resta.

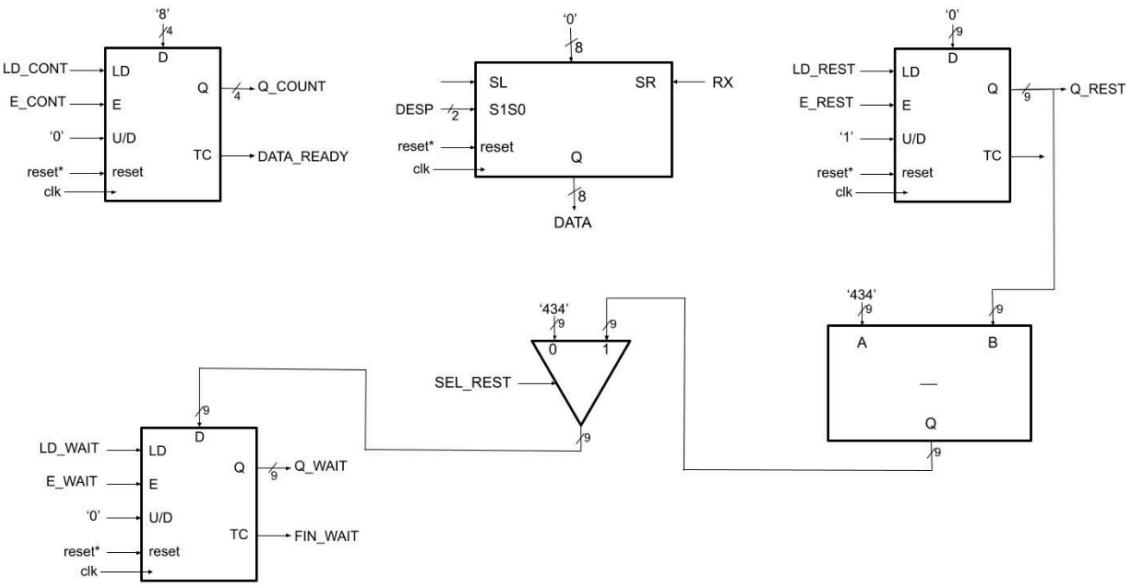


Figura 11: UP UART

La unidad de control de la UART tiene 7 estados. Comenzamos en el E0 cuando se resetea el sistema. En este estado, activaremos la señal LD_WAIT que cargará los ciclos a esperar en el contador. Cuando recibamos un bit con el valor ‘0’ significa que ha comenzado la transmisión de datos, y pasaremos al estado E1, en el que cargaremos los valores de los otros contadores, además de darle el valor ‘11’ a DESP, lo que hará que se cargue el valor ‘0’ en 8 bits.

Hasta que se active la señal DATA_READY, es decir, hasta que se hayan cargado los 8 bits de la transmisión en el registro de desplazamiento, primero entraremos en un bucle en el estado E2 en el que esperaremos los ciclos necesarios para recibir un bit. Una vez hayamos esperado los ciclos se activará FIN_WAIT y pasaremos al estado E3, en el que volveremos a cargar los ciclos a esperar, para el siguiente bit, y daremos a DESP el valor ‘01’ con el que desplazaremos a la izquierda añadiendo el nuevo bit a la derecha. También restaremos uno al contador que lleva la cuenta de los bits recibidos. El estado E4 es un estado vacío pero necesario, porque el sistema necesita dos ciclos para decidir cuál será el siguiente estado, por lo que de no estar este estado, volvería a activar una segunda vez todas las señales del estado E3.

Una vez tengamos el dato al completo, deberemos esperar a que el módulo REC_PIXEL nos mande la señal RecACK, indicando que ha recibido los primeros 8 bits, mientras se envía, entraremos en otro bucle en E5 en el que

iremos contando los ciclos que pasan hasta que se recibe la señal RecACK. Cuando la recibamos pasaremos al estado E6 en el que volveremos a cargar el contador de bits recibidos y el de los ciclos a esperar, esta vez, en este último el valor cargado no será 434, sino el resultado de la resta entre 434 y los ciclos que hemos estado esperando a RecACK, al activar SEL_REST.

Finalmente, entraremos en un último bucle el estado E7 en el que esperaremos los ciclos determinados en el estado anterior, hasta que volveremos al estado E0, donde comenzaremos a recibir un nuevo dato.

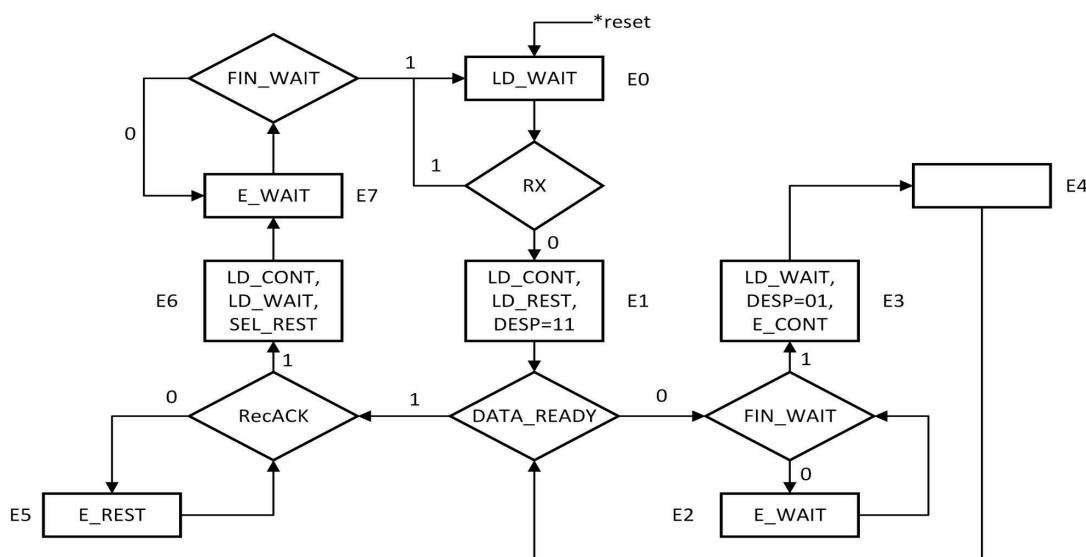


Figura 12: UC UART

Descripción VHDL: Primero, hemos creado la entidad de UART con los puertos de entrada y salida correspondientes mostrados en la figura 10. Después, hemos definido la arquitectura de la entidad declarando las señales y los estados del UART. Por otro lado, hemos implementado el proceso de la unidad de control y otro proceso para decidir el estado siguiente. Además, hemos implementado la lógica de activacion de señales y para acabar, hemos implementado la unidad de proceso del UART para lo que hemos creado un proceso por cada bloque de la UP.

Simulación: Hemos diseñado un testbench llamado tb_UART para simular y probar nuestra entidad UART. Primero que nada, creamos la entidad de tb_UART sin ningún puerto y definimos la arquitectura de la simulación, es decir, el puerto y las señales, que serán las mismas que en el UART.vhd.

Despues, definimos el DUT (Device Under Test) que es donde hemos conectado las señales con los puertos correspondientes del UART. Luego hemos definido un ciclo de reloj de 20ns, y para finalizar, hemos dado los valores adecuados al reset y al RX con los tiempos de espera reales que ocurren en la ejecución con la finalidad de poder comprobar su funcionamiento al analizar el cronograma.

4.3.5 REC_PIXEL:

Funcionalidad: La UART envía los bits de ocho en ocho, y para saber el color de un pixel necesitamos 2 bytes, es decir, 16 bits. Este módulo es el encargado de unir los primeros 8 bits que recibirá de la UART con los siguientes 8 bits, para así saber con qué color se deberá pintar el pixel.

SEÑALES DE ENTRADA y SALIDA

Señales de entrada: PixelACK, DATA y DATA_READY

Señales de salida: RecACK, Pixel y Pixel_Rec.

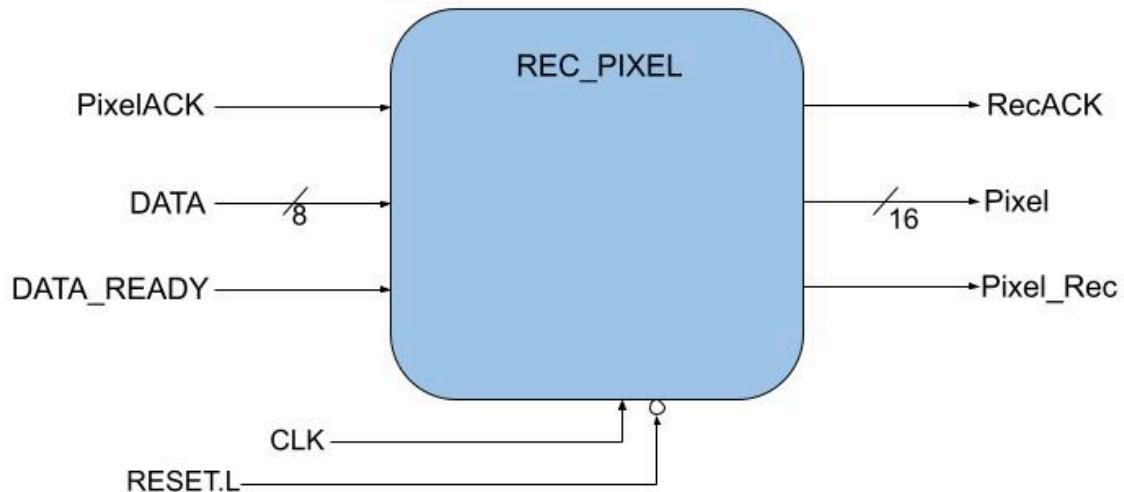


Figura 13: Diagrama del módulo REC_PIXEL

En la unidad de proceso se han utilizado los siguientes componentes:

- Dos registros. Uno para almacenar los primeros 8 bits y otro para los siguientes 8.

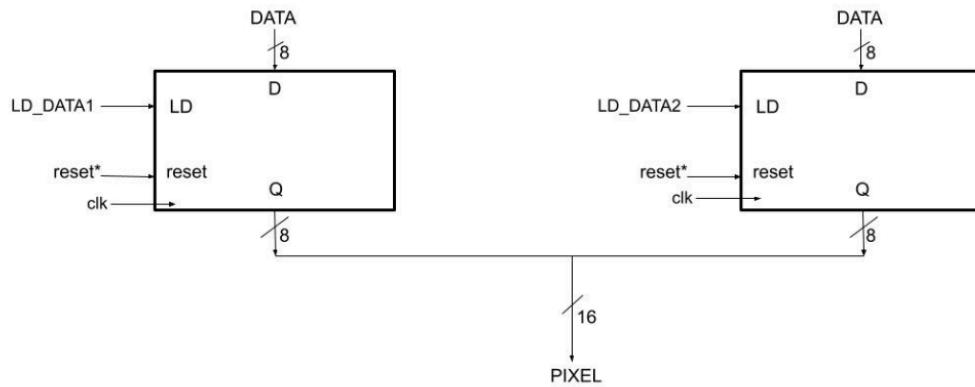


Figura 14 : UP REC_PIXEL

La unidad de control de REC_PIXEL comienza en el estado E0 al presionar reset. Cuando se active la señal DATA_READY, que indica que ya se han almacenado los 8 bits de la transmisión, de la UART, pasará al estado E1. En este estado cargará en el primer registro los 8 bits y activará la señal RecACK indicando a la UART que ha recibido los 8 bits. El siguiente estado es E2, en el que permaneceremos hasta que la señal DATA_READY vuelva a tener el valor 0, para asegurarnos de que no recibimos los mismos 8 bits nuevamente. Después, pasaremos a E3, donde esperaremos a que lleguen los nuevos 8 bits, que cargaremos en el otro registro en el estado E4. Finalmente, en el estado E5 esperaremos a la señal PixelACK que nos mandará LCD_DRAWING y volveremos al estado inicial E0.

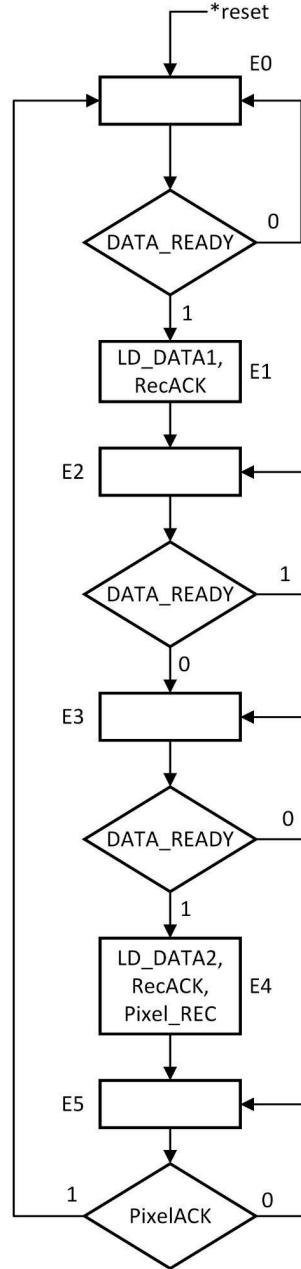


Figura 15: UC REC_PIXEL

Descripción VHDL: Comenzamos creando la entidad de REC_PIXEL y añadiendo los puertos de entrada y salida que podemos observar en la figura 13. A continuación, declaramos las y estados de REC_PIXEL, definiendo su arquitectura. También, hemos definido la unidad de control implementando el estado presente y el estado siguiente. Por último, hemos implementado tanto la lógica de activación de las señales como la unidad de proceso de REC_PIXEL, creando un proceso por cada componente.

Simulación: Para llevar a cabo la simulación del módulo REC_PIXEL hemos creado el testbench TB_REC_PIXEL. Para empezar, hemos creado la entidad tb_REC_PIXEL sin puertos, además de definir la arquitectura de la simulación, compuesta por el puerto y las señales anteriormente utilizadas en REC_PIXEL.vhd. Lo siguiente ha sido definir el DUT, en el que hemos conectado las señales con sus correspondientes puertos de REC_PIXEL. Después, hemos definido el ciclo de reloj de 20 ns, para finalmente dar los valores correspondientes a las diferentes señales, para poder observar el funcionamiento del módulo mediante el cronograma.

5. TEST Y VALIDACIÓN

Para hacer los test de nuestro diseño, hemos utilizado dos herramientas: ModelSim y Quartus. En el momento en el que finalizamos el archivo VHDL del LCD_CTRL o LCD_DRAWING, procedimos a hacer la simulación en el testbench del módulo correspondiente. Una vez implementados los testbench, ejecutamos la simulación y tras analizar el cronograma, comprobamos que había ido correctamente. En el caso del LCD_CTRL comparamos los cronogramas con las resoluciones de egela para asegurarnos de que estaban bien:

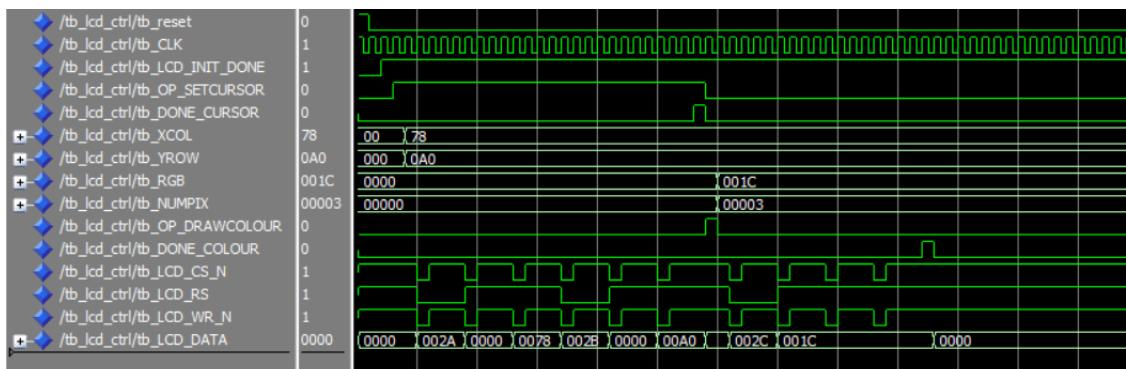


Figura 16: Cronograma LCD_CONTROL

Analizando el cronograma, podemos ver que primero se activa la señal LCD_INIT_DONE que es la que avisa de que la pantalla LT24 ya está disponible para ser usada. Después, se activa la señal OP_SERCURSOR, XCOL toma el valor '78' e YROW toma el valor '0A0'. Es importante saber que para posicionar el cursor en un punto determinado, hay que enviar cierta información por la salida LCD_DATA, más concretamente, algunos comandos y datos. Además, para enviar un comando o un dato, hacen falta 4 ciclos (80 ns) entre los cuales en el primero, habrá que activar las señales LCD_CS_N y LCD_WR_N (en lógica negativa), y desactivarlas en los tres ciclos siguientes; y la señal LCD_RS tiene que tomar el valor L durante los 4 ciclos, pero si es un parámetro del comando, tiene que valer H.

Si nos fijamos en el cronograma, podemos ver que estas señales siguen ese comportamiento ya que e LCD_CS_N y LCD_WR_N se activan durante un ciclo mientras que LCD_RS toma el valor L durante los 4 ciclos en los que se manda el comando 2A. Después, LCD_CS_N y LCD_WR_N se activan durante

un ciclo mientras que LCD_RS toma el valor L durante los 4 ciclos otra vez pero esta vez para mandar el comando 2B junto a los parámetros 0 y 0A0. Aquí lo que podemos deducir es que los comandos y datos para mover el cursor funcionan adecuadamente. Más adelante, RGB toma el valor 001C y NUMPIX toma el valor 3. Además, podemos ver que se manda el comando 2C (que es el comando para dibujar pixeles) junto al parámetro 001C (RGB) y se mantiene activo durante los 12 ciclos correspondientes, ya que al haber escogido 3 como NUMPIX, las señales LCD_CS_N, LCD_WR_N y LCD_RS tienen que activarse y desactivarse como anteriormente explicado (que tardarán 4 ciclos) 3 veces.

En el caso del LCD_DRAWING, también seguimos los mismos pasos, por lo que después de implementar y simular el testbench, nos aseguramos de que el cronograma era como el que esperábamos obtener:

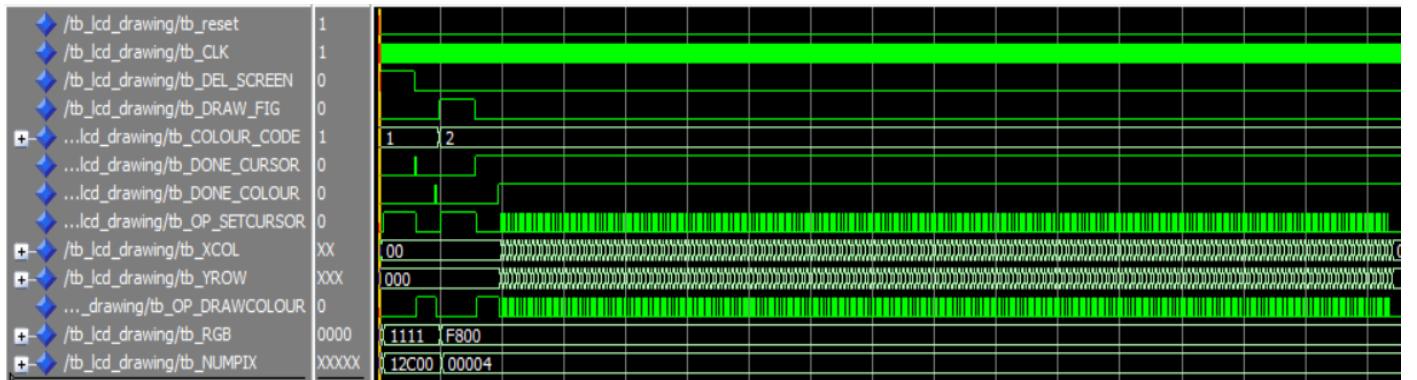


Figura 17: Cronograma LCD_DRAWING

A pesar de que en la imagen no se pueda ver correctamente, primero se simula la ejecución de borrar pantalla por lo que aprovechando que las señales han sido generadas manualmente y no desde el LCD_CTRL, hemos reducido el tiempo de espera entre el OP_DRAWCOLOUR y DONE_COLOUR. Por otro lado, en la operación de dibujar la línea hemos podido observar que funcionaba como esperábamos ya que después de que se active la señal que avisa de la operación observamos como los valores de X e Y van ascendiendo hasta llegar a los valores límite de la pantalla.

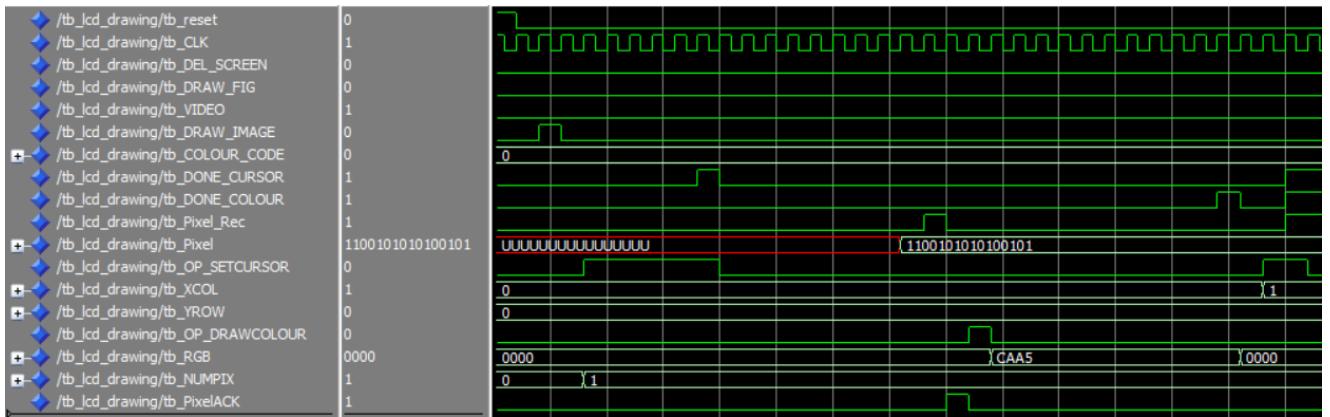


Figura 18: Cronograma LCD_DRAWING(2)

En este otro cronograma del LCD_DRAWING, podemos ver la ejecución de la funcionalidad de DRAW_IMAGE. Tal y como se ve en la imagen, primero se activa la señal DRAW_IMAGE para desactivarse un ciclo más tarde. Por otro lado podemos observar que X e Y tienen el valor 0 y 0 por lo que se esta dibujando el primer pixel. Además, vemos que se activa OP_SETCURSOR hasta que DONE_CURSOR se activa y más adelante vemos como se activa la señal que nos avisa de que el pixel está preparado, PIXEL_REC. Después, podemos ver como se activa la señal PixelACK lo cual es indicativo de que todo ha ido correctamente. Por otro lado, un ciclo más tarde podemos ver como se activa la señal OP_DRAWCOLOUR durante un ciclo hasta que se activa la señal DONE_COLOUR. Esto nos avisa de que la operación se ha completado con éxito. Unos ciclos más adelante, podemos ver como XCOL toma el valor 1 y se llama a OP_SETCURSOR por lo que queda claro que ha llegado a la siguiente iteración.

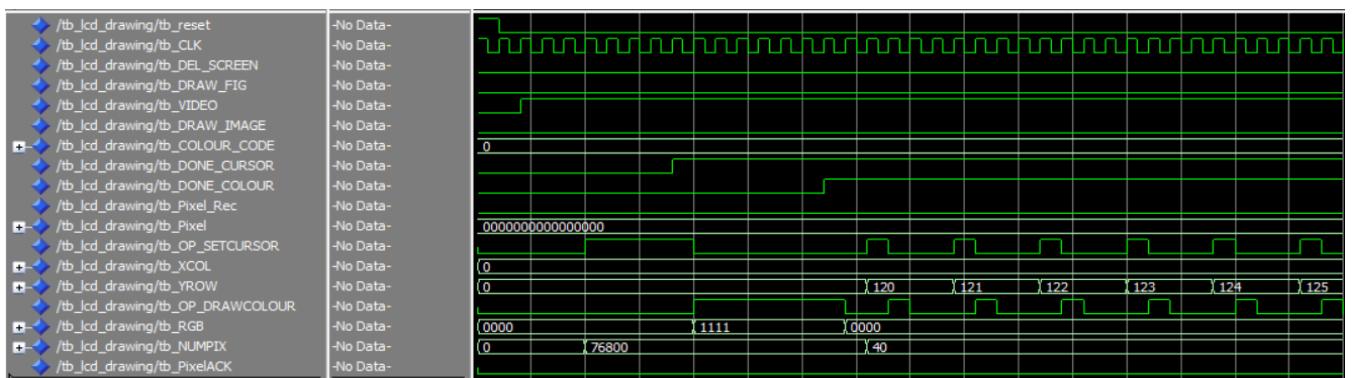


Figura 19: Cronograma VIDEO

Este cronograma muestra resumidamente la funcionalidad de VIDEO. Primero se activa la señal VIDEO, y después de que OP_SETCURSOR y DONE_CURSOR se activen, vemos que NUMPIX toma el valor de 76800 ya que al inicio de esta funcionalidad siempre se borra la pantalla. Más adelante, vemos como NUMPIX pasa a ser 40 lo cual es una buena señal ya que en el primer frame solo se dibuja la mitad del cuadrado. Además, después de que se borra la pantalla vemos que YROW toma un valor de 120 y se va incrementando lo cual es correcto. El problema con este cronograma es que para ver la funcionalidad real tendríamos que estar manualmente activando y desactivando las señales DONE_CURSOR y DONE_COLOUR cientos de miles de veces ya que el testbench solamente tiene el LCD_DRAWING y no el LCD_CTRL por lo que no es viable. Es por esto que dejamos estas señales activas y tal y como se ve en el siguiente cronograma, los valores que toman las variables son los que buscábamos a la hora de diseñar el módulo tal y como hemos explicado en el apartado de diseño del sistema.

En este otro cronograma, que también es de la funcionalidad de video, hemos ejecutado 34 millones de ns por lo que deberíamos ver como NUMPIX pasa de 40 a 80 y luego a 40 ya que en el primer y último frame solamente se dibuja la mitad del cuadrado y en el segundo frame el cuadrado entero, y es justamente lo que ocurre. Además vemos como YROW toma el valor 200 durante mucho tiempo, y esto es debido a que cuando va sumando de 120 a 200, en los 525000 ciclos de espera que hay que hacer para que el video vaya a los FPS deseado, YROW se queda en esa posición, no obstante, si hacemos zoom podemos ver como el valor de YROW se va sumando que es lo, que se veía en la imagen anterior.

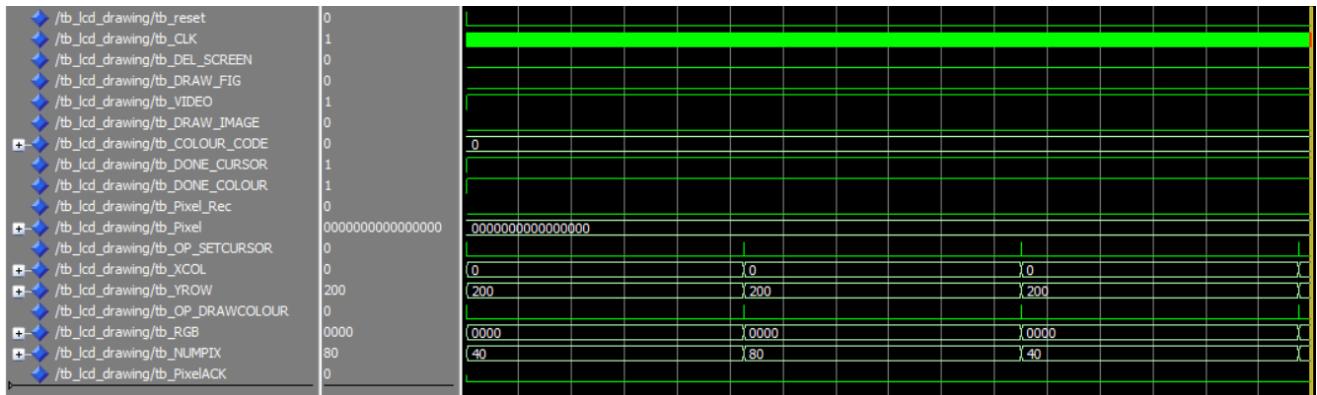


Figura 20: Cronograma VIDEO(2)

Por otro lado, tenemos la simulación del módulo UART. Podemos ver como RX al principio toma el valor 0. Cuando eso ocurre, aunque en la imagen no se aprecie bien ya que ocurre durante un breve instante, DESP toma el valor 11 y DATA se establece en 8 bits de 0. Por otro lado, vemos como se respetan los ciclos de espera que hay entre bit y bit que manda RX cuando despues de cada espera se activa FIN_WAIT brevemente y carga el dato nuevo. Además, se puede observar claramente como cada vez que hay que cargar un bit nuevo, (que aunque en esta imagen no se vea DESP toma el valor 01) DATA mete ese bit por la derecha y pierde el bit más significativo hasta llegar a los 8 bits. Cuando esto ocurre podemos ver que DATA_READY se activa correctamente. Por otro lado, podemos observar como Q_WAIT va reduciendo su valor de 1 en 1 hasta llegar a 1 aunque realmente baja hasta 0 ya que en 0 solamente está un breve instante antes de cargar el 8 de nuevo.

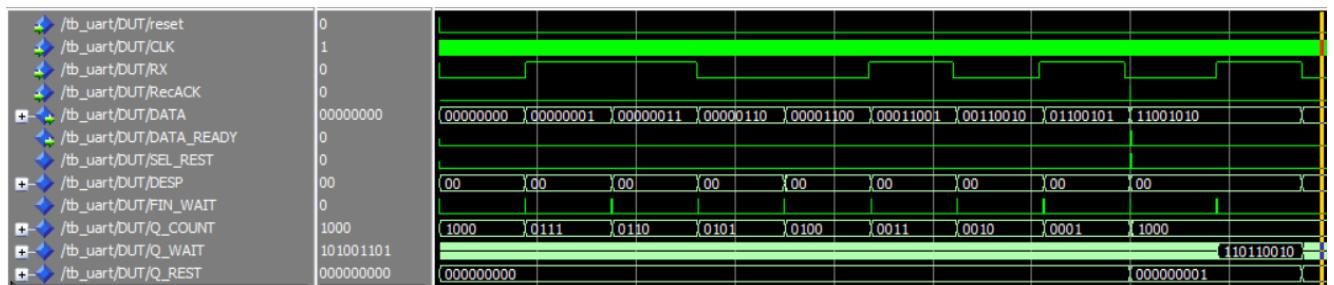


Figura 21: Cronograma UART

Para finalizar, tenemos el cronograma del módulo REC_PIXEL. Lo primero que vemos es que después de que DATA se complete, se activa la señal

DATA_READY que avisa al módulo de que el dato ya está disponible. Después de que esta señal se active, observamos cómo se activa el RecACK correctamente para avisar de que ya hemos recibido el dato y completar así el handshake. Después de esto, vemos como estos 8 bits de DATA se cargan correctamente en los 8 bits más significativos de Pixel. Más tarde, todo este proceso se repite una vez más cargando DATA correctamente en los 8 bits menos significativos de Pixel y vemos también que se activa la señal Pixel_Rec que avisa al LCD_DRAWING de que ya puede cargar el Pixel en RGB.

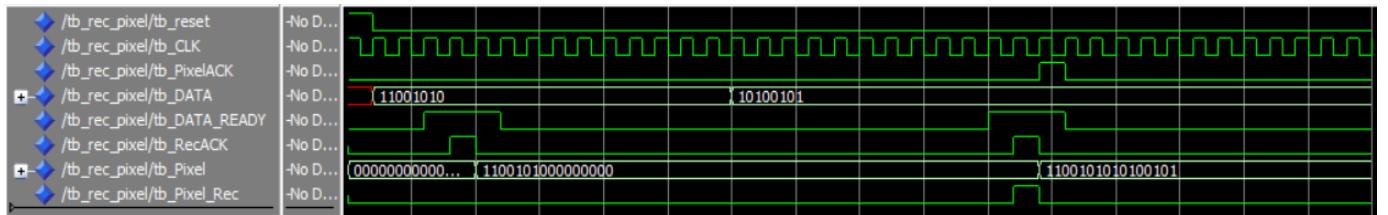


Figura 22: Cronograma REC_PIXEL

Los resultados de todos estos test se ven reflejados cuando se ejecuta el proyecto de Quartus ya que no nos da ningún error de compilación, tampoco generamos ningún latch, no da errores a la hora de programar la FPGA y al testearlos sobre la FPGA nos funcionan correctamente.

6. MANUAL DE USUARIO

Guía de uso:

Enchufar el cable de alimentación de la FPGA a la corriente.

Cargar programa en la FPGA .

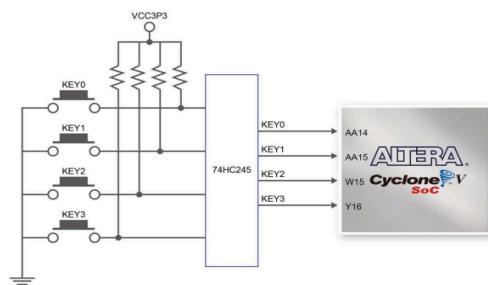


Figura 23: Esquema botones De1-Soc

-Seleccionar acción que se desea realizar mediante los botones:

Boton0: Reiniciar

Boton1: Dibujar imagen

Boton2: Borrar Pantalla

Switch9 = 0 + Boton3: Dibujar línea

Switch9 = 1 + Boton3: Video



Figura 24: Esquema switches De1-Soc

-Mandar imagen mediante ordenador, para la función de dibujar imagen:

Abrir programa ‘Tera Term’.

Escoger puerto (normalmente COM3).

Pulsar ‘File’ → ‘Send’ y escoger la imagen que se desea enviar.

Importante: Antes de enviar la imagen, seleccionar la casilla ‘binary’.

-Seleccionar color mediante switches:

Número para cada color:

000 → Negro

001 → Blanco

010 → Rojo

011 → Verde

100 → Azul

101 → Marrón

110 → Naranja

111 → Rosa

7. CONCLUSIONES Y PROPUESTAS DE POSIBLES MEJORAS

En conclusión, mediante este proyecto hemos aprendido cómo diseñar sistemas digitales de un nivel de complejidad superior a los vistos hasta ahora en otras asignaturas. Para ello, hemos diseñado un sistema dividido en varios módulos, que da solución al problema que se nos ha presentado, el de dibujar una línea diagonal en la pantalla, y borrar la pantalla.

Además, hemos utilizado por primera vez elementos físicos para probar nuestros diseños, es decir, no nos hemos limitado a simular mediante programas el funcionamiento de nuestras creaciones, sino que hemos podido, en este caso mediante la FPGA DE-1 SoC y la pantalla LT24, ver de verdad en qué hemos invertido nuestro tiempo y esfuerzo.

Por si fuera poco, también hemos tenido que aprender a utilizar un nuevo entorno de programación con nuevas características para poder modelar los componentes utilizados en nuestro diseño. El lenguaje, ‘VHDL’ ha supuesto otro pequeño reto para nosotros que hemos tenido que superar mediante aprendizaje y prueba y error, dando solución a los diferentes problemas que iban apareciendo a medida que escribíamos el código.

Finalmente, como conclusión y como posible mejora está el trabajo en grupo, ya que hemos concluido que el reparto de tareas y una buena gestión del grupo son vitales a la hora de realizar tareas de esta magnitud. Es por ello que como mejora deberíamos haber realizado un reparto eficiente de los trabajos a realizar desde el principio, para así poder optimizar el tiempo que hemos pasado con este proyecto.

8. BIBLIOGRAFÍA

- [1] Olatz Arbelaitz Gallego, Txelo Ruiz Vazquez, Olatz Arregi Uriarte, Agustin Arruabarrena Frutos, Izaskun Etxeberria Uztarroz, Amaia Ibarra Lasa (2005). Sistema digitalen diseinu-hastapenak: Oinarrizko kontzeptuak eta adibideak. UEU.
- [2] Guía de los programas Quartus Prime y ModelSim (parte I y parte II).
- [3] Facultad Informática SS. Guía rápida de uso de la LCD LT24.
- [4] Diapositivas y demás documentos de ‘e-gela’.

9. APÉNDICES

Aquí se encuentran todos los documentos e información relacionada con el proyecto:

<https://github.com/gorkadaboi/DCSD/tree/main>

10. TABLA DE DEDICACIÓN SEMANAL

	HORAS DEDICADAS
SEMANA 1	2
SEMANA 2	2
SEMANA 3	2
SEMANA 4	4
SEMANA 5	4
SEMANA 6	4
SEMANA 7	4
SEMANA 8	6
SEMANA 9	2
SEMANA 10	2
SEMANA 11	2
SEMANA 12	2
SEMANA 13	20
SEMANA 14	2
SEMANA 15	4
SEMANA 16	4

**Enero (Tiempo dedicado a la memoria)

SEMANA 17	4
SEMANA 18	8