UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN MENCIÓN EN SISTEMAS ELECTRÓNICOS

DISEÑO DE UN SISTEMA ELECTRÓNICO MEDIANTE FPGA PARA LA SIMULACIÓN DE VIDEOJUEGOS

Autor:

Ángel Díaz Puerto

Tutor:

Jesús Arias Álvarez

Agradecimientos

En primer lugar, tengo que agradecérselo a toda mi familia que es la que me ha dado la oportunidad de haber llegado hoy en día hasta aquí, por haberme cedido siempre el hombro para apoyarme en él cuando lo necesitaba y que para mí siempre han sido un ejemplo de esfuerzo y dedicación a seguir por supuesto.

En segundo lugar, a mi tutor Jesús Arias Álvarez por toda su sabiduría que me ha transmitido durante estos últimos años de carrera y por haberme guiado y ayudado tanto a lo largo del desarrollo de este proyecto.

A la ETSIT Uva por haberme brindado unos profesores excelentes, en especial los del departamento de electrónica por enriquecer cada día más el conocimiento sobre esta materia que cada día me gusta más.

Finalmente, y no por ello menos importante a todos mis amigos por todos los momentos buenos y malos que hemos pasado juntos y que seguiremos pasando día tras día y especialmente a Paula por ayudarme tanto en mi vida como lo hace.

Resumen

Sin lugar a dudas, la intervención de los videojuegos en las vidas humanas ha sido el causante de increíbles desarrollos en el ámbito de las telecomunicaciones y la electrónica, buscando siempre ese fin colectivo de enriquecernos mediante el ocio y aprendiendo de él poco a poco.

Debido a esto se desarrolla un equipo que sea capaz de desarrollar las capacidades de simulación de videojuegos de los años 80-90, con unas características similares a las de la época, haciendo que funcionen como deben principalmente los componentes que definen una videoconsola, que son en nuestro caso la FPGA, la interfaz de video con el VGA, la interfaz de DB9 para el joystick y finalmente la memoria RAM.

Índice general

Índice de figuras

1. Introducción

1.1 Objetivos del proyecto

El ocio y el entretenimiento siempre ha estado en la mente del ser humano, por esto es por lo que la electrónica también se ha ramificado por este camino para darnos un mayor número de herramientas con las que poder disfrutar y pasar un rato agradable. El objetivo de este proyecto se basa principalmente en esta necesidad de recrear un dispositivo que nos permita la simulación de videojuegos mediante la utilización de una FPGA como corazón del proyecto.

Gracias a los conocimientos adquiridos a lo largo de estos años de estudio en el grado se pretende diseñar un dispositivo que se asemeje a computadoras de los años 80-90 que permitían la simulación de videojuegos de la época. Nos basaremos en el dispositivo "Sinclair ZX Spectrum", que se trataba de un ordenador de 8 bits basado en el microprocesador Zilog Z80A.

1.2 Fases del proyecto

A continuación, para una mejor comprensión de como se ha llevado a cabo el proyecto, este se ha dividido fundamentalmente en los siguientes apartados, que engloban los aspectos más fundamentales del mismo:

1. Introducción
2. Especificaciones técnicas del proyecto
3. Desarrollo Hardware: diseño y construcción de la placa del dispositivo
4. Desarrollo Software: comprobación del correcto funcionamiento de las partes que lo componen.
5. Herramientas para la evaluación del producto.
6. Resultados obtenidos.
7. Documentación utilizada.

1.3 Visión general

Introducción: observando este apartado realizado nos muestra el objetivo principal del proyecto realizado y son descritos brevemente los puntos más importantes que lo definen.

Especificaciones técnicas del proyecto: en este punto se establecen una serie de retos que son necesarios que cumpla nuestro diseño, junto con las características que nosotros queremos que posea. Se comienza por realizar un estudio de mercado para fijarnos en cómo ha sido diseñado y poder replicar de alguna manera, los aspectos que sean más necesarios para nuestro diseño y los demás son desechados.

Desarrollo Hardware: siguiendo con el plan trazado en el punto anterior, comenzamos pensando varios diseños que puedan adaptarse correctamente a las necesidades que tenemos con el producto. Tras prueba y error, damos con el diseño final del proyecto y obtenemos un esquemático final con el que desarrollaremos el posterior archivo de la huella del dispositivo, con el que encargaremos la producción de la PCB. Por supuesto, en este punto, al tener finalizado el esquemático y la huella, tratamos la elección correcta de los valores de los componentes, el montaje de estos en la PCB y la verificación de que todo funciona como se espera.

Desarrollo Software: en este apartado se describe el proceso de realización de los programas que nos permiten comprobar el correcto funcionamiento de los 3 apartados fundamentales de nuestro diseño, que son: interfaz VGA para las señales de video, interfaz PS2 para las señales que envía el teclado que será utilizado con el diseño y finalmente la parte de la memoria RAM externa que hemos añadido al producto. Estos programas han sido diseñados en lenguaje de Verilog, que es la herramienta de descripción de hardware utilizada para esta familia de FPGAs.

Herramientas para la evaluación del producto: utilizando el programa de visor de onda Gtkwave y el simulador de iVerilog para una primera visualización en la máquina virtual y una vez obtenida una correcta simulación, pasar a realizar el proceso de síntesis con nuestra FPGA y poder comprobarlo que funciona todo correctamente en el producto. Las herramientas utilizadas para la síntesis será el programa yosys y el programa nextpnr (place and route), que se encarga de routar a los pines correspondientes de la FPGA para que todo quede unido donde le corresponde.

Resultados obtenidos: en este apartado se realizan una serie de pruebas utilizando el procedimiento de los programas del apartado anterior para comprobar si se ha logrado cumplir

los objetivos impuestos para el producto. Además, se muestran una serie de conclusiones para una supuesta finalización del producto.

Documentación utilizada: para finalizar, en este capítulo se muestra toda la bibliografía empleada en el desarrollo del proyecto junto con el código desarrollado para el funcionamiento de la placa.

2. Especificaciones técnicas del proyecto

2.1 Estudio de mercado

Una vez se consigue la idea del producto que se quiere diseñar, se ha realizado una búsqueda de equipos similares que permitan la simulación de videojuegos, que es la tarea que se espera que desempeñe nuestro equipo electrónico.

En el ámbito comercial existen una serie de equipos electrónicos a la venta con características similares, aunque un poco más escasas que nuestro dispositivo debido a que la tecnología avanza con los años y los componentes existentes en la actualidad, aunque se ha intentado que se asemejen lo más posible suelen ofrecer unas mejores prestaciones.

En primer lugar, tenemos el *Apple II* que fue diseñado en los años 70, tenía como corazón un procesador 6502 de MOS Technology que trabajaba alrededor de 1MHz, 4kB de memoria RAM y permitía enviar la señal de video a un monitor en color (de la misma manera que lo hace nuestro dispositivo) pero este utilizaba un modulador de radio-frecuencia.

Ilustración 1. Familia de computadores Apple II

En segundo lugar, como producto más destacado de la época tenemos el *Atari 800*, que este encontró su gran éxito debido al impacto que tuvo la consola 2600. Esta computadora de 8 bits, integraba el procesador 6502B pero que corría a una velocidad de 1.7MHz, poseía una memoria RAM de 16kB.

Ilustración 2. Familia de computadores Atari 800.

Estos dos fueron los ejemplos que más impacto han tenido en nuestra búsqueda de un productor similar, aunque finalmente nos hemos basado en el *ZX Spectrum,* que fue diseñado en la década de los 80. Aunque este ordenador de 8 bits no fue el de mejor hardware, era el que más se asemejaba a nuestras necesidades. Este corría un procesador Z80A de Zilog con una velocidad del reloj de 3.5MHz (como podemos observar, este permitía un mayor número de operaciones en su procesador que la competencia que tenían una menor velocidad en sus relojes) con una memoria RAM de entre 16 y 128kB según la versión del equipo que se escogiera. Una de las peculiaridades del sistema gráfica del Spectrum es que era capaz de mostrar una matriz de 256x192 pixeles, es decir 1 bit por pixel, pero con colores de gráficos y fondo seleccionables para cada bloque de 8x8 pixeles de la pantalla.

Ilustración 3. Familia de computadores Sinclair ZX Spectrum.

Cogiendo ideas de estos dispositivos especialmente del último nos fijamos que como señales de audio estos tenían conectores de entrada/salida de 3.5mm, aunque nosotros hemos optado por un altavoz que nos parecía una mejor idea para nuestro diseño. Como estos ordenadores venían con un teclado integrado en su placa, se ha optado por añadir un conector PS2 para poder conectarle un teclado con el que poder comunicarnos con la computadora. Finalmente optamos por añadir un conector DB9 debido a que todos estos productos eran capaces de admitir joystick para poder jugar de una manera más sencilla a los videojuegos cargados en sus memorias. Para representar la señal de video en un monitor al que conectarlo hemos optado por una interfaz VGA, debido a la disponibilidad de monitores, aunque esto supone duplicar la frecuencia de barrido horizontal de la señal de video.

2.2 Estudio de especificaciones

De igual manera que si una empresa nos pidiera desarrollar un producto para ellos, deben ponerse sobre la mesa una serie de especificaciones que son necesarias que posea nuestro producto. Algunas especificaciones las hemos nombrado en apartados anteriores, pero ahora se van a comentar de una manera más específica y detallada.

Comenzamos en primer lugar, el equipo debe ser capaz de correr videojuegos de 8 bits de manera fluida y sin ninguna interrupción en su ejecución. Debido a la complejidad del dispositivo, nosotros lo que queremos conseguir es que el producto sea capaz de pasar 3 pruebas que pasaremos a diseñar más adelante vía software para que, en primer lugar, nuestro equipo deba reconocer las señales que le mandamos desde las distintas partes de la placa, específicamente de nuestra FPGA y sepa capaz de “dibujar” correctamente en un monitor lo que nosotros le estamos indicando que pinte, que como veremos será la impresión de una serie de barras de colores y algunas con su respectivo degradado para que sepamos que funciona correctamente lo que le mandamos.

Debido a que el diseño de nuestro producto debe ser lo suficientemente robusto, la segunda prueba que debe pasar, es que a la hora de conectarle un teclado mediante la interfaz del PS2, sepa reconocer correctamente las teclas que el usuario está pulsando y esas teclas pulsadas puedan ser guardadas para su posterior uso. Para tener una realimentación de que nuestro teclado funciona de una manera correcta, realizaremos una unión entre esta prueba y la anterior para así poder tener una realimentación de que nuestro dispositivo está funcionando de la manera que se pretende. Para ello, utilizaremos el valor del código de barrido (no es el valor ASCII de la tecla) de la tecla que estamos pulsando para hacérsela pasar a la interfaz VGA, que

en el código de colores que nosotros elijamos nos pinte el código binario de la tecla que estamos pulsando.

Finalmente, la última especificación necesaria de nuestro prototipo está relacionada con la memoria RAM. Lo que queremos conseguir de alguna manera es que escribamos un dato en ella y posteriormente seamos capaz de leer su valor mediante la dirección de memoria, comprobando que no ha cambiado su valor y sea igual que el valor escrito en esta. La realimentación que podemos tener en esta prueba será similar a la anterior, imprimiendo en la pantalla el dato que vayamos a escribir en un código de color, por ejemplo, en el rojo y luego a la hora de leer el dato de la memoria RAM, imprimirlo en otro color por ejemplo el verde y comprobarla que los códigos de colores son los mismos para ver que, ese valor no se ha visto alterado al pasar por la memoria.

Además de estas especificaciones que son las fundamentales que nuestro prototipo debe tener, por otro lado, tiene un conector DB9 que deberá ser capaz de interactuar con un joystick que será conectado a ese puerto. Por otro lado, los videojuegos que queramos utilizar en nuestro producto, serán cargados mediante una tarjeta SD, por lo que el diseño tendrá un lector de tarjetas SD y transmitir la información a las distintas partes del diseño.

Para terminar, es necesario que nuestro producto tenga audio para introducir al usuario en una experiencia más enriquecedora de juego, luego además se deberá añadir un altavoz con su respectivo potenciómetro para que el volumen sea perfectamente ajustable a las necesidades de cada uno. Una prueba que se podría hacer sería la de generar un tono, generando una señal de forma cuadrada y con ello sería suficiente.

2.3 Estudio del sistema y selección de los componentes

En este apartado, una vez que se han establecido las especificaciones que queremos que nuestro diseño posea, luego ahora pasamos a desarrollar el hardware que sea capaz de llevarlas a cabo, mediante la elección correcta de los componentes concretos, además de los componentes pasivos necesarios para darnos los correctos valores de tensión en las distintas zonas de nuestro diseño.

2.3.1 FPGA

Como “cerebro” que se encargue de manejar todo el funcionamiento y las conexiones de los componentes ha sido escogido el componente con nombre “fpga lattice ice40hx4k”.

Esta pequeña, pero a la vez potente pieza ha sido diseñada por la empresa lattice y está unificada por un total de 7680 celdas lógicas, notar que el fabricante nos indica que posee 3520 (menos de la mitad de las que realmente posee) y 20 bloques de memoria RAM cuando en realidad tiene 32. Además, está formada por una memoria integrada de unos 128kbits. Este se encuentra alimentado a una tensión de unos 3.3V y 1.2V que obtendremos gracias a nuestros reguladores de tensión, debido a que nosotros obtenemos la alimentación a través de un conector USB consiguiendo un voltaje de 5V. La frecuencia máxima a la que puede operar esta FPGA es variable y depende del diseño que se realice, pero una señal de reloj de 30MHz se puede conseguir de manera muy sencilla.

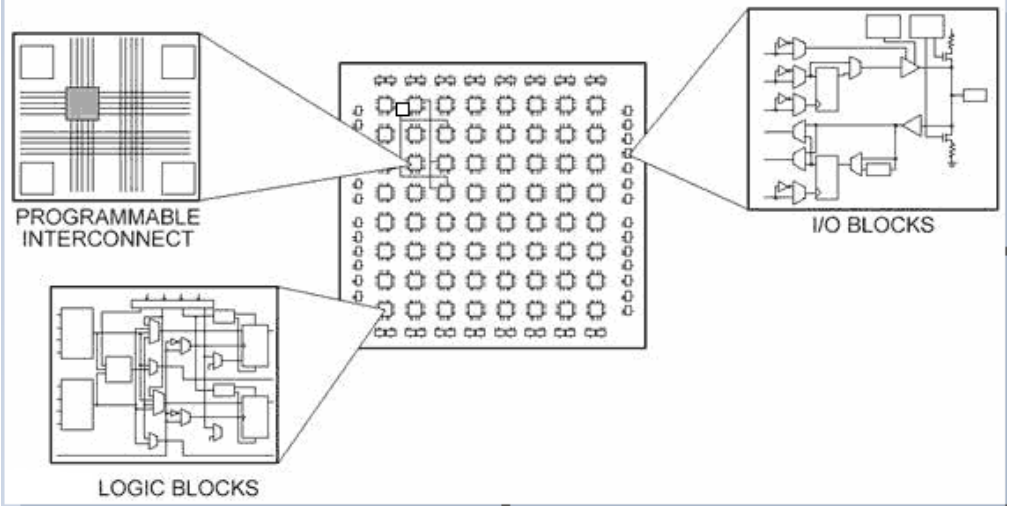


Ilustración 4. Distintas implementaciones dentro de las FPGAs.

Para programar este cerebro utilizaremos la comunicación mediante un conector de programación que hemos diseñado para nuestra placa que posee 11 pines y que conecta con una placa externa que permite la configuración de la FPGA y a la vez la programación de una memoria flash no volátil mediante un bus SPI.

Motivación para la elección del modelo frente a un micro: herramientas de dominio público, hablar también sobre los prototipos previos como la ICE40HX1K (que posee pocas celdas). Por otro lado, como principales ventajas de la elección de este dispositivo frente a la opción de escoger un microcontrolador, es que los microcontroladores usan procesos secuenciales, es decir que pueden ejecutar únicamente una tarea a la vez, mientras que las FPGA pueden

ejecutar varios procesos en paralelo. Debido a esto, son la mejor opción para aplicaciones con señales de alta velocidad o que se procesan en tiempo real.

Otra diferencia respecto a los microprocesadores, es que estos se programar con un lenguaje como ensamblaje como son C o C++, pero en cambio los FPGA, como se trata de un circuito en blanco con miles de puertas lógicas esperando a ser configuradas para crear otros circuitos, utilizamos un lenguaje de descripción de hardware o HDL como puede ser Verilog o VHDL. Por lo que podemos aclarar que las FPGA pueden ser reprogramadas para cambiar completamente la funcionalidad que poseen, en cambio los microprocesadores no se pueden debido a que ya tienen una circuitería y un conjunto de instrucciones estático para un conjunto de aplicaciones concretas.

2.3.2 Memorias

En primer lugar, expondremos información relevante acerca de nuestra memoria flash SPI N25Q64, como por ejemplo que tiene una capacidad máxima de unos 8MB con un tamaño de palabra de 8 bits. El SPI posee una velocidad normal de lectura de unos 50MHz y tiene una opción denominada “fast read” o de rápida lectura en la cual su velocidad de lectura puede ascender hasta la cifra de 133MHz. En esta memoria es donde almacenaremos datos de configuración de la FPGA, pues esta es capaz de leer su contenido de forma automática tras un reset o power-on. Además, podemos dar uso como almacenamiento de datos no volátil a la memoria sobrante.

En segundo lugar, contaremos con una memoria RAM (Random Access Memory), aquí es donde se cargan todas las instrucciones que ejecuta la unidad central de procesamiento o CPU emulada en la FPGA. Se denominan de acceso aleatorio porque se puede leer o escribir en una posición de memoria con un tiempo de espera equitativo para cualquier posición, no es necesario hacerlo de manera secuencial para acceder a la información de una manera más rápida.

Esta tiene un tamaño de 1Mbit, con una organización distribuida en 128k x 8 bits o también podría ser 64k x 16 bits, es decir un número de palabras de 128k con un número de 8 bits por palabra o 64k como número de palabras con un tamaño de 16 bits. El bus de direcciones posee un ancho de 16 bit y la frecuencia del reloj alcanza una tasa de 100MHz, consiguiendo unos tiempos de acceso aleatorios máximos de aproximadamente unos 10ns. Esta memoria posee una tensión de alimentación a 3.3V.

2.3.3 Alimentación

Como núcleo de nuestra alimentación tenemos un conector micro-USB con el que conseguiremos los 5V que posteriormente mediante unos reguladores de tensión obtendremos las distintas tensiones nominales que necesitan algunos componentes de nuestro prototipo, que son por ejemplo 3.3V y 1.2V.

El micro USB únicamente está instalado como fuente de alimentación de nuestro dispositivo, sin ninguna utilidad más que esa (no sirve como puerto de comunicaciones, a no ser que se programe un controlador USB en la FPGA). Por otro lado, tenemos un primer regulador de tensión LD1117S33CTR, conectado a la alimentación del USB, que este nos ofrece un voltaje de alimentación en su salida de 3.3V, con el que alimentaremos nuestra FPGA, conector de tarjeta SD, memoria RAM conector de programación, oscilador del cristal de cuarzo y el amplificador para el altavoz. y una corriente de salida máxima de unos 500mA (es el máximo en la conexión USB). A continuación, tenemos conectado a la salida otro regulador de tensión MCP1725-ADJE/SN, con el que obtenemos un voltaje de salida de 1.2V. Estos 1.2V serán utilizados para alimentar el núcleo de nuestra FPGA. Por otro lado, en la placa de programación se ha instalado un conector mini USB para poder alimentar esta placa, que este proporciona también un voltaje de 5V.

2.3.4 Reloj de cristal de cuarzo

En nuestro prototipo, se ha optado por un diseño de un oscilador con un cristal de cuarzo de 16MHz, el modelo concreto es el 9B-16.000MAAJ-B. Se ha elegido esta frecuencia debido a la versatilidad del valor de la frecuencia, debido a que posteriormente mediante el firmware realizado para las distintas aplicaciones de nuestra aplicación, va a ser necesario la implantación de un PLL que se encargue de darnos frecuencias de trabajo mayores o menores dependiendo de la aplicación.

Este tipo de osciladores de cristal se caracterizan por la estabilidad de frecuencia y pureza de fase que presentan dada por el resonador, en realidad estos vibran mecánicamente, aunque debido a las propiedades características del cuarzo como son la piezoelectricidad y la vibración mecánica genera una vibración eléctrica y viceversa. Estos pueden ser utilizados como filtros de frecuencia, aunque la aplicación en nuestro prototipo es la de oscilador.

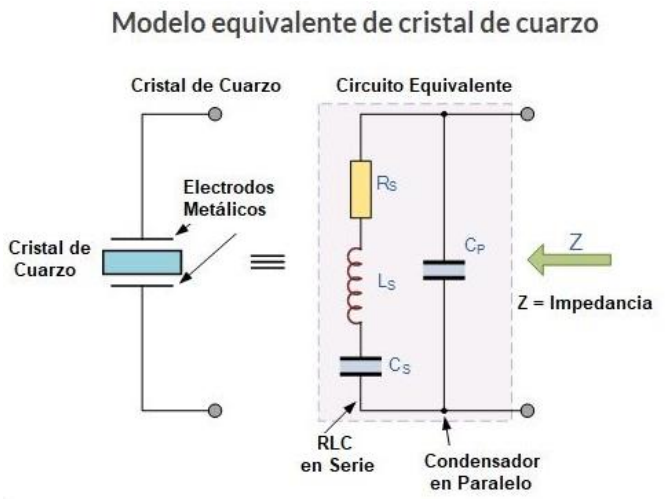


Ilustración 5. Modelo equivalente esquemático de un cristal de cuarzo.

En nuestro diseño, hemos implementado un oscilador de tipo Colpitts, con el transistor Q1 en configuración de colector común, que genera una onda senoidal en el pin 21 de la FPGA y un Schmitt-trigger interno convierte esta onda en una señal cuadrada.

2.3.5 DAC

Teniendo en cuenta que la señal de video debe ser representada de manera analógica a través de la interfaz del VGA, debemos realizar una previa conversión digital-analógica debido a que las señales que salen de los pines de la propia FPGA se tratan de pulsos digitales. Tras analizar los distintos DAC que existen en el mercado, a la hora del desarrollo de nuestro prototipo se pensó que no era necesario instaurar ningún DAC sofisticado que elevara el precio del producto sin necesidad, por lo que se optó por diseñar 3 DAC distintos, 1 para cada salida de cada color, es decir 1 para el rojo, otro para el verde y otro para el azul.

En nuestro caso la conmutación la hacen los pines de la FPGA siendo Vref = 3.3V.

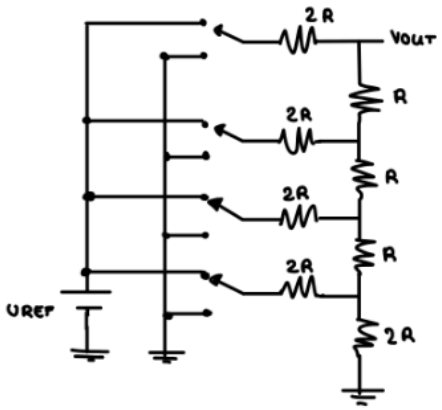


Ilustración 6. Representación esquemática de una red R-2R.

Estos DAC consisten en unas redes de resistencias, también se denominan redes R-2R o escalera de resistencias. Esto permite de una forma simple y económica implementar un DAC alternando los 2 posibles valores de resistencias en una escalera. Las ventajas de esta implementación es que posee una alta velocidad de conversión de los datos digitales y que posee un funcionamiento sencillo, sin tener que invertir demasiado presupuesto, únicamente el coste de las resistencias que los conforman, aunque sea elevado el número de estas, sigue saliendo mucho más barato que una implementación con DAC integrado.

2.3.6 Potenciómetro

Este componente pasivo ha sido instaurado en nuestro prototipo con el fin de controlar las corrientes que circulan por el circuito que se encarga de la generación del audio. En definitiva, variando la resistividad de este componente conseguimos que se aumente o disminuya el volumen que emite el altavoz incorporado en el producto.

Este componente, que se utiliza como resistencia variable, cuenta con una resistividad de 100Ω, además posee un orificio pasante en el que podremos insertarle una rueda con la que se nos facilitará la tarea de modificar su resistividad.

2.3.7 Interfaces

Una interfaz se trata del método a través del cual la FPGA envía o recibe datos del exterior. Pasaremos a analizar una a una las distintas interfaces que hemos utilizada en el diseño de nuestro dispositivo, ya que tenemos interfaces de video, audio, etc y son fundamentales en la correcta implementación del producto.

2.3.7.1 VGA

VGA “Video Graphics Array” o matriz de gráficos de video, posee las siguientes características:

- La resolución que genera en una pantalla es de 640x480 píxeles, aunque realmente el máximo de píxeles horizontales es de 800 y los píxeles verticales tienen un máximo de 600.
- El conector posee 15 contactos de tipo D subminiatura.
- La señal que es emitida a través de estos pines es de tipo analógica, luego como hemos comentado en apartados anteriores es necesaria esa conversión digital-analógica para que funcione de manera correcta.
- El modo estándar de gráficos o modo plano posee un máximo de 2^{12}=4096 colores, sin embargo, el uso de 12 bits/pixel implicaría una cantidad de memoria excesiva, y además los sistemas emulados no suelen tener tanta resolución, limitándose típicamente a un máximo de 4 bits/pixel.
- A la salida de cada DAC tenemos una resistencia que denominamos R2 que junto con la resistencia de salida del DAC, se encargan de atenuar de 3.3V a 0.7V que es el nivel máximo de las señales de video, aunque también debemos tener en cuenta la resistencia de terminación de 75Ω del monitor.

Ilustración 7. Conector VGA hembra.

2.3.7.2 Teclado PS2

El conector PS2 o puerto PS2 toma su nombre de la serie de computadoras correspondiente a IBM Personal System/2 creada por IBM y empleada para la conexión de teclados y ratones. Se trata en ambos casos de un puerto serial, pero en el caso del teclado es bidireccional (aunque la comunicación PC -> teclado únicamente se usa para encender/apagar los LEDs del teclado), que es el periférico que controlaremos nosotros en nuestro producto, ya que en este diseño solo se ha previsto la comunicación teclado -> FPGA. Para permitir esa comunicación bidireccional en el teclado es necesario en ambos lados un colector abierto. Las computadoras normales de sobremesa no son capaces de identificar el teclado y el ratón si se intercambian las posiciones de ambos.

La interfaz consta de sólo 2 señales, datos y reloj. La señal de datos de la conexión PS2 tiene una frecuencia de reloj de entre 10kHz-16kHz, enviando tramas de una longitud de 11 bits, en los que el primer bit se trata de el bit de comienzo, los 8 siguientes bits se tratan de los bits de datos, correspondientes a la tecla pulsada, el siguiente bit se trata del bit de paridad impar y por último 1 bit de parada.

Ilustración 8. Interfaces PS2 de teclado y ratón.

2.3.7.3 Almacenamiento

El almacenamiento es una parte importante de nuestro producto y es donde se guardarán posteriormente los juegos que se quieran utilizar en nuestro producto. Se ha instaurado un lector de tarjetas SD para que haya espacio suficiente para cargar los juegos que nos permita el almacenamiento de la tarjeta que se compre (dependerá del usuario en cuestión).

Una tarjeta SD o digital segura se trata de una memoria flash extraíble utilizada para almacenar información digital, como programas y archivos. Las tarjetas SD como ya hemos mencionado se utilizan para ampliar el almacenamiento de la memoria ROM de nuestros dispositivos. La velocidad de lectura y escritura de este tipo de tarjetas es de 12.5MB/s, pero estas velocidades pueden aumentar en gran medida dependiendo del tipo de tarjeta que escojamos.

Adicionalmente, la propia Flash SPI se puede usar con este fin pues la configuración de la FPGA ocupa tan solo unos 130kB de los 8192kB de la memoria disponible.

2.3.7.4 Audio

La interfaz de audio que es utilizada en nuestro dispositivo se trata un de altavoz. La señal que llega al altavoz será previamente amplificada para que esta sea lo suficientemente potente para que sea audible. Hemos instaurado un potenciómetro que como ya se ha comentado en apartados anteriores, este se trata de una resistencia variable que nos facilita la tarea de aumentar o disminuir el volumen del altavoz.

Esta es una señal digital que nos permite la generación de tonos con formas de onda rectangulares, pero también serviría, como salida de audio multinivel (DAC) si se implementa una modulación PWM.

2.3.7.5 Joystick

Debido a que se trata de un producto que se utilizará para jugar a videojuegos, es necesario que se pueda conectar un joystick a la placa para facilitar la experiencia de juego del usuario.

El joystick que sería compatible sería el atari 2600, que únicamente posee las 4 posiciones del joystick y un botón. Por otro lado, debería ser testado la posibilidad de una compatibilidad con un joystick de Nintendo NES que posee un mayor número de botones en concreto son 8, ya que este joystick está basado en el circuito integrado CD4021 e incorpora un poco de lógica para transmitir la información del estado de los pulsadores en serie.

Ilustración 9. Conector para joystick.

2.3.8 Componentes pasivos

Además de los componentes principales mencionados anteriormente, se han empleado otros cuantos componentes como pueden ser resistencias, condensadores y diodos, los cuales han sido empleados para diferentes tipos de tareas, como pueden ser los condensadores para acoplos/desacoplos en la alimentación, las resistencias como DAC de video y divisores de tensión y los diodos se han utilizado como fijadores de tensión en algunas partes del diseño, para que en algunas zonas críticas la tensión no pueda sobrepasar de un cierto valor para no dañar algún componente o que se confunda a la hora de leer algún valor lógico. Cabe destacar que el valor de algunos condensadores o resistencias han sido modificados debido a la escasez de componentes, pero esta elección ha sido realizada en zonas del diseño donde no era crítica una ligera modificación de la resistencia o condensador empleado.

3. Desarrollo Hardware

3.1 Diseño diagrama de bloques

Para la realización del diseño del hardware de nuestro producto se ha utilizada la herramienta de software Proteus (versión 8.11), con algunas de sus principales utilidades como pueden ser el diseño y creación del esquemático de nuestro proyecto, la generación de nuevos componentes que no están disponibles dentro de las propias librerías del programa mencionado y finalmente una vez terminado el esquemático, la creación del listado de los materiales que conforman nuestro proyecto o también denominado "*Bill of materials" (BOM).*

Este esquemático está formado por todos los componentes que permiten el correcto funcionamiento del prototipo. Por lo que ha sido necesario el diseño de los componentes que no se encuentran en las librerías, es decir, es necesario realizar un diseño esquemático de dichos componentes y a la vez hay que realizar el diseño de su huella para la posterior fabricación de la PCB. Esto requiere que ambos diseños cuenten con el mismo número de conexiones que el componente original y en el caso de la huella que su diseño posea las mismas dimensiones que el producto real, para que luego pueda ajustarse correctamente al sitio que se le asignó. Una vez mencionados estos pasos, se pasa a mostrar el esquemático diseñado y se procederá al análisis de cada parte que lo conforma.

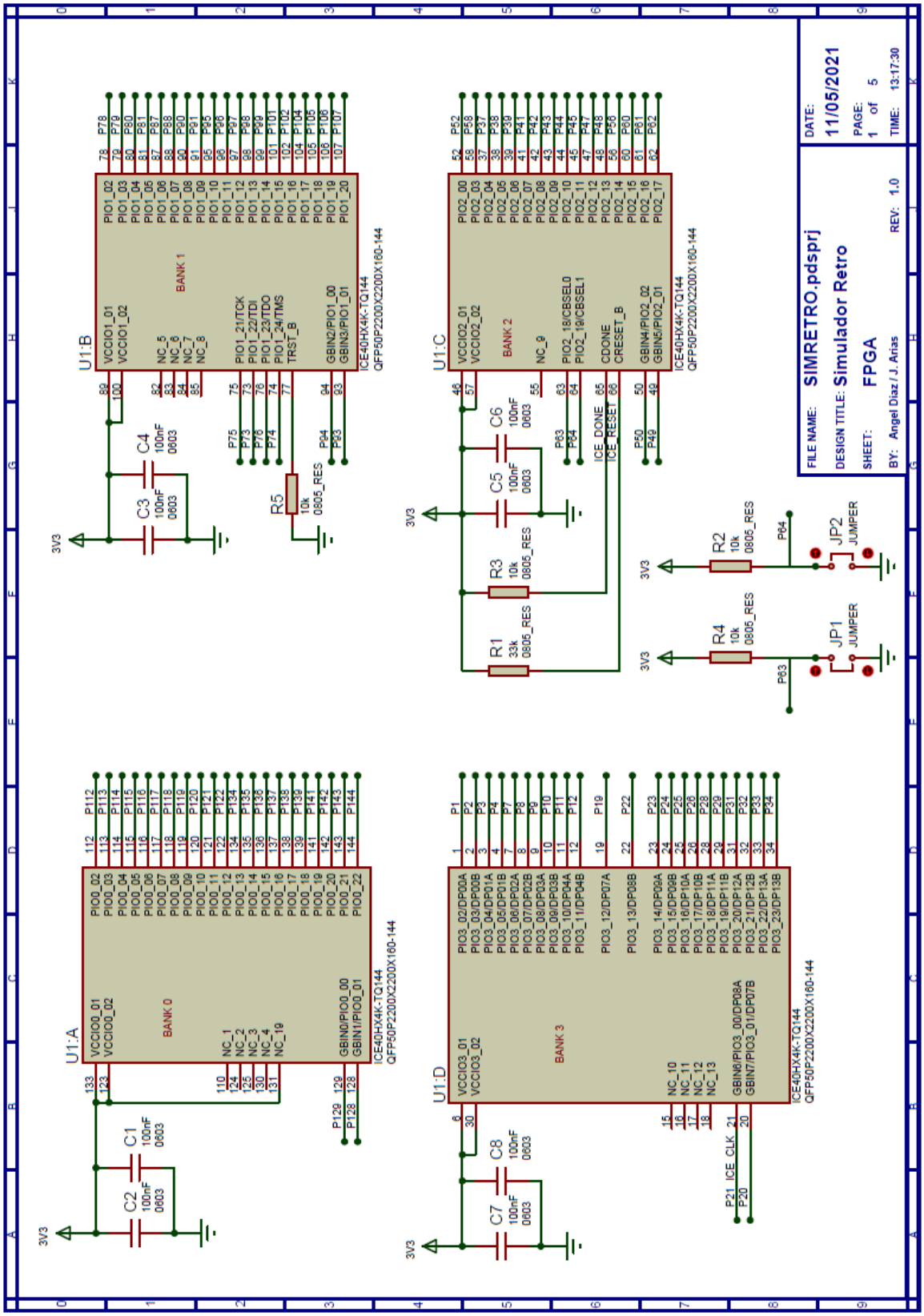


Ilustración 10. Hoja 1 del esquemático: FPGA.

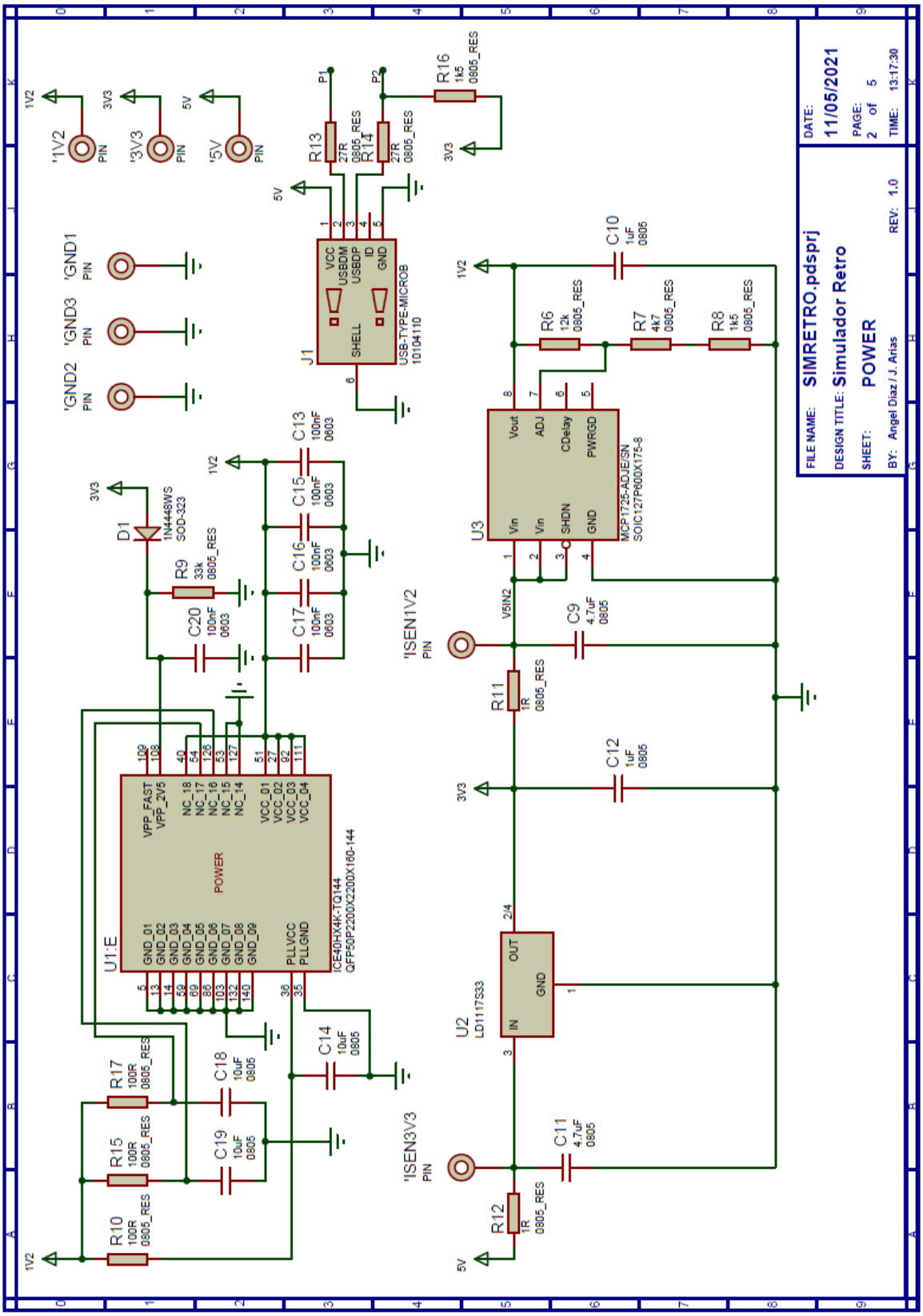


Ilustración 11. Hoja 2 del esquemático: Alimentación.

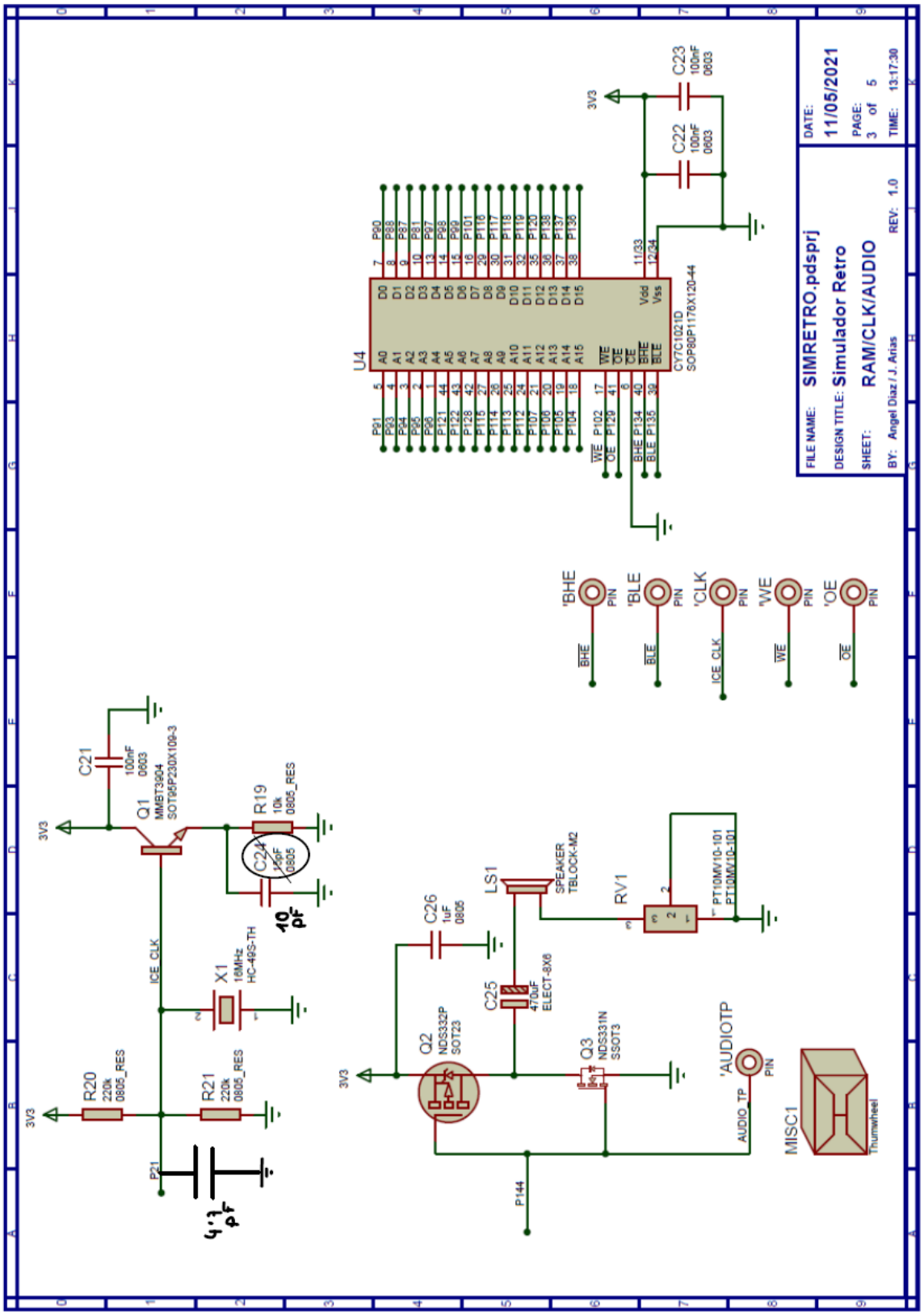


Ilustración 12. Hoja 3 del esquemático: RAM/Reloj/Audio.

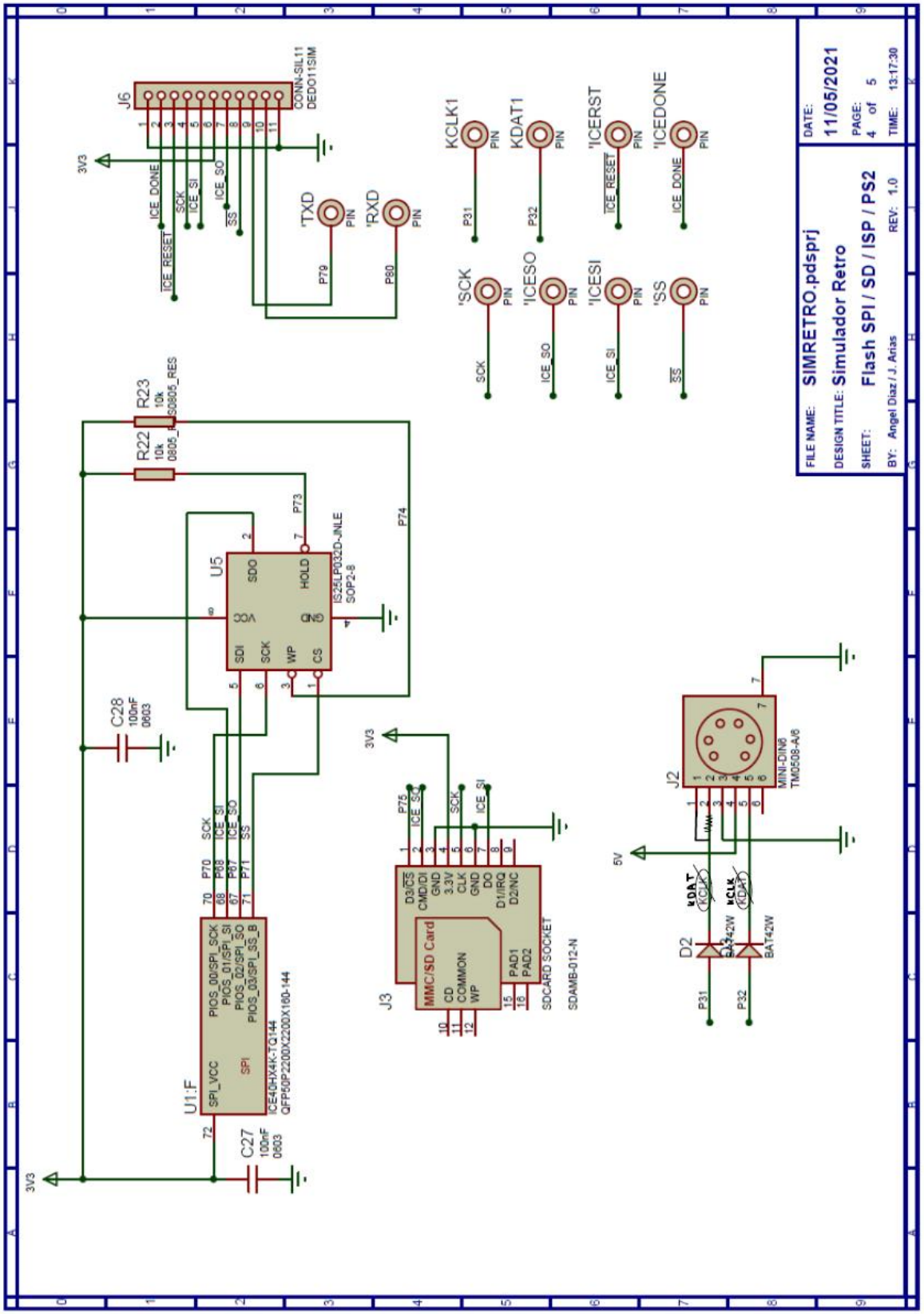


Ilustración 13. Hoja 4 del esquemático: Flash SPI /lector SD/ ISP / interfaz PS2.

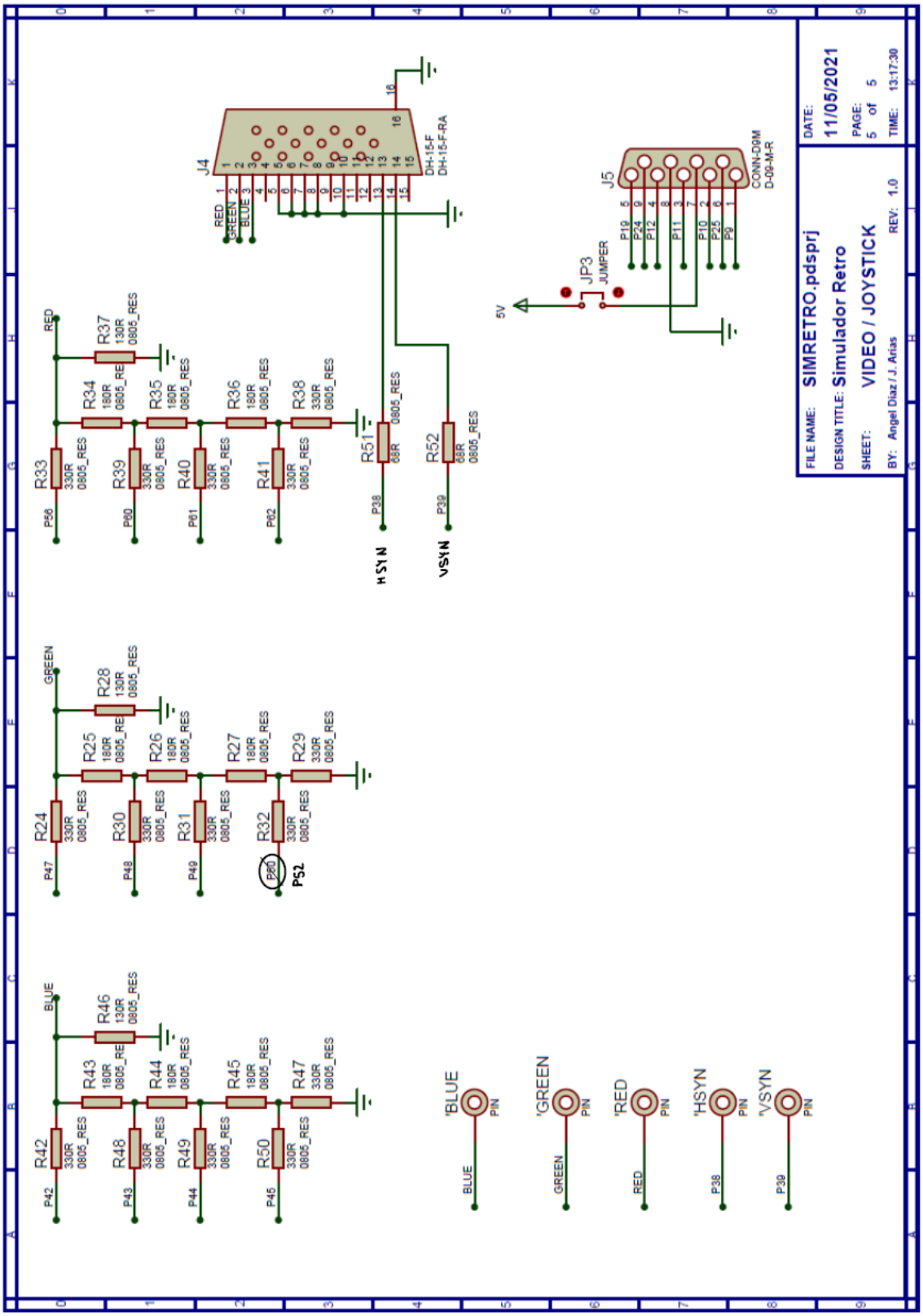


Ilustración 14. Hoja 5 del esquemático: Video / Joystick.

3.1.1 Conexionado de la FPGA

Como se puede ver en la primera figura, el módulo de nuestra FPGA está compuesto por 6 piezas distintas pero que funcionarán posteriormente como una única conformando el chip de nuestra FPGA. A los primeros 4 módulos se les proporciona una tensión de alimentación que como se puede observar se trata de 3.3V y además le añadimos 2 condensadores de desacoplo, conectados en paralelo, uno por cada pin de alimentación. En el esquemático se muestra la FPGA de 1K, los pines NC que están conectados se refieren a la FPGA 4K, debido a que en esta placa se pueden soldar los dos modelos de FPGA. Por otro lado, el banco número 1 posee una resistencia de pull-down con valor de 10kΩ.En el banco 2, a parte de los 2 condensadores de desacoplo, tenemos colocados 2 resistencias de pull-up conectados a los pines de DONE y de RESET de la FPGA. En el banco 3 de la FPGA podemos ver que el pin 21 se ha asignado para que a través de este salga la señal de reloj de nuestra placa. En el 4º banco que sería el núcleo de nuestra FPGA, este en cambio está alimentado a 1.2V, con 3 resistencias de pull-up y los valores de los condensadores de desacoplo poseen una capacidad bastante mayor que las de los otros módulos debido a que se trata de la alimentación de los PLLs y es una zona crítica en donde debemos asegurarnos que no se cuela ninguna componente alterna (ruido). En el otro extremo tenemos colocados 4 condensadores de desacoplo por el mismo motivo. Finalmente, para terminar, tenemos el último y 5º banco de la FPGA que es el que se encuentra conectado directamente con la memoria Flash SPI, alimentado también a 3.3V con sus correspondientes condensadores de desacoplo y las resistencias de pull-up necesarias para el correcto funcionamiento de la memoria, que en nuestro caso hemos utilizado la memoria IS25LP032D-JNLE, aunque el modelo montado es el N25Q64, con el doble de memoria y disponible en stock.

Los valores de alimentación comentados en este punto de conexionado de la FPGA son conseguidos a través de la alimentación que nos ofrece el conector USB, entregando 5V de tensión. Estos 5V se hacen pasar por un primer regulador de tensión que es el LD1117S33 que se encargan de reducir la tensión a 3.3V (down-conversion). Una vez conseguidos esos 3.3V, estos se hacen pasar por otro regulador de tensión que este nos dará un valor de tensión de 1.2V a su salida, pero este se trata del regulador MCP1725-ADJE/SN. Ambos son reguladores lineales pues la poca potencia necesaria no justifica la complejidad de unos reguladores conmutados.

3.1.2 DAC

En la última figura de nuestro diseño esquemático como podemos ver tenemos posicionados nuestros 3 DAC o redes R-2R. Estos poseen 4 entradas correspondientes a 4 pines diferentes de la FPGA y que a cada salida de cada DAC tenemos una componente de color (azul, verde o roja), que estas salidas irán conectadas directamente a 3 pines del conector VGA, para la posterior

visualización de la señal de video en un monitor. Se ve que los valores de las resistencias no son exactamente unas el doble que las otras, (aunque los valores se aproximan más si tenemos en mente la resistencia serie de los pines de la FPGA ($\sim 20\Omega$). pero se realizaron los cálculos y simulaciones necesarias para comprobar que los DAC generaban los escalones de manera correcta y con esos valores funcionaba de manera correcta, así que esos fueron los valores finales utilizados en nuestro diseño.

Como podemos visualizar en la figura también tenemos 3 pines de prueba situados para cada componente de color, colocados por si fuera necesario cualquier tipo de monitoreo a través de un osciloscopio para poder comprobar los valores de las señales que están llegando a esos pines a la hora de configurar las señales de video.

3.1.3 Potenciómetro

Como se mencionó en el apartado 3.1, a la hora del diseño de nuestro prototipo fue necesario el diseño e implementación de algunos componentes que no existían en las librerías propias de Proteus, pues este componente es uno de ellos. Este potenciómetro está conectado al altavoz que este a su vez está conectado a un amplificador inversor a través de un condensador electrolítico, que realiza la función de anular cualquier componente de continua, que se puede colar en la salida de este amplificador. Este es encargado de amplificar la señal que le llega a través del pin número 144 de la FPGA, para que pueda ser lo suficientemente audible a través del altavoz conectado.

El potenciómetro o resistencia variable está conectado para poder variar la corriente entrante a este altavoz para así poder aumentar o disminuir el volumen de los tonos que emite este altavoz. El valor de nuestro potenciómetro tiene una resistividad máxima de 100Ω, que se irá modificando según quiera el usuario con la rueda que se introduce a través del orificio pasante que posee.

3.1.4 Interfaces y reloj

La primera interfaz con la que nos encontramos en nuestro diseño en la segunda figura adjunta del esquemático se trata del conector USB, donde vemos que hemos utilizado 5 de sus 6 pines de conexión. En el caso de nuestro prototipo, utilizamos esta conexión únicamente para obtener la alimentación de 5V, pero si en un futuro quisiéramos utilizarlo como puerto de datos, también se podría haciendo las correcciones necesarias via firmware. Este conector tiene 2 de sus pines

conectados a los primeros pines de la FPGA y en el segundo como se puede ver en el diseño se le ha conectado una resistencia de pull-up.

Como segunda interfaz nos encontramos con el conector PS2, en el que únicamente se utilizan 5 pines de los 7 que lo componen debido a que estos 2 que no se utilizan no están implementados. En el pin 5 hemos conectado la señal de reloj junto con un diodo que impide que los 5V del teclado lleguen al pin, que quedará en 3.3V gracias a un pull-up interno y en el pin 1 la señal de datos con el mismo procedimiento que el reloj. Tenemos dos errores de diseño que se comentará más adelante en el apartado de errores del proyecto, pero se cambió el conexiado del PS2 debido a que el pin 2, como podemos ver en la captura del datasheet del ps2, el pin 2 no está implementado y el pin del reloj es el 5. El de datos está mal conectado también porque es el número 1. Por el contrario, las conexiones de alimentación estaban correctamente situadas en el diseño, introduciendo los 5V por el pin número 4 que le corresponde y las tierras en los pines 3 y 7.

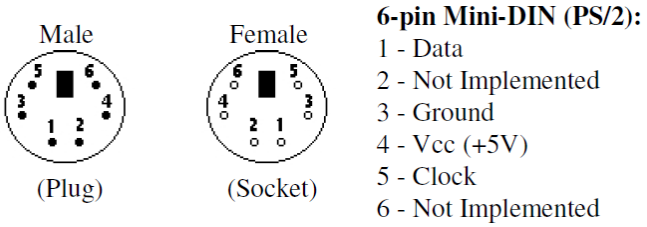


Ilustración 15. Conexionado de los pines de la interfaz PS2.

En esta misma figura del esquemático tenemos otra interfaz más que se trata de la de la tarjeta SD, que como vemos únicamente se han utilizado los pines de alimentación correspondientes alimentando al lector de tarjetas con 3.3V de alimentación y utilizando los pines 1,2,5 y 7 para conectarlos a los pines correspondientes de la FPGA que se tratan del P75, ICE_SO, SCK_ ICE_SI. Este lector posee dos conexiones a tierra que han sido correctamente conectadas como se puede comprobar.

Cabe comentar también que en esta figura tenemos instaurado el conector de programación de 11 pines, con los que serán conectadas, nuestro diseño y a parte una pequeña placa diseñada por Jesús Arias que nos servirá de gran ayuda para programar nuestra placa y realizar las pruebas necesarias a la hora de cargar nuestros programas de firmware. Este conector está alimentado a una tensión de 3.3V y a su vez a los correspondientes pines de la FPGA.

Para finalizar tenemos diferentes pines de prueba, para poder monitorear las señales más críticas para comprobar que todo funciona como debe en cada proceso, como pueden ser las señales de datos y reloj del PS2, el ICE_RST de la FPGA, ICE_DONE, ICESI, ICESO, …

En la última figura de nuestro esquemático tenemos instaurados las últimas 2 interfaces que nos faltan por mencionar que se tratan del VGA y el DB9. El VGA como podemos ver tiene las 3 entradas que son las componentes de color generadas mediante los 3 DAC diseñados, que son rojo, verde y azul. Además, tiene conectados 2 pines conectados a través de 2 resistencias con valor de 68Ω. El conector DB9 tiene todos sus pines conectados a pines de la FPGA para la correcta comunicación, menos 2 de ellos el número 7 que lo tiene conectado a través de un jumper a la alimentación de 5V y el número 8 conectado a tierra.

Para finalizar tenemos que comentar el circuito oscilador que conforma nuestro cristal de cuarzo y que hace que este no pare de oscilar cuando la placa está en funcionamiento. Este circuito ha sido conectado con el pin 21 de la FPGA donde le mandaremos la señal de reloj generado mediante esta implementación. Las señales de reloj de los dispositivos deben ser señales muy precisas y que funcionen de manera muy concreta debido a que son la base del funcionamiento de la placa, debido a esto se escogen los cristales de cuarzo debido a que son muy estables generando estas señales. Se trata de un oscilador de tipo Colpitts, con el transistor Q1 en configuración de colector común, que genera una onda senoidal en el pin 21 de la FPGA y un Schmitt-trigger interno convierte esta onda en una señal cuadrada.

3.1.5 Memorias

En la 3ª figura adjunta de nuestro diseño del esquemático tenemos instaurada una de las memorias externas de nuestro diseño que se trata de la memoria RAM SOP80P1176X120-44. Esta memoria está alimentada a 3.3V con sus 2 característicos condensadores de desacoplo para que se vea aumentada la capacidad de ese nodo y en alterna podamos obtener una tierra lo más perfecta posible, para que no haya variaciones en la tensión en ese nodo que pueda desestabilizar el funcionamiento del dispositivo. Esta memoria se caracteriza porque tiene un bus de direcciones de 16 bits y un bus de datos con otros 16 bits en paralelo. Además, posee una señal de habilitación de escritura o WE (write enable), una señal de habilitación de salida o OE (output enable), la señal BHE (Byte High Enable) que se encarga de activar los 8 bits más significativos del bus de datos, es decir los bits del D15-D8, la señal BLE (Byte Low Enable) encargada de activar los 8 bits menos significativos del bús, del D7 al D0 y finalmente tenemos la señal de habilitación del chip o CE (chip enable). Todas estas señales mencionadas anteriormente son activas en baja, luego para activarlas es necesario que exista en su entrada un “0” lógico. Debido a esto tenemos conectado permanentemente la señal de CE a una tierra,

para que la memoria funcione en todo momento que conectamos la alimentación de nuestra placa.

En la figura 4 de nuestro esquemático está situada nuestra memoria Flash SPI IS25LP032D-JNLE conectada a el último banco de nuestra FPGA. En primer lugar, el pin 1 se trata de la señal CS (Chip select) conectado directamente al módulo de la FPGA, que como podemos ver es activo en baja y cuando este pin está en alto, el dispositivo está deshabilitado y el pin número 2 se encuentra en un estado de alta impedancia, en cambio, cuando el CS se encuentra en baja este habilita el dispositivo, situándolo en el modo de funcionamiento. Los pines 2,5 y 6 se tratan de señales SPI habituales y los pines 1 y 7 se encargan de las funciones especiales como pueden ser protección de escritura, hold o alternativamente bits de datos para modos de 4 bits por ciclo (Modo Quad-SPI).

3.2 Diseño de la PCB

Una vez completado el proceso de diseño y comprobación del esquemático de que todo está conectado como debe, se procede a comenzar con el diseño del circuito impreso o PCB, para llevar a cabo esta tarea como ya se comentó anteriormente se seguirá utilizando la herramienta de software Proteus 8.11.

Lo primero que debemos hacer antes de comenzar con la colocación de los componentes sobre la PCB, tenemos que asegurarnos de que contamos con todas las huellas y si algunas no existen debemos diseñarlas con las medidas exactas que nos indican los datasheet de cada componente. Una huella hace referencia a la serigrafía y los pads que nos sirve de referencia a la hora de posicionar los componentes sobre la PCB diseñada. Es sumamente importante que estas huellas sigan al pie de la letra las dimensiones impuestas por los fabricantes para que los pines de los componentes comprados encajen a la perfección con los pads y con el diseño del circuito interno de la PCB. A la hora de crear las huellas puede ser de gran ayuda añadir algo representativo como una marca en una esquina para saber que esa esquina marca la referencia para saber cómo orientar el componente en nuestra PCB. En caso de que la huella no coincidiera con el número de pines o la dimensión de estos, los pads de la huella no estuvieran asignados correctamente y el fallo es muy grave que no se puede solucionar de alguna manera cuando la placa ya ha sido fabricada, esto podría llevarnos a tener que rediseñar de nuevo la PCB y volver a fabricarla.

El único componente que ha tenido que ser diseñado desde 0, es el potenciómetro. Se tuvo que realizar el diseño para añadirlo en el esquemático y además crear su huella para insertarla en la PCB. Con ayuda del datasheet, obtenemos las dimensiones que deben tener las distintas partes

que conforman este componente, como la distancia entre pads, el orificio que posee en el medio del componente para insertar la rueda con la que manejaremos su resistividad, etc.

Otros componentes como pueden ser los conectores de las interfaces como son, el conector USB, el conector PS2, el lector de tarjetas y el DB9, no ha sido necesario el diseño desde 0 debido a que se ha podido reutilizar tanto el diseño del componente como el diseño de la huella de otros proyectos que ha realizado mi tutor responsable de este proyecto, Jesús Arias.

Una vez han sido diseñadas las huellas necesarias se procede a colocar los componentes en el layout de la PCB. Las dimensiones de la placa PCB inicialmente es de aproximadamente 100x100mm, que son las dimensiones estándar que propone en fabricante con 2 capas. A la hora de la colocación de los componentes en su respectivo sitio asignado para el correcto diseño de nuestra placa, se van a tener en cuenta una serie de normas sobre la compatibilidad electromagnética, como pueden ser:

- Los condensadores de desacoplo deben estar situados lo más cerca posible del componente al que se conectan para protegerlo, especialmente los de menor valor en el caso de que se conecte más de uno.
- Las pistas no deben formar áreas grandes para evitar que se formen bucles de corriente generando acoplamiento magnético entre pistas. Las pistas pueden llegar a actuar como antenas, si conseguimos de alguna manera que estas radien menos al exterior también se conseguirá que las emisiones recibidas de otras partes del circuito o desde el exterior puedan afectar en mucha menor medida al funcionamiento de nuestro circuito.
- En caso de que fuera posible conseguir uno o varios planos de masa de manera que la señal pueda tener el camino de vuelta lo más corto posible y con una impedancia común lo más baja posible también.

Una vez establecidas las normas de diseño de nuestra placa PCB y las restricciones que podemos llegar a tener con la colocación de los componentes en el producto, pasamos a la colocación de los componentes en el diseño virtual de nuestra PCB. Los primeros elementos que han sido colocados en nuestro diseño han sido en primer lugar, nuestra FPGA que ha sido situada en el centro de la PCB como corazón de esta que es. Posteriormente a su lado ha sido situada la memoria RAM externa, no muy alejada de esta para facilitar el posterior rutado entre ambas. El siguiente paso fue añadir las diferentes interfaces (micro USB, VGA, PS2, DB9) que tenemos en nuestro diseño debido a que ocupan bastante espacio, para saber con certeza el hueco que nos quedaría posteriormente para los componentes restantes.

Cuando se ha terminado de colocar los conectores de las interfaces, el resto de componentes que conforman nuestro producto se colocan por la PCB restante intentando que estos ocupen el menor espacio posible y que el posicionamiento que adopten en la PCB ayude lo más posible al rutado posterior de manera que se puedan cumplir en la medida de lo posible las normas de compatibilidad electromagnética. En el caso de nuestro diseño, no se consiguió reducir el tamaño de 100x100mm debido a la cantidad de componentes que tiene nuestro producto.

El siguiente paso al posicionamiento de los componentes en la placa, es el rutado de las pistas de los componentes. En esta fase del diseño seguimos la estrategia de rutar el mayor número posible de pistas por la cara superior. Debido a que esta es la norma que se ha impuesto más importante, a la hora de ir rutando algunos componentes estos podrían ser reubicados en la placa para poder conseguir el menor número de conexiones por la cara inferior de nuestra placa. Esta estrategia tiene como fin que en la cara inferior tengamos un único plano de masas con el menor número de secciones posibles, con esto lo que se consigue es disminuir el camino de retorno de las señales y que por tanto se pueda producir alguna diferencia de potencial que afecte a nuestra referencia/tierra y pueda afectar al correcto funcionamiento del circuito. Como veremos más adelante en las figuras de nuestra PCB finalizada, algunas conexiones han sido realizadas por la cara inferior debido a que era imposible encontrar un camino por el que conectarlas en la cara superior. Gracias a que los pines de la FPGA son casi todos equivalentes la asignación de señales a pines se ha ido modificando sobre la marcha para conseguir un rutado lo más simple posible.

Debido a que el programa de software Proteus posee una herramienta que es denominada “autorutado”, esta herramienta no ha sido utilizada debido a que lo que hace es realizar una serie de cálculos e intentar diseñar las pistas más cortas posibles, pero no distingue por caras y realiza el diseño de las rutas por ambas caras (aunque a veces no es capaz ni de realizarlo correctamente) y eso es lo que no queremos en nuestro diseño por lo que se optó por un rutado manual. Pese a que este lleva más tiempo conseguimos un rutado mucho más correcto desde el punto de vista de la compatibilidad electromagnética.

Una vez terminado el rutado de las pistas, han sido añadidas las serigrafías que nos ayudarán a posicionar los componentes en nuestro producto y nos será de gran ayuda a la hora de soldar los componentes cuando haya llegado el diseño de nuestra placa. A parte de los componentes, como ya mencionamos en apartados anteriores, se han añadido distintos puntos de test junto con la serigrafía de qué señal es. Esto nos será de gran utilidad a la hora de comprobar si el hardware funciona correctamente y también puede ser útil a la hora de depurar nuestro diseño software ya que se han incluido un montón de puntos de test para las señales más críticas de nuestro diseño.

Por último, añadir que también se han añadido unas serigrafías en el único hueco libre de la placa, con el nombre del proyecto, el nombre del alumno del TFG y una serigrafía con el logo de la escuela ETSIT Uva. Después de todo este proceso, han sido generados los archivos Gerber y estos son los que han sido enviados al fabricante de PCB, PCBWay. Estos ficheros contienen toda la información necesaria para la correcta fabricación de la placa de circuito impreso. Finalmente se adjunta en las siguientes figuras la cara superior e inferior del diseño terminado de la PCB.

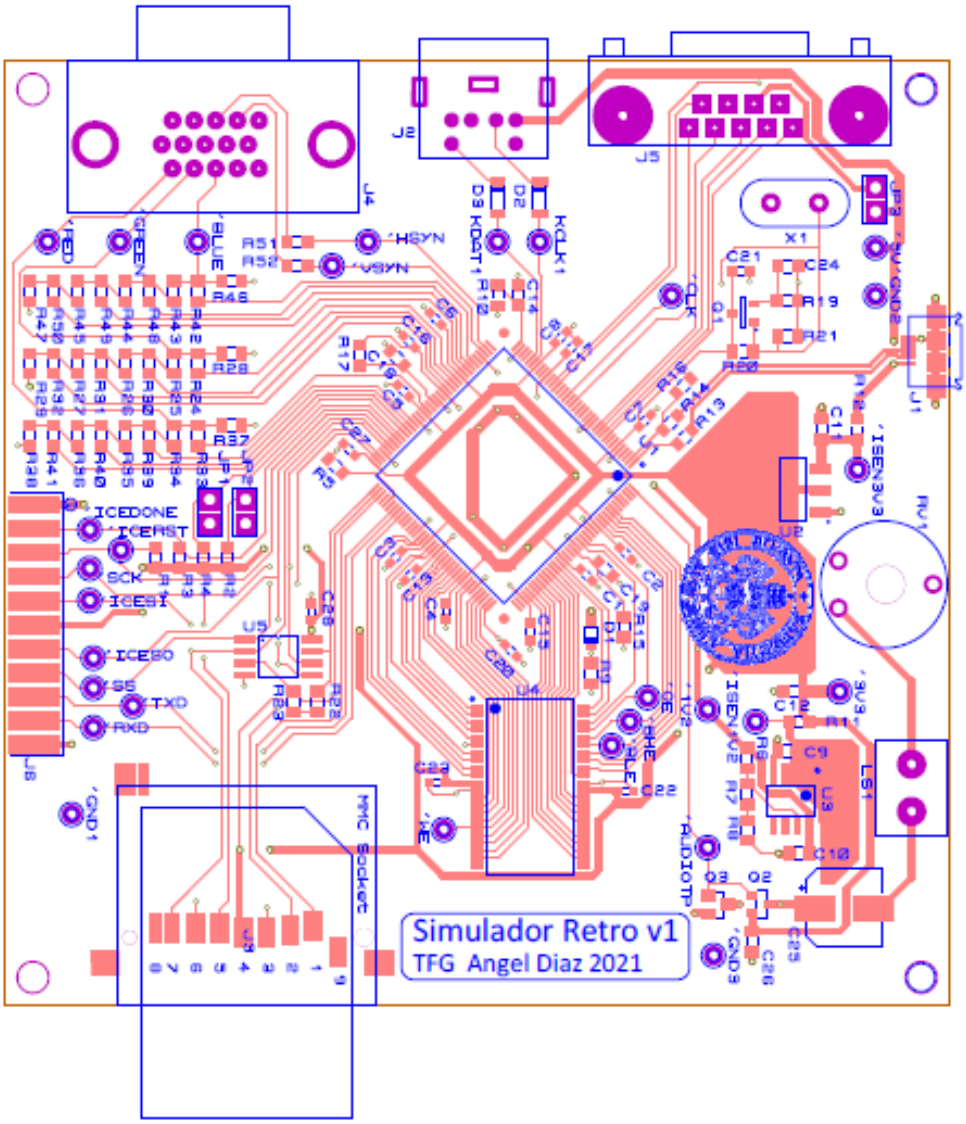


Ilustración 16. Cara superior del diseño de la PCB.

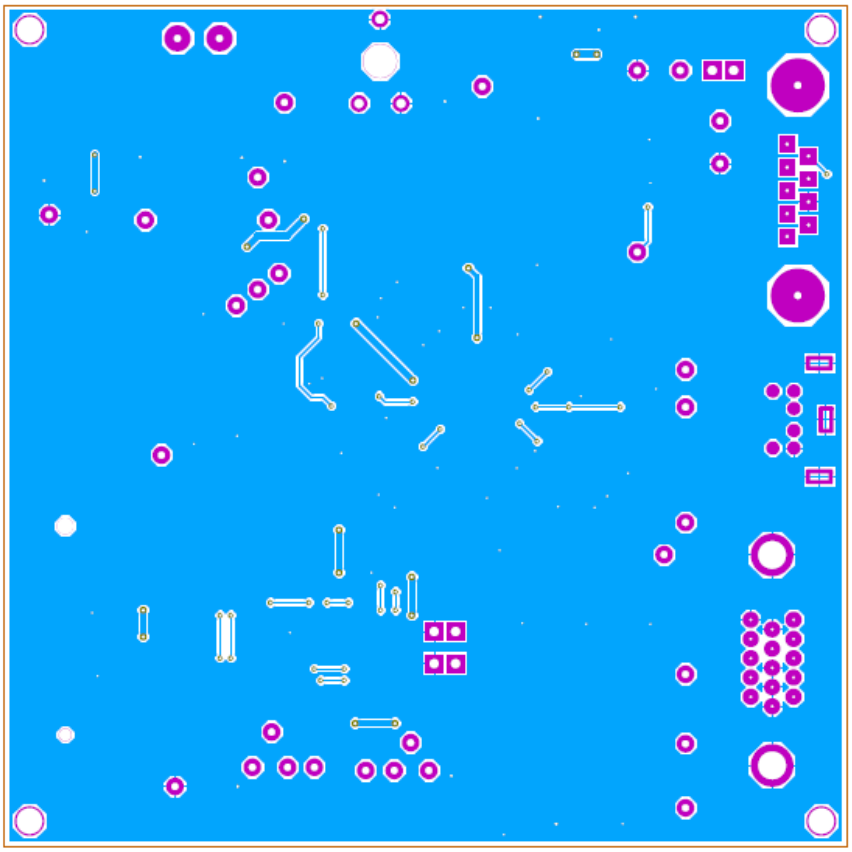


Ilustración 17. Cara inferior del diseño de la PCB.

3.3 Fabricación de la PCB

El proceso de fabricación de nuestra placa de circuito impreso ha sido llevado a cabo por la empresa asiática PCBWay. Su fábrica está situada en la ciudad oriental Shenzhen, en China. En su entorno web nos ofrecen servicios muy diversos, con distintas opciones y precios en función del circuito impreso que se quiera fabricar, desde prototipos PCB, plantillas SMD, PCBs flexibles, avanzadas o impresión 3D. En el caso de nuestro pedido ha sido el encargo de 5 PCBs con nuestro diseño anteriormente propuesto.

Las principales ventajas de esta empresa es el bajo precio para la calidad de producto que se ofrece. El pedido tardo en llegar apenas una semana desde que se realizó el encargo pasando por aduanas y viajando desde China hasta España.

3.4 Lista de materiales (Bill of materials – BOM)

Todos los componentes utilizados para la composición de nuestro producto final se recogen en un listado de materiales denominado por el programa Proteus como BOM, porque es de este programa software desde el cuál extraemos este listado de materiales. Las siglas proceden del inglés y hacen referencia a *"Bill of materials".* En este documento se pueden añadir los campos que se crean necesarios, como podemos ver en la figura adjunta de nuestro listado, en primer lugar, se muestra la cantidad de componentes necesarios, la siguiente columna está relacionada con la referencia del componente dentro de nuestro esquemático, para posteriormente saber dónde colocar a la hora de realizar el montaje de los componentes de nuestra PCB. La tercera columna posee el valor del componente correspondiente y la cuarta columna posee el código de su huella. Las siguientes columnas están relacionadas con la empresa a la que se debe pedir cierto componente, el código de componente de la empresa, el distribuidor y el código de stock.

Esto permite calcular de una forma más simple los costes asociados a la fabricación de nuestra PCB y además facilita la tarea de realizar los pedidos al distribuidor de componentes en cuestión.

Si en nuestro caso se quisiera realizar una estimación del precio real de nuestro producto, a parte del coste de los componentes, deberíamos añadir el coste de la placa de circuito impreso, el tiempo empleado por los diseñadores, mano de obra y algunas variables más que podrían afectar en el aumento del precio del producto.

En relación a esto pasamos a estimar un precio de nuestro producto que aproximadamente en relación a los precios del mercado tenemos:

- Condensadores: 18 x 0.048 = 0.864€
- Resistencias: 51 x 0.001 = 0.051€
- FPGA: 5.82€
- Reguladores de tensión: 0.68 + 0.25 = 0.93€
- Memoria SRAM: 2.65€
- Memoria Flash: 1.54€
- Transistores: 0.5 x 3= 1.5€
- Diodos: 0.33 x 3= 0.99€
- Conectores: $\sim 8€$

Haciendo un coste total de los componentes montados en la placa de unos 22€, más casi 1€ del coste de la PCB, hace un total de 23€ del coste del producto. Eso sin contar la mano de obra y

las horas de montaje y programación que esta conlleva, pudiendo aumentar bastante el coste del producto.

Bill Of Materials for Simulador Retro

Design Title Simulador Retro
Author Angel Diaz / J. Arias
Document Number
Revision 1.0
Assembly Variant BOM_reducido
Design Created viernes, 22 de noviembre de 2019
Design Last Modified martes, 11 de mayo de 2021
Total Parts In Design 101

28 Capacitors

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
18	C1-C8, C13, C15-C17, C20-C23, C27-C28	100nF	0603				
2	C9, C11	4.7uF	0805				
3	C10, C12, C26	1uF	0805				
3	C14, C18-C19	10uF	0805				
1	C24	15pF	0805	Kemet	C0805C150J5GACTU	Farnell	1414668
1	C25	470uF	ELECT-8X6	Panasonic	EEEFC1E471AP	Farnell	1244366
Sub-totals:							

51 Resistors

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
2	R1, R9	33k	0805_RES				
7	R2-R5, R19, R22-R23	10k	0805_RES				
1	R6	12k	0805_RES				
1	R7	4k7	0805_RES				
2	R8, R16	1k5	0805_RES				
3	R10, R15, R17	100R	0805_RES				
2	R11-R12	1R	0805_RES				
2	R13-R14	27R	0805_RES				
2	R20-R21	220k	0805_RES				
15	R24, R29-R33, R38-R42, R47-R50	330R	0805_RES				
9	R25-R27, R34-R36, R43-R45	180R	0805_RES				
3	R28, R37, R46	130R	0805_RES	Multicomp	MCWR08X1300FTL	Farnell	2447568
2	R51-R52	68R	0805_RES				
Sub-totals:							

5 Integrated Circuits

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
1	U1	ICE40HX4K-TQ144	QFP50P2200X2200X160-144	Lattice Sem.	ICE40HX4K-TQ144	RS	772-0057
1	U2	LD1117S33	SOT230P700X180-4	ST	LD1117S33CTR	Farnell	1467779
1	U3	MCP1725-ADJE/SN	SOIC127P600X175-8	Microchip	MCP1725-ADJE/SN	Farnell	1834878
1	U4	CY7C1021D	SOP80P1176X120-44	Cypress Sem.	CY7C1021DV33-10ZSXI	RS	181-7610
1	U5	IS25LP032D-JNLE	SOP2-8	ISSI	IS25LP032D-JNLE	Farnell	2787053
Sub-totals:							

3 Transistors

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
1	Q1	MMBT3904	SOT95P230X109-3	Nexperia	MMBT3904,215	Farnell	1757935
1	Q2	NDS332P	SOT23	ON semiconductor	NDS332P	Farnell	1471069
1	Q3	NDS331N	SSOT3	ON semiconductor	NDS331N	Farnell	1839006
Sub-totals:							

3 Diodes

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
1	D1	1N4448WS	SOD-323	Multicomp	1N4448WS-7-F	Farnell	2306367
2	D2-D3	BAT42W	SOD123	Multicomp	BAT42W+	Farnell	2675343
Sub-totals:							

11 Miscellaneous

Quantity	References	Value	PCB Package	MANUFACTURER	PARTNUMBER	Supplier	Stock Code
1	J1	USB-TYPE-MICROB	10104110	Hirose	ZX62-B-5PA(33)	Farnell	2554978
1	J2	MINI-DIN6	TM0508-A/6	TE Connectivity	5749180-1	RS	710-0441
1	J3	SDCARD SOCKET	SDAMB-012-N	Multicomp	SDAMB-01215BT00	Farnell	9186140
1	J4	DH-15-F	DH-15-F-RA	ASSMANN WSW	A-HDF 15 A-KG/T	RS	674-0971
1	J5	CONN-D9M	D-09-M-R	TE Connectivity	2301843-1	Farnell	2857982
3	JP1-JP3	JUMPER	CONN-SIL2				
1	MISC1	Thumwheel		Amphenol Piher	JPEPL5034INI	Digikey	1993-JPEPL5034INI-ND
1	RV1	PT10MV10-101	PT10MV10-101	Amphenol Piher	PT10MV10-101A2020-S	Digikey	1993-PT10MV10-101A2020-S-ND
1	X1	16MHz	HC-49S-TH	TXC	9B-16.000MAAJ-B	Farnell	1842216
Sub-totals:							

Totals:

martes, 11 de mayo de 2021 13:21:14

Ilustración 18. Listado de materiales - BOM.

3.5 Montaje de los componentes en la PCB

El montaje de los componentes sobre la placa de circuito impreso fue realizado en el laboratorio de proyectos, laboratorio número 21 de la ETSIT en la Universidad de Valladolid. El laboratorio cuenta con los materiales necesarios para realizar dicha tarea con total precisión debido a que están en posesión de soldadores con elevada temperatura de fusión (alrededor de los 300ºC), microscopios ópticos y digitales, pinzas de precisión, estaño de distintos tipos y grosores para distintos tipos de soldaduras dependiendo del tamaño del componente a soldar y otras herramientas que son de gran utilidad a la hora del montaje de los componentes sobre la PCB, como puede ser también los rotuladores de flux, que este compuesto es de gran utilidad debido a que nos ayuda en gran medida a la hora de soldar pines como por ejemplo los de la FPGA que poseen un tamaño muy reducido.

A la hora de comenzar con el montaje de la placa es aconsejable trazar una estrategia y una organización cuando se vaya a comenzar con la tarea, siendo una buena recomendación decidir por qué componentes vamos a comenzar a soldar. Por ejemplo, en el caso de este producto se comenzó con la soldadura de la FPGA porque es una de las más críticas y la que más problemas podría causar, debido al número de pines que está posee y el tamaño que estos tienen, debido a que todos deben quedar correctamente posicionados en los pozos de estaño de cada pin para que estos hagan un contacto correcto con las pistas que les corresponden. En los circuitos integrados es de vital importancia fijarse en la orientación y el sentido de los componentes ya que estos suelen ser simétricos como el caso de la FPGA y se pueden cometer errores fatales que ocasionaran un no funcionamiento del producto, debido a que los pines no están soldados donde deben o en el peor de los casos una mala alimentación del componente, siendo alimentado con tensiones negativas donde no debe y puede llegar a estropearse, siendo necesario el reemplazo del componente. Por esos estos circuitos integrados suelen tener una marca en una de las esquinas que nos indica hacia donde debemos orientar el componente en nuestra placa para que no haya ninguna equivocación.

En segundo lugar, se procedió con la soldadura de la memoria RAM, la cual también era crítica por las mismas razones que la FPGA y posteriormente se procedió a soldar componentes como los reguladores de tensión, transistores utilizados, etc, dejando en último lugar los componentes pasivos debido a que se cuenta con un mayor número de estos y finalmente el montaje de los componentes terminó con el montaje de los componentes de inserción como las interfaces y el cristal de cuarzo.

A la hora de realizar la soldadura de los pads es muy importante asegurarse que los componentes quedan correctamente soldados, para esto se recomienda la utilización de las

pinzas de precisión y el microscopio para realizar un poco de presión sobre el pad que se ha soldado y comprobar que no se mueve, quedándose completamente fijado. Como ya hemos comentado anteriormente el flux es una herramienta muy útil debido a que se encarga de eliminar los óxidos de los pads y nos ayuda a concentrar el calor para que se distribuya el estaño de manera uniforme en la soldadura. En la siguiente figura tenemos el resultado final después de haber añadido todos los componentes en nuestra placa de circuito impreso.

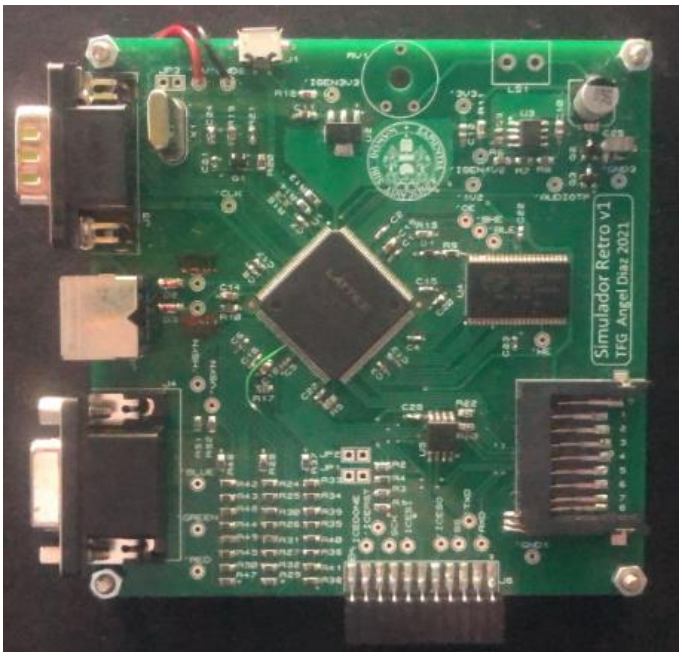


Ilustración 19. PCB con todos los componentes montados.

3.6 Errores en el diseño

Pasamos a comentar los tres errores que no fueron corregidos antes de enviar a fabricar la placa y montar los componentes en esta, pero que por fortuna no eran errores muy graves y han podido solventarse sin mucha complicación:

- El primer error se haya en el conexionado de la FPGA con la interfaz de video VGA, debido a que, viendo la lista de pines de la FPGA y su correspondiente asignación, se vió que el pin 50 no está disponible en la FPGA 4K que es el modelo que estamos utilizando para este prototipo. La señal correspondiente (Green_0) estaba conectada a ese pin que no tiene ninguna utilidad en ese pin luego se conectó mediante un cablecillo uniendo ambas pistas para que la señal actuará también sobre el pin 52, así tuviera utilidad a la hora de realizar nuestro firmware.

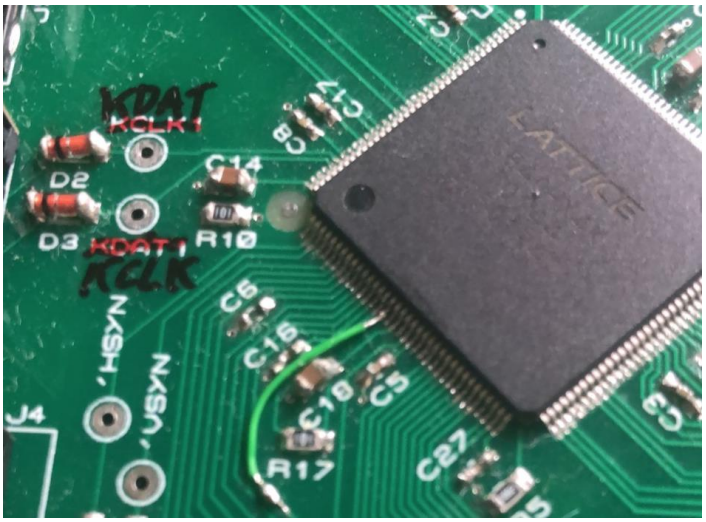


Ilustración 20. Corrección primer error diseño.

- El segundo error tiene que ver esta vez con una de las interfaces, esta vez es con el PS2. Se realizó una mala asignación de pines en el diseño del esquemático y por consecuencia, el diseño de la PCB se realizó de manera incorrecta, debido a que el pin del dato estaba conectado al pin 2 que no se usa, cuando debería ir conectado al pin número 1. Al darse cuenta del fallo a la hora de verificar el hardware, se realizó una unión de ambos pines mediante estaño de soldadura y así quedo corregido el error. Aunque la serigrafía de los puntos de prueba KCLK y KDAT ha quedado intercambiada, eso no afecta al correcto funcionamiento de la interfaz.

Ilustración 21. Corrección segundo error diseño.

- El último error ha sido que se han tenido que ajustar las capacidades del oscilador del reloj para mejorar el arranque de la señal de reloj, debido a que antes de realizar esta modificación había veces que no arrancaba el dispositivo.

4. Desarrollo Software

4.1 Introducción

Una vez detallado toda la estructura que engloba al desarrollo del hardware de nuestro producto y se consigue llevar a cabo correctamente se procede a implementar el software que compruebe que las partes fundamentales de nuestro proyecto funcionan correctamente y como se espera.

El desarrollo del firmware puede ser la parte más tediosa del proyecto debido a que no sabemos lo que nos puede llevar la comprobación del correcto funcionamiento de los distintos componentes que nos hemos puesto como meta, que en nuestro caso se tratará del funcionamiento de la señal de video del VGA, la correcta comunicación del teclado PS2 con nuestro dispositivo y finalmente la última prueba a realizar es que la memoria RAM externa se comunica correctamente con nuestra FPGA y podemos escribir en ella y leer esos datos escritos de ella.

4.2 Entorno de trabajo

Para el desarrollo del firmware hemos optado por la opción de utilizar una máquina virtual en la que podamos correr una versión de Linux denominada Lubuntu debido a que se estaba trabajando directamente desde una computadora Macintosh. En esta máquina virtual hemos asignado una memoria de video de 128MB, una memoria base de la placa base de 4096MB y 2 CPU para poder trabajar de manera holgada con nuestra máquina virtual a la hora de realizar las simulaciones pertinentes. La instalación de la máquina virtual es bastante sencilla, aunque a la hora de ponerse manos a la obra dio una serie de problemas de sincronización y funcionamiento, aunque pasado un tiempo se consiguieron arreglar y la instalación fue todo un éxito.

Una vez nos encontramos dentro de esta maquina virtual, a la hora de desarrollar nuestro firmware lo único que necesitaremos es un editor de archivos en el cuál podamos escribir nuestro código para realizar la funcionalidad que nosotros queramos implementar en nuestro proyecto. Una vez hayamos terminado de escribir el código necesitaremos un compilador para saber que nuestro código no posee ningún error y funciona correctamente en la simulación. El último se trata de sintetizar el código con nuestra FPGA para poder probar el código en nuestro producto y no en la propia simulación de la máquina virtual. Los programas utilizados serán descritos con mayor detalle en apartados posteriores. Todas las instrucciones de compilación/sintetización de nuestro código las realizaremos desde la consola de comandos que nos ofrece el sistema operativo de Linux. Por ejemplo, el comando de compilación utilizado es "make" y el comando de sintetización es "make sint".

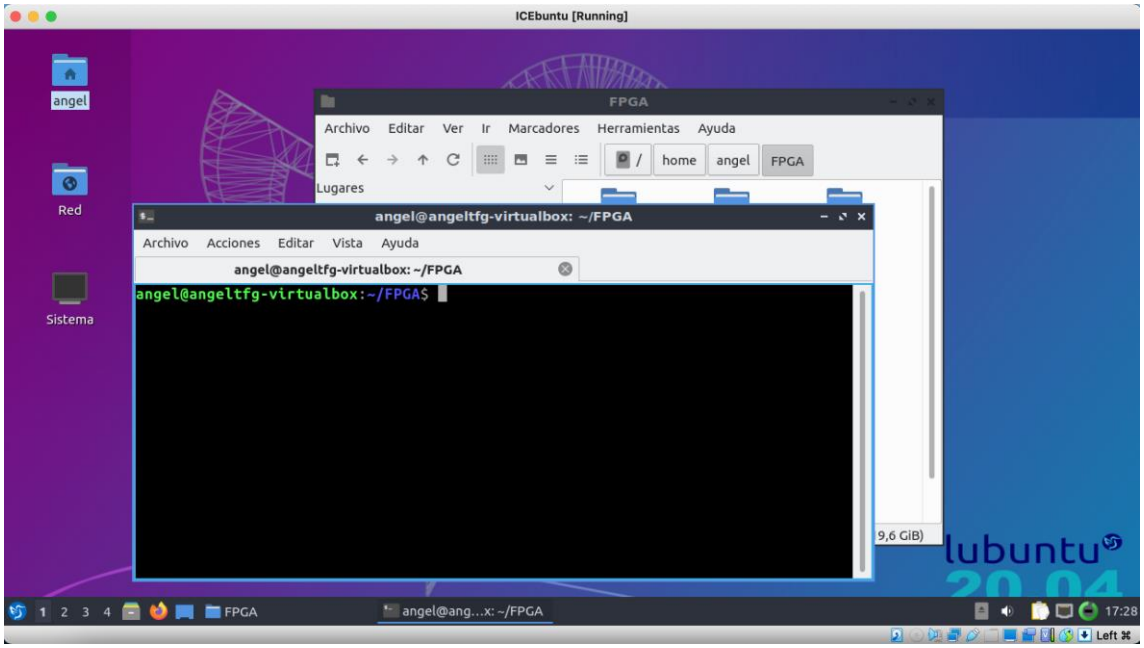


Ilustración 22. Interfaz de trabajo (Máquina Virtual).

4.3 Desarrollo principal

A continuación, pasaremos a contar más en detalle de que trataran las pruebas de validación de nuestro producto para comprobar que los distintos componentes principales que se utilizan en un simulador de videojuegos funcionan correctamente.

Las pruebas necesarias para dar esta validación van a ser distinguidas en 3 diferentes apartados, por una parte, comprobaremos que funciona correctamente la señal de video que llega al conector VGA, por otro lado, nos aseguraremos que la interconexión teclado FPGA funciona como debe y finalmente realizaremos una serie de escrituras y lecturas con su correspondiente comprobante para corroborar que nuestra memoria RAM externa funciona bien. El lenguaje de programación utilizado para la realización del firmware para el diseño de las pruebas será el lenguaje Verilog.

En primer lugar, en nuestro desarrollo tenemos distintos archivos que se interconectarán para obtener información unos de otros. En cada prueba, debemos destacar los siguientes archivos:

- Pines.pcf (PCF: Physical Constraints File): en este archivo se han definido todos los pines de la FPGA, los cuáles varios de ellos serán utilizados en nuestros diseños firmware. Como se puede ver en la figura adjunta, la asignación del pin se realiza con la sentencia "set_io" el siguiente campo es el nombre que se le da a dicho pin y el último campo es el pin correspondiente en el cuerpo de la FPGA.

- **NOMBRE_PRUEBA.v :** en el campo NOMBRE_PRUEBA le asignaremos como su propio nombre dice el nombre de la prueba correspondiente que vamos a realizar y en este archivo es donde vamos a realizar el diseño del código con las instrucciones necesarias que queremos que realice nuestro diseño.

- Main.v : en este archivo es donde instanciamos nuestros módulos creados en el archivo NOMBRE_PRUEBA.v para que queden ahí recogidos todos, junto con las entradas y salidas necesarias para la prueba. Este archivo es la base para la realización de la síntesis.

- Tb.v : aquí dentro del archivo tb, que recibe su nombre por "testbench", es donde definiremos los valores iniciales de nuestras señales de los módulos generados en el archivo NOMBRE_PRUEBA.v y también podrán ser generadas formas de onda que nos sean necesarias para realizar las pruebas necesarias en nuestra simulación. Este archivo

únicamente es utilizado en la compilación, no interviene en nada que tenga que ver con la síntesis.

- Makefile: este archivo es el que se encarga de realizar la compilación de nuestro código y en el se puede realizar una compilación simple de un único o archivo o podemos realizar una compilación de varios archivos a la vez si queremos cruzar procesos como ocurrirá en la segunda y tercera prueba de nuestro diseño.

Además de los archivos nombrados anteriormente como podemos ver, en la carpeta de cada prueba tenemos otros muchos archivos que son generados debido a las compilaciones y otros son necesarios para que los procesos de los programas funcionen como deben. Los archivos son los siguientes como podemos ver en la siguiente ilustración.

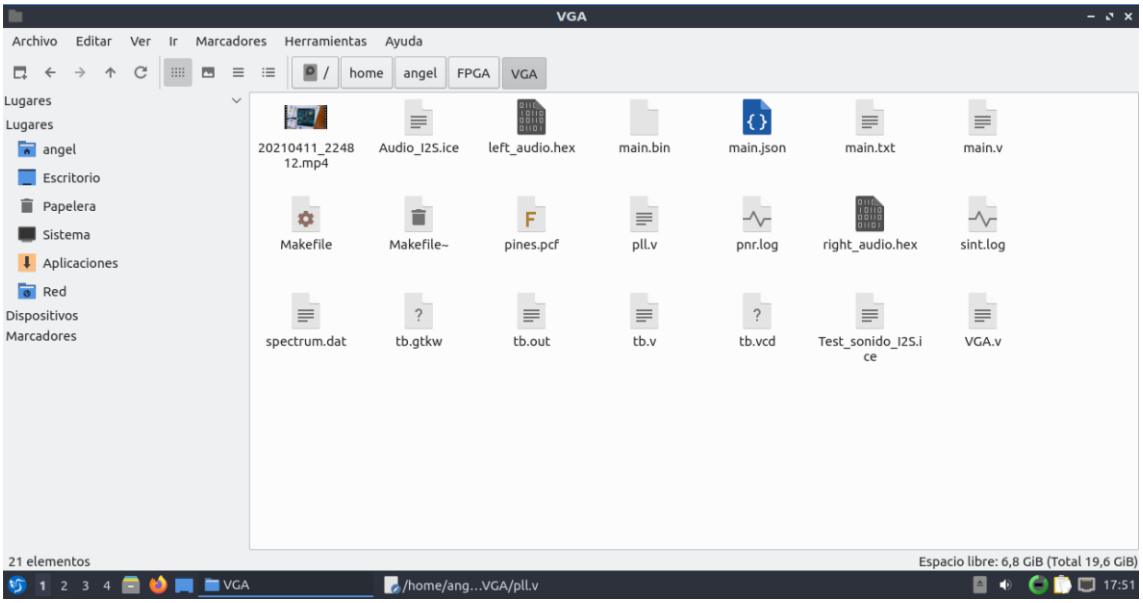


Ilustración 23. Conjunto de archivos que conforma una prueba.

4.3.1 Prueba de funcionamiento de la interfaz de video VGA

La primera de las pruebas que se implementará en nuestro prototipo es como ya hemos comentado para corroborar que el video se muestra correctamente en un monitor mediante el conector VGA. Para ello debemos programar un generador de señal de prueba para el VGA, creando la forma de onda de las señales para una resolución de 640x480 con una tasa de refresco del monitor de 60Hz. El monitor posee 2 dimensiones, horizontal y vertical. En el plano horizontal la forma de onda del VGA tiene las siguientes características: un área visible de 640 píxeles y el área no visible esta formado por las siguientes partes, comenzando por el

denominado “front porch” que consta de 16 píxeles, el “sync pulse” consta de 96 píxeles y finalmente esta área no visible termina con el denominado “back porch” que lo conforman 48 píxeles. Por lo que realizando la cuenta de 640 píxeles del monitor visibles más 96+48+16, esto nos da una cantidad de 160 píxeles en la pantalla no visibles, por lo que el tamaño total del plano horizontal es de 640+160=800 píxeles. Se han de generar 25.000.000 píxeles por segundo y eso nos obliga a utilizar un PLL de la FPGA para generar un reloj de 25MHz a partir de la señal de 16MHz del pin 21.

De la misma manera está dividido el plano vertical, pero como era de esperar las dimensiones no son la mismas en ambos planos, el área visible son 480 líneas, el “front porch” son 10 líneas, el “sync pulse” son 2 líneas y el “back porch” lo forman 33 líneas, obteniendo un número de líneas no visibles en el plano vertical de 45 y un número total de líneas de 525 líneas.

Como podemos ver en el código de nuestra prueba comenzamos definiendo los límites de los planos vertical y horizontal, generando los correspondientes contadores que han sido nombrados como “hcount” y “vcount” (de contador vertical y horizontal) para que se vaya recorriendo cada píxel de la pantalla y se reseteen cuando lleguen al extremo de cada plano.

Una vez definidos los contadores y los límites pasamos a diseñar las señales de sincronismo vertical y horizontal que son fundamentales para en el funcionamiento de nuestro VGA. Como hemos mencionado anteriormente en el plano horizontal tiene un tamaño de 96 píxeles y estas señales de sincronismo ambas son activas en baja, es decir cuando el pulso se sitúe en “0” lógico es cuando está en funcionamiento. Debido a que la señal de sincronismo entra en juego después del “front porch” y este tenía un valor de 16 píxeles se ha definido una condición de que el sincronismo horizontal valga 0 entre los valores 656 y 752 de la pantalla que son los que le corresponden en el plano horizontal, valiendo 1 en todos los demás píxeles de la pantalla.

Se procederá de la misma manera para el sincronismo vertical, pero con sus respectivos valores, valiendo “0” entre las líneas 490 y 492, y valiendo 1 en los demás.

Posteriormente a la realización de la señal de sincronismo y definidos los límites de la pantalla, pasamos a imprimir algunos datos en la pantalla dando distintos datos a las componentes de color. Se ha decidido generar un patrón de barras de colores en la pantalla que dependerá del contador horizontal. Haciendo distintas secciones de 80 píxeles, generando columnas de colores de tales dimensiones.

Para poder compilar y que se realice la simulación correctamente debemos añadir en el archivo tb.v las señales que deseamos generar que en este caso únicamente, se ha añadido el diseño de

la señal de reloj que cada 5 instantes cambie de valor lógico y una duración total de la simulación de 250.000 instantes de tiempo. Obteniendo la siguiente simulación con un valor en escalera en la componente roja en esa sección aumentada de la simulación total. La señal que se genere en la componente de color rojo en la tercera sección de la pantalla, es debido a que introducimos el valor de hcount[3:0], debido a que las componentes de color son de 4 bits, este valor introducido debido a que se trata de un contador, generará un rampa de colores o un gradiente de rojos en la pantalla, por lo que el tipo de la señal que se genera es lo esperado debido a que se forman escalones de valores en la señal como vemos en la siguiente ilustración.

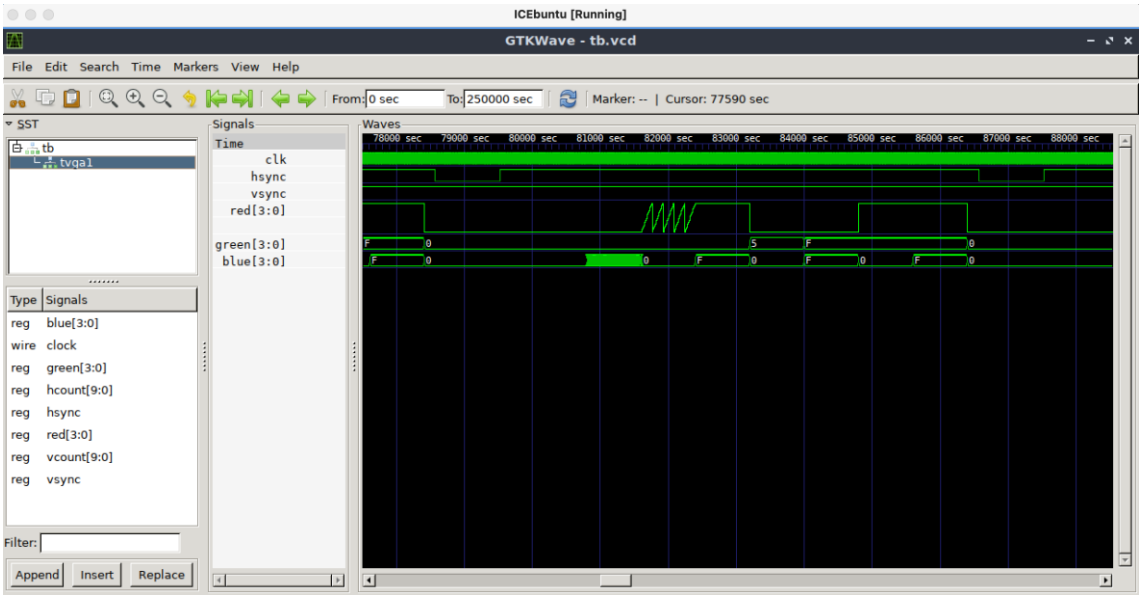


Ilustración 24. Simulación primera prueba. Visionado de ondas GTKWave.

Una vez realizada la simulación y se ha comprobado que la forma de las señales que aparecen en el visualizar de ondas del GTKWave, pasamos a realizar la síntesis de nuestro código junto con nuestro proyecto para comprobar que realmente realiza lo que queremos en la pantalla del monitor. Debido a las distintas combinaciones de valores que hemos introducido en algunas franjas, obtenemos el resultado que se ve en la siguiente ilustración.

Ilustración 25. Resultado primera prueba del prototipo (VGA).

Una vez comprobado que la señal de video funciona correctamente, pasamos a comprobar que obtenemos la misma forma de onda en las componentes de color para sección de la pantalla. Como se puede comprobar en la ilustración que estamos monitoreando en el pin de test del rojo, la señal obtenida es la siguiente y en efecto posee la misma forma de onda en la realidad que en la simulación.

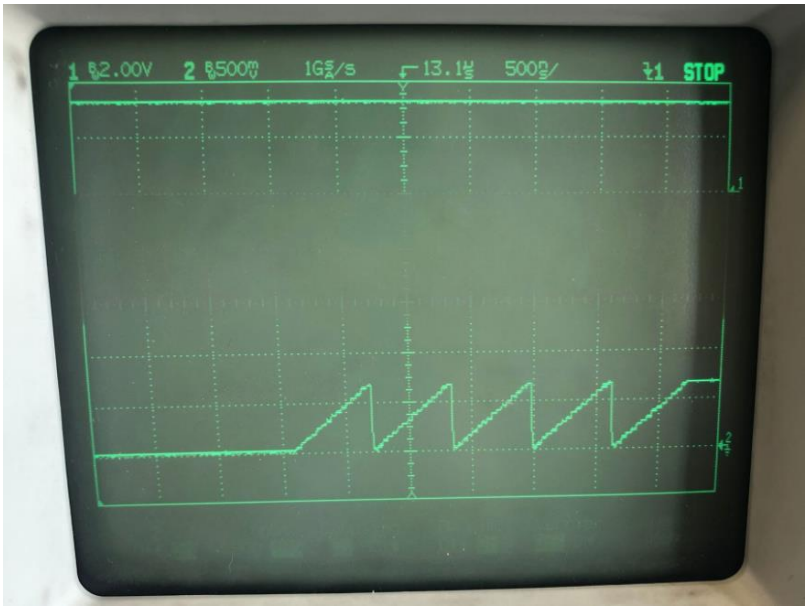


Ilustración 26. Forma de onda vista en el osciloscopio desde el pin de test RED.

A continuación, se adjunta el código completo de la primera prueba, en primer lugar, el archivo VGA.v:

```
1 //-------------------------------------------------------------------------
2 //-- Generador señal de prueba para comprobar el funcionamiento del VGA
3 //-------------------------------------------------------------------------
4
5 //Debemos generar la forma de onda de las señales para una resolución de 640x480 y
6 //una tasa de refresco de 60Hz.
7 //PLANO HORIZONTAL: tenemos un área visible de 640pixeles (área visible) +
8 //+ 16pixeles(front porch) + 96pixeles (Sync pulse) + 48pixeles(Back porch) -> 640+160=800 pixeles
9 //PLANO VERTICAL: 480 pixeles + 10pixeles(front porch) + 2pixeles (sync pulse) + 33pixeles(back porch) ->
10 //-> 480 + 45 = 525 pixeles.
11 module TEST_VGA(
12   input clock,
13   output reg [3:0] red,
14   output reg [3:0] green,
15   output reg [3:0] blue,
16   output reg hsync,
17   output reg vsync
18 );
19
20 reg [9:0] hcount = 0;
21 reg [9:0] vcount = 0;
22
23 //generamos los límites de la pantalla y añadimos los contadores.
24 always @(posedge clock)
25 begin
26     if(hcount == 799) //limite linea horizontal.
27     begin
28       hcount <= 0;
29       if(vcount == 524) //limite linea vertical.

30         vcount <= 0;
31       else
32         vcount <= vcount+1'b1; //si no hemos llegado al limite de ninguna linea seguimos contando.
33     end
34     else
35       hcount <= hcount+1'b1;
36
37  //como ambas señales son activas en baja, procedemos de la siguiente manera:
38  //la señal de sincronismo vertical posee una duracion de 2 lineas, como la zona visible termina
39  // en 480 + 10 del Front porch, la señal de sincronismo comienza en el 490 y termina en el 492.
40   if (vcount >= 490 && vcount < 492)
41     vsync <= 1'b0; //mientras este entre estos valores se mantendrá a 0 porque es activa en baja.
42   else
43     vsync <= 1'b1;
44
45  //la señal de sincronismo vertical posee una duracion de 96 lineas, como la zona visible termina
46  // en 640 + 16 del Front porch, la señal de sincronismo comienza en el 656 y termina en el 752.
47   if (hcount >= 656 && hcount < 752)
48     hsync <= 1'b0; //está igual que la vertical.
49   else
50     hsync <= 1'b1;
51
52 end
53
54 //generamos el patrón de colores que aparecerá en pantalla, dependiendo del valor del contador horizontal.
55 always @ (posedge clock)
56 begin
57     if (hcount < 80 && vcount < 480)
58     begin
```

```
59        green <= 4'b0000;
60        blue <= 4'b0000;
61        red <= 4'b0000;
62      end
63      else if (hcount < 160 && vcount < 480)
64      begin
65        green <= 4'b0000;
66        blue <= hcount[3:0]+vcount[4:1];
67        red <= 4'b0000;
68      end
69      else if (hcount < 240 && vcount < 480)
70      begin
71        green <= 4'b0000;
72        blue <= 4'b0000;
73        red <= hcount[3:0];
74      end
75      else if (hcount < 320 && vcount < 480)
76      begin
77        green <= 4'b0000;
78        blue <= 4'b1111;
79        red <= 4'b1111;
80      end
81      else if (hcount < 400 && vcount < 480)
82      begin
83        green <= vcount[4:1];
84        blue <= 4'b0000;
85        red <= 4'b0000;
86      end
87      else if (hcount < 480 && vcount < 480)
88      begin
89        green <= 4'b1111;
90        blue <= 4'b1111;
91        red <= 4'b0000;
92      end
93      else if (hcount < 560 && vcount < 480)
94      begin
95        green <= 4'b1111;
96        blue <= 4'b0000;
97        red <= 4'b1111;
98      end
99      else if (hcount < 640 && vcount < 480)
100     begin
101       green <= 4'b1111;
102       blue <= 4'b1111;
103       red <= 4'b1111;
104     end
105     else
106     begin
107       green <= 4'b0000;
108       blue <= 4'b0000;
109       red <= 4'b0000;
110     end
111 end
112 endmodule
```

Ilustración 27. Código archivo VGA.v. Primera prueba.

Además se adjunta también el archivo tb.v para saber los valores de la simulación:

```
 5 module tb();
 6
 7 //-- Registos con señales de entrada
 8 reg clk=0;
 9
10 wire [3:0] red;
11 wire [3:0] blue;
12 wire [3:0] green;
13
14 wire hsync;
15 wire vsync;
16
17 TEST_VGA tvga1(.clock(clk), .red(red), .green(green), .blue(blue), .hsync(hsync), .vsync(vsync));
18 always #5 clk=~clk;
19
20 //-- Proceso al inicio
21 initial begin
22         //-- Fichero donde almacenar los resultados
23         $dumpfile("tb.vcd");
24         $dumpvars(0, tb);
25
26         //# 319 $display("FIN de la simulacion");
27         # 250000 $finish;
28 end
29
30 endmodule
```

Ilustración 28. Código archivo tb.v. Primera prueba.

4.3.2 Prueba de funcionamiento de la interfaz del teclado PS2

La segunda prueba que se diseña en el entorno de trabajo de nuestro firmware es que la conexión con la interfaz del teclado PS2 funciona de manera correcta, comprobamos que recibimos una tecla pulsada en el teclado y para obtener una realimentación de que la tecla que hemos pulsado es la correcta, imprimiremos el valor binario de esta mediante la interfaz VGA en el monitor para corroborar que es la correcta. Para entender el funcionamiento de las señales que genera el propio teclado PS2 se adjunta la siguiente figura en la que como se puede ver, el primer bit que envía el teclado es el bit de "START" que siempre posee el valor de "0" lógico, los siguientes 8 bits se tratan de los bits de datos correspondientes a la tecla pulsada, el siguiente bit es el bit de paridad, que en nuestro diseño lo trataremos siempre como "0", es decir como paridad impar y finalmente el último bit enviado es el bit de "STOP" de fin de trama del teclado.

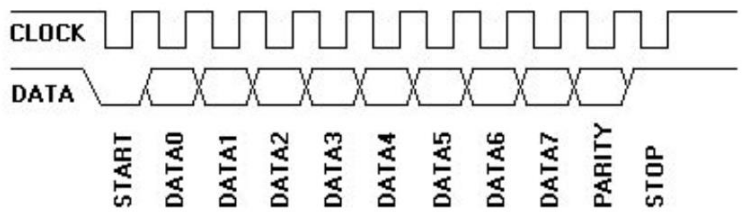
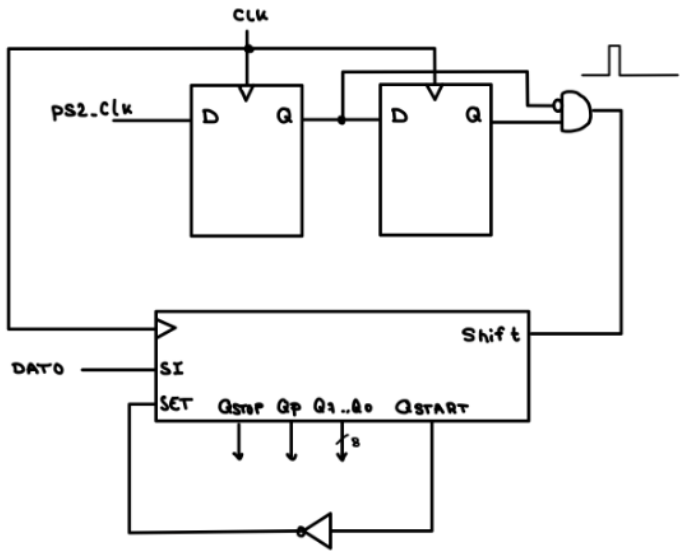


Ilustración 29. Señal generada por el teclado PS2.

Se añade a continuación un esquema que nos ayudará a seguir el diseño del firmware, donde veremos los distintos bloques que conforman nuestra prueba y los pasos a seguir para diseñarla

correctamente. En primer lugar, añadimos dos registros que se tratan de flip-flop tipo D en los que la señal de datos será el reloj del propio teclado para evitar posibles variaciones indeseadas en esta señal que puedan hacer confundir a nuestro prototipo y de esa manera no funcione como queremos nuestro diseño. Estos dos flip-flop son controlados por la señal de reloj global del sistema y a la salida del segundo flip-flop añadimos una puerta AND invirtiendo la entrada de la salida del primer flip-flop, de manera que esto actúe como detector de los flancos de bajada del reloj del teclado.



Esquema 1. Segunda prueba.

Una vez asignada la señal de desplazamiento, pasamos a realizar el diseño de nuestro registro de desplazamiento, que tiene como entrada serie el dato de la tecla pulsada, posee una señal de set para inicializar todos los bits de salida del registro a 1 cuando el valor del primer bit recibido de la trama, como se trata del bit de "START" y este siempre vale 0, lo invertiremos para así activar nuestra señal de set y la última entrada es la de la señal de desplazamiento. Como salida tenemos el dato en paralelo que recogeremos para imprimirlo mediante la señal de video posteriormente. Únicamente imprimiremos por pantalla los bits [8:1] del dato debido a que son los únicos que nos interesan. Como se puede observar en el siguiente código tenemos la realización de lo explicado anteriormente, en el archivo VGA.v:

Una vez se ha diseñado el código del VGA pasamos a realizar el testbench correspondiente para conseguir en la simulación lo que buscamos. Esta vez diseñaremos la forma de onda del teclado ps2, que posee unos valores de frecuencia mucho menores que la del reloj de nuestro sistema que posee 25MHz, el reloj oscila entre 10-15kHz. Escogiendo una frecuencia de funcionamiento de por ejemplo 10kHz, obtenemos un periodo de 100us y un semiperiodo de 50us, por lo que se

pensó que un buen valor de retardo a la hora de introducir los valores de la señal que vamos a generar en el testbench sería de 12.500 unidades de tiempo, para obtener un periodo de la señal del reloj del ps2 de 25.000 unidades, para que le diera tiempo de sobra al dato a ser añadido correctamente. El dato es escrito cuando la señal del ps2 se sitúa en nivel bajo.

Procediendo con la simulación en el visor de ondas podemos como el valor de la salida del registro de desplazamiento comienza siendo 7FF debido a que la señal de set ha sido activada por el flag valiendo 1, ese es nuestro punto de partida. Y como se puede ver en la ilustración el valor de los bits de salida se ve modificado en los flancos de bajada del reloj del ps2 y también podemos observar como se van desplazando los bits con cada pulso de la señal de desplazamiento por el registro hasta que finalmente han llegado todos los bits de la trama y se vuelve a activar el flag haciendo que el valor de los bits de salida del registro, vuelvan a ser todos 1. Se adjunta una captura de la simulación descrita en este párrafo a continuación:

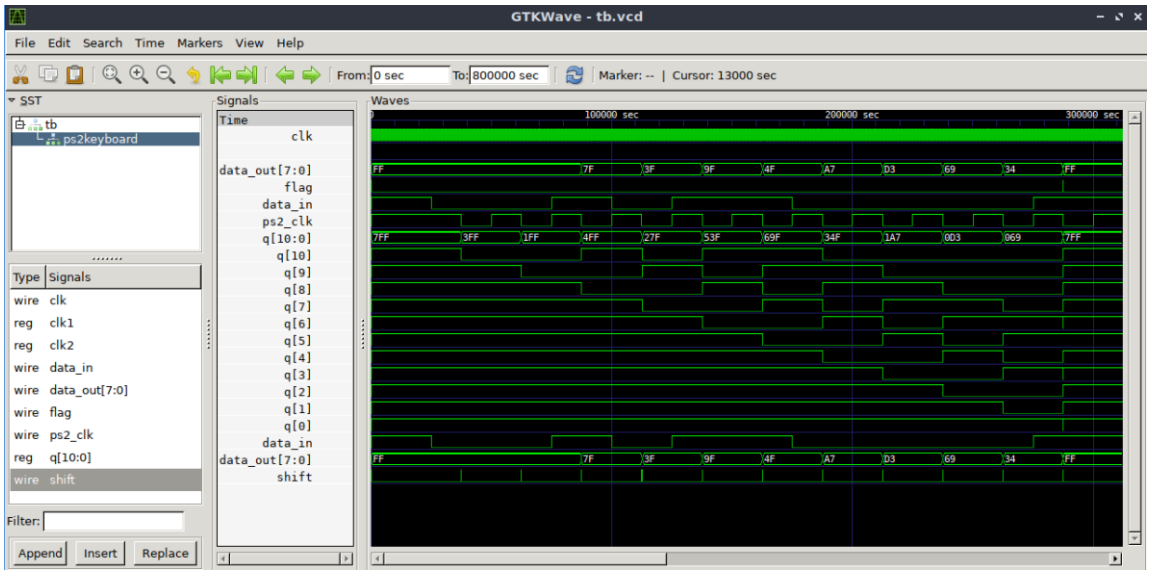


Ilustración 30. Simulación segunda prueba. Visionado de ondas GTKWave

Una vez comprobado que en la simulación obtenemos lo que buscábamos, procedemos a realizar la síntesis de los archivos del ps2 y del VGA, para que podamos introducir nuestro parámetro de salida del ps2 como parámetro de entrada del VGA y poder imprimirlo por pantalla para poder tener esa realimentación de la que hablábamos anteriormente. Realizando la síntesis obtenemos el siguiente resultado de la prueba, se adjuntan únicamente las pruebas de 2 teclas pulsadas, pero se comprobaron la mayoría de teclas del teclado y funcionaban todas a la perfección. Para poder identificar que valor tiene cada tecla, en el datasheet correspondiente

del PS2, se puede encontrar una tabla con la conversión de código de cada tecla, que se adjunta a continuación, tras los resultados obtenidos en la prueba, que se han introducido en el color rojo en la segunda franja de la pantalla:

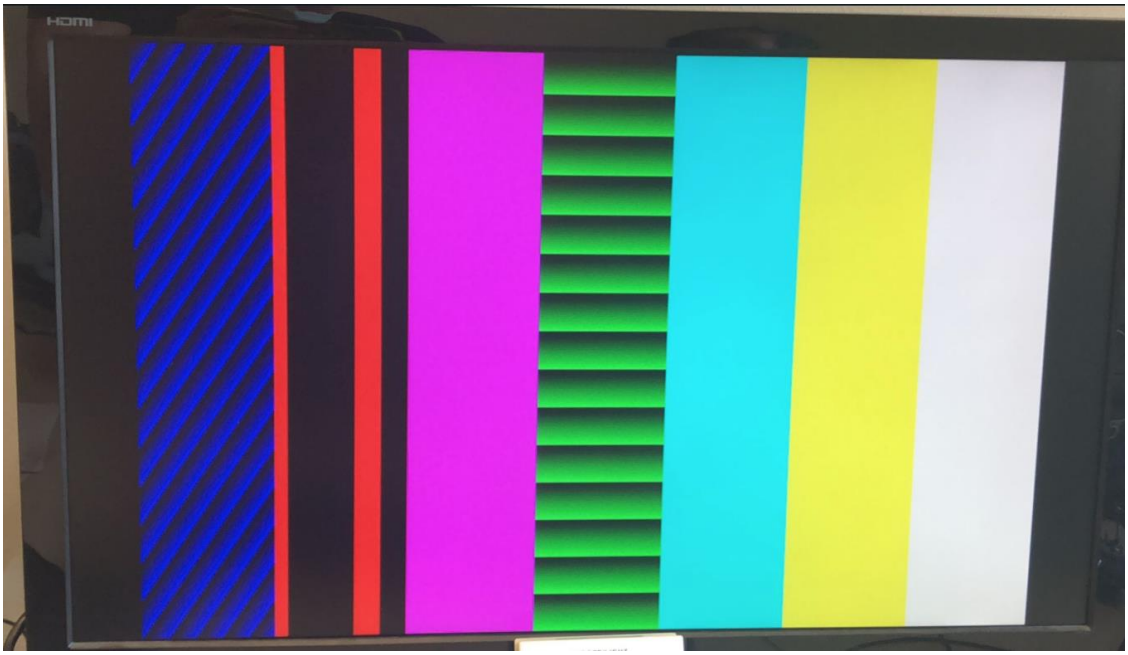


Ilustración 31. Resultado segunda prueba del prototipo (PS2-VGA). Tecla pulsada F7.

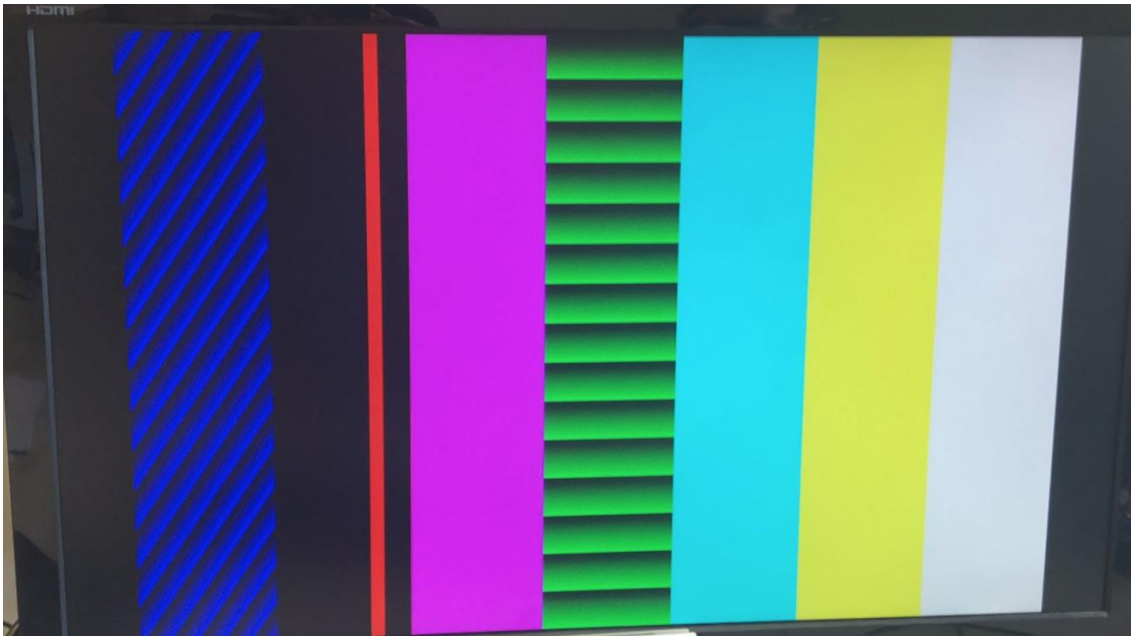


Ilustración 32. Resultado segunda prueba del prototipo (PS2-VGA). Tecla pulsada F9.

Como podemos comprobar los valores que nos aparecen en el monitor son los correctos debido a que la tecla F7, si la pulsamos nos envía el código 83 en hexadecimal que se corresponde con 10000011 que como podemos ver se imprime correctamente. Y la tecla F9

corresponde con el código 01, que como podemos ver está correcto debido a que se imprime una pequeña franja roja al final simulando ese 1 binario.

Keyboard Scan Codes: Set 2

*All values are in hexadecimal

101-, 102-, and 104-key keyboards:

KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK	-----	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

Ilustración 33. Códigos de las teclas. Teclado PS2.

A continuación, se adjunta el código completo de los archivos involucrados en el desarrollo de esta segunda prueba. Comenzando por el archivo PS2.v:

```
1 //----------------------------------------------------------------------------
2 //-- Generador señal de prueba para comprobar el funcionamiento del PS2 (recepción de dato)
3 //----------------------------------------------------------------------------
4
5
6 //El teclado escribe un bit en la linea de datos cuando el reloj está en alto y
7 //el host lo recibe cuando el reloj está en bajo.
8
9 //Creamos un modulo de un FF tipo D para controlar que el dato salga con la frecuencia del reloj
10 module ps2_scan(
11         input clk,
12         input ps2_clk,
13         input data_in,
14         output [7:0] data_out,
15         output flag
16 );
17
18 reg [10:0] q=11'h7ff;
19 reg clk1;
20 reg clk2;
21
22 always @ (posedge clk)
23 begin
24         clk1<=ps2_clk;
25         clk2<=clk1;
26 end
27
28 assign shift= clk2 & (~clk1);
29
30
31 always @ (posedge clk)
32 begin
33         if(flag) q<=11'h7FF;
34         else if(shift)
35                 q<= {data_in, q[10:1]};
36 end
37
38 assign data_out=q[8:1];
39 assign flag=~q[0];
40
41 endmodule
```

Ilustración 34. Código archivo PS2.v. Segunda prueba.

El archivo VGA.v:

```
11 module TEST_VGA(
12   input clock,
13   input [7:0] data_in,
14   output reg [3:0] red,
15   output reg [3:0] green,
16   output reg [3:0] blue,
17   output reg hsync,
18   output reg vsync
19 );
20
21 reg [9:0] hcount = 0;
22 reg [9:0] vcount = 0;
23
24 reg [7:0] dsh;
25
26 //generamos los límites de la pantalla y añadimos los contadores.
27 always @(posedge clock)
28 begin
29     if(hcount == 799) //limite linea horizontal.
30     begin
31       hcount <= 0;
32       if(vcount == 524) //limite linea vertical.
33         vcount <= 0;
34       else
35         vcount <= vcount+1'b1; //si no hemos llegado al limite de ninguna linea seguimos contando.
36     end
37     else
38       hcount <= hcount+1'b1;
39
40  //como ambas señales son activas en baja, procedemos de la siguiente manera:
41  //la señal de sincronismo vertical posee una duracion de 2 lineas, como la zona visible termina
42  // en 480 + 10 del Front porch, la señal de sincronismo comienza en el 490 y termina en el 492.
43   if (vcount >= 490 && vcount < 492)
44     vsync <= 1'b0; //mientras este entre estos valores se mantendrá a 0 porque es activa en baja.
45   else
46     vsync <= 1'b1;
47
48  //la señal de sincronismo vertical posee una duracion de 96 lineas, como la zona visible termina
49  // en 640 + 16 del Front porch, la señal de sincronismo comienza en el 656 y termina en el 752.
50   if (hcount >= 656 && hcount < 752)
51     hsync <= 1'b0; //está igual que la vertical.
52   else
53     hsync <= 1'b1;
54
55 end
56
57 //generamos el patrón de colores que aparecerá en pantalla, dependiendo del valor del contador horizontal.
58 always @ (posedge clock)
59 begin
60     if (~hsync) dsh<=data_in;
61     if (hcount < 80 && vcount < 480)
62     begin
63       green <= 4'b0000;
64       blue <= 4'b0000;
65       red <= 4'b0000;
66     end
```

```
67      else if (hcount < 160 && vcount < 480)
68      begin
69        green <= 4'b0000;
70        blue <= hcount[3:0]+vcount[4:1];
71        red <= 4'b0000;
72      end
73      else if (hcount < 240 && vcount < 480)
74      begin
75        green <= 4'b0000;
76        blue <= 4'b0000;
77        red <= dsh[7]?4'b1111:4'b0000;
78        if(hcount[2:0]==4'b111) dsh<={dsh[6:0],1'b0};
79      end
80      else if (hcount < 320 && vcount < 480)
81      begin
82        green <= 4'b0000;
83        blue <= 4'b1111;
84        red <= 4'b1111;
85      end
86      else if (hcount < 400 && vcount < 480)
87      begin
88        green <= vcount[4:1];
89        blue <= 4'b0000;
90        red <= 4'b0000;
91      end
92      else if (hcount < 480 && vcount < 480)
93      begin
94        green <= 4'b1111;
95        blue <= 4'b1111;
96        red <= 4'b0000;
97      end
98      else if (hcount < 560 && vcount < 480)
99      begin
100       green <= 4'b1111;
101       blue <= 4'b0000;
102       red <= 4'b1111;
103     end
104     else if (hcount < 640 && vcount < 480)
105     begin
106       green <= 4'b1111;
107       blue <= 4'b1111;
108       red <= 4'b1111;
109     end
110     else
111     begin
112       green <= 4'b0000;
113       blue <= 4'b0000;
114       red <= 4'b0000;
115     end
116 end
117 endmodule
```

Ilustración 35. Código archivo VGA.v. Segunda prueba.

Finalmente se adjunta el tb.v para saber los datos iniciales de la simulación:

```
1 //------------------------------------------------------------------
2 //-- Banco de pruebas
3 //------------------------------------------------------------------
4
5 module tb();
6
7 //-- Registos con señales de entrada
8 reg clk=0;
9 reg ps2_clk=0;
10 reg data_in=0;
11 wire flag_recepcion;
12 wire [7:0] data_out;
13
14 ps2_scan ps2keyboard(.clk(clk), .flag(flag_recepcion), .ps2_clk(ps2_clk), .data_in(data_in), .data_out(data_out));
15
16 always #5 clk=~clk;
17
18 //-- Proceso al inicio
19 initial begin
20         //-- Fichero donde almacenar los resultados
21         $dumpfile("tb.vcd");
22         $dumpvars(0, tb);
23
24         clk = 1;
25         data_in = 1;
26         ps2_clk = 1;
27         #25000
28         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
29         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
30         data_in = 1; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
31         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
32         data_in = 1; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
33         data_in = 1; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
34         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
35         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
36         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
37         data_in = 0; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
38         data_in = 1; #12500 ps2_clk = 0; #12500 ps2_clk = 1;
39
40
41         //# 319 $display("FIN de la simulacion");
42         #500000 $finish;
43 end
44
45 endmodule
```

Ilustración 36. Código archivo tb.v. Segunda prueba.

4.3.3 Prueba de funcionamiento de la memoria RAM externa y conector DB9.

Para dar por finalizado el proyecto de diseño del dispositivo tanto del hardware como del software, pasamos a diseñar el código correspondiente a la tercera y última prueba, que consistirá en probar el correcto funcionamiento de la memoria RAM externa que hemos añadido al producto. Esto lo comprobaremos mediante la escritura de datos en ella y posteriormente leyendo los datos escritos y teniendo como realimentación nuestro conector VGA para imprimir los datos leídos de la memoria en la pantalla.

Por otro lado, para comprobar el funcionamiento del conector DB9 y a su vez que sirva como mando para realizar ciertas acciones como escribir o borrar la memoria RAM, se ha diseñado un pequeño mando que consta de dos botones, con sus correspondientes conexiones y resistencias

y además tenemos 2 leds para saber si el dato ha sido borrado (led rojo) y para que se sepa si el dato que se está escribiendo es el mismo que está siendo leído de la memoria RAM.

Como en la prueba anterior se añade un esquema que nos ayudará a entender de una manera más sencilla el funcionamiento de nuestro diseño firmware, haciendo el recorrido por este mucho más visual. Las señales de estado y zero serán controladas por el flanco de subida de la señal vsync, estado nos marca si estamos en estado de escritura si este tiene un valor lógico de 1 y si tiene un valor lógico de 0 estaremos en el estado de lectura. La señal de "zero" nos indica que a la memoria se le asignará todo valores de 0 para ser borrada y este le será asignado al botón marrón. La señal de estado se le asigna a la salida OE que esta señal es la que nos habilita la salida de los datos. A la variable "xa" es a la que se le será asignada los valores de las direcciones en las que serán escritos los datos y que posteriormente imprimiremos los valores de estas direcciones en la pantalla para saber si estamos leyendo bien los datos de la memoria. Además, se le asignará la señal de WE (Write Enable) que es la señal que nos permite escribir los datos en la memoria al botón blanco mediante la realización de la puerta NAND con la entrada de estado y la señal de reloj negada. Indicar que cabe mencionar que las instancias SB_IO de main.v para el bus de datos bidireccional de la SRAM xBHE y xBLE estaban siempre activas (datos de 16 bits).

Dentro del bucle principal del desarrollo, realizamos un apartado en el que, si la señal vsync posee un valor de 0, el valor de xa pasará a ser 0 y la señal que controlará el encendido del led estará encendida. En cambio, si la señal vsync se encuentra en 1 lógico, el valor de las direcciones se ve incrementado actuando como un contador y si el dato de entrada no coincide con el valor de las direcciones no se encenderá el led.

La última asignación que hacemos en nuestro código significa que si pulsamos el botón marrón el valor de salida del módulo serán todos los datos 0 y se borrará la memoria y seguirá saliendo el valor de las direcciones de memoria que serán impresas en la pantalla.

En este apartado, apenas podemos realizar simulaciones, únicamente podemos visualizar los valores que adoptarán las variables xa y comprobar que se escribirán los datos de salida cada vez que la señal WE valga 1. Como tenemos activada la señal de OE continuamente los datos pasarán constantemente a la salida de nuestra memoria.

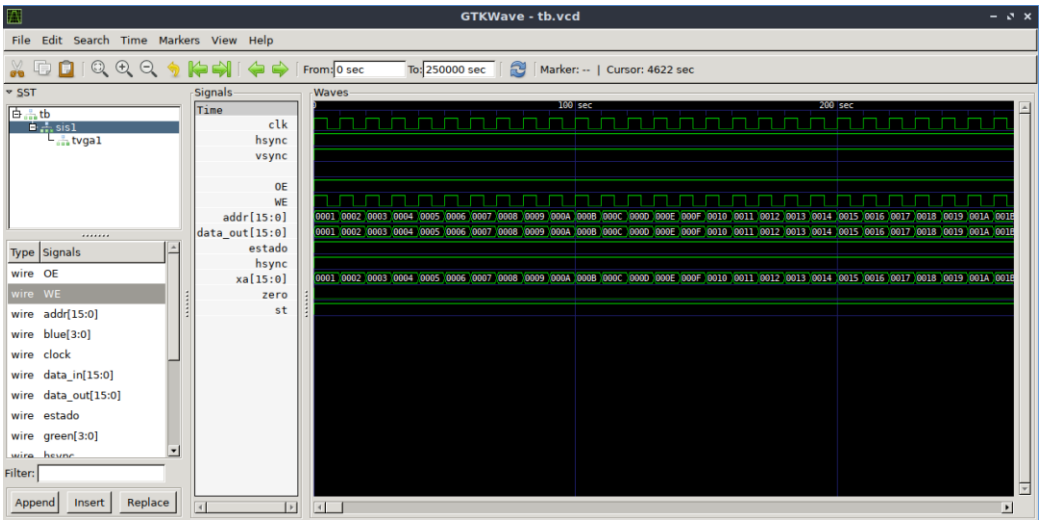


Ilustración 37.Simulación tercera prueba. Visionado de ondas GTKWave

Observando que la simulación pasa los datos correctamente por los registros cuando debe pasamos a realizar la síntesis junto al archivo VGA.v en donde únicamente en vez de imprimir las barras de colores que realizábamos en pruebas anteriores únicamente se van a imprimir valores directamente correspondientes a los datos que nos vienen de la memoria. Tras realizar la síntesis obtenemos los siguientes resultados. Como podemos comprobar en la primera ilustración se imprimen correctamente los valores de las direcciones en nuestro monitor mediante el VGA y en la segunda ilustración adjunta, podemos ver como se enciende el led verde debido a que está leyendo constantemente datos de la memoria y está comprobando continuamente que los valores son los mismos que se han escrito.

Ilustración 38.Resultado tercera prueba del prototipo. Impresión direcciones. (VGA-DB9-RAM).

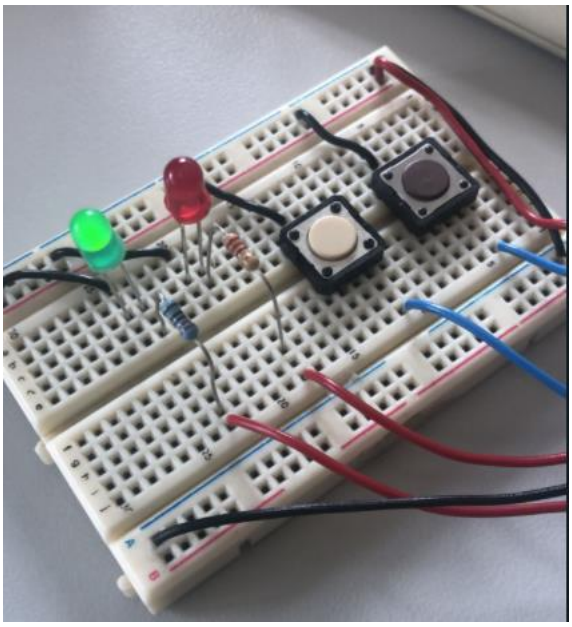


Ilustración 39.Resultado tercera prueba del prototipo. LED verde encendido. (VGA-DB9-RAM)

Si en cambio se pulsa el botón marrón de nuestro joystick que va conectado al conector DB9, como se puede ver la pantalla se queda en negro completamente debido a que se borran todos los datos de la memoria y se imprimen 0 en la pantalla. Y por consiguiente el led verde no se enciende debido a que el dato escrito y el dato leído no es el mismo debido a que se están escribiendo 0 en la pantalla.

Ilustración 10. Resultado tercera prueba del prototipo. Borrado de pantalla. (VGA-DB9-RAM)

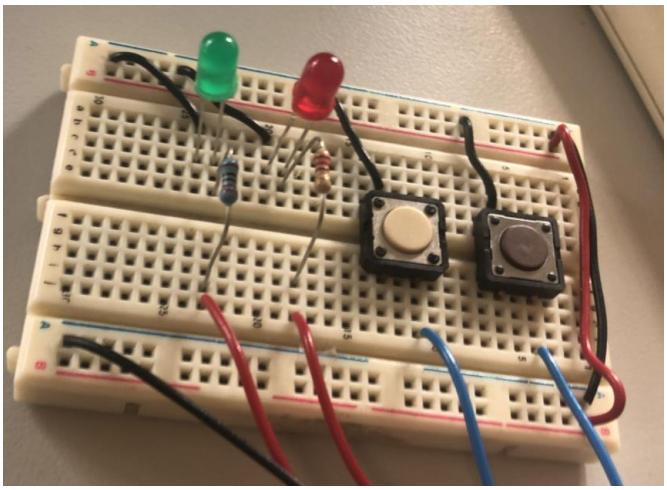


Ilustración 11.Resultado tercera prueba del prototipo. LED verde apagado. (VGA-DB9-RAM)

Finalmente se adjunta el montaje completo que hemos utilizado para la realización de esta tercera y última prueba.

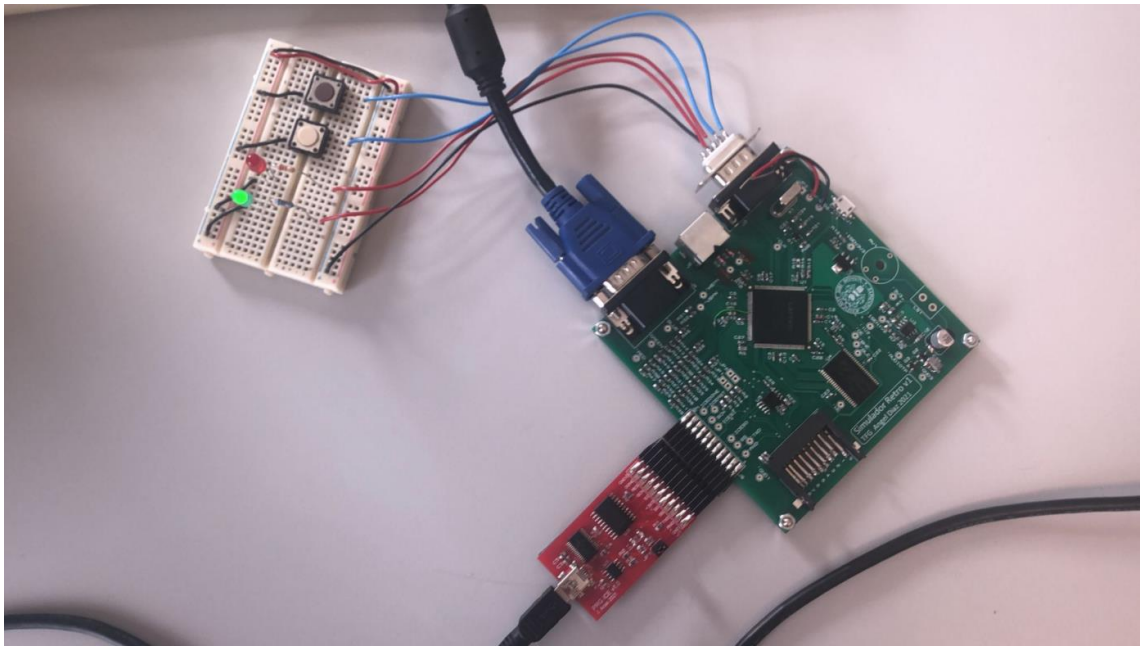


Ilustración 12. Montaje completo tercera prueba.

Para finalizar con esta prueba se añade el código completo utilizado para llevar a cabo esto:

Archivo system.v:

```
module sistema(
  input clock,
  output  [3:0] red,
  output  [3:0] green,
  output  [3:0] blue,
  output  hsync,
  output  vsync,

  input [15:0] addr,
  input [15:0] data_in,
  output WE,
  output OE,
  output [15:0] data_out,
  input estado,
  input zero,
  output okey
);


TEST_VGA tvga1(.clock(clock), .red(red), .green(green), .blue(blue), .hsync(hsync), .vsync(vsync), .data_in(data_in));


reg [7:0]tecla;
reg st;
reg zr;
reg okey;
//wire estado;

reg [15:0] xa=0;
always @ (posedge vsync) st<=estado;
always @ (posedge vsync) zr<=zero;
assign addr=xa;

assign WE= ~(st & (~clock));


//si estado=1 -> nos encontramos en el estado de escritura; estado=0->lectura.

assign OE=st; //ya que es activo en baja.
//generamos el contador de direcciones

always @ (posedge clock)
        begin
                if(~vsync) begin
                        xa<=0;
                        okey<=1;
                end else begin
                        xa<=xa+1;
                        if(xa!=data_in) okey<=0;
                end


        end

assign data_out=zr?16'h0:xa;

endmodule
```

Ilustración 13. Código archivo system.v. Tercera prueba.

Archivo tb.v:

```
5  module tb();
6
7  //-- Registos con señales de entrada
8  reg clk=0;
9
10 wire [3:0] red;
11 wire [3:0] blue;
12 wire [3:0] green;
13
14 wire hsync;
15 wire vsync;
16
17 reg data_in=0;
18 wire [7:0] data_out;
19 reg zero=0;
20
21 sistema sis1(.clock(clk), .red(red), .green(green), .blue(blue), .hsync(hsync), .vsync(vsync),
22 .addr(addr), .data_in(data_in), .WE(WE), .OE(OE), .data_out(data_out), .estado(1),.zero(zero));
23 always #5 clk=~clk;
24
25
26 //-- Proceso al inicio
27 initial begin
28
29         //-- Fichero donde almacenar los resultados
30         $dumpfile("tb.vcd");
31         $dumpvars(0, tb);
32
33         clk = 1;
34         data_in = 1;
35
36
37         //# 319 $display("FIN de la simulacion");
38         # 250000 $finish;
39 end
40
41 endmodule
42
43
```

Ilustración 14. Código archivo tb.v. Tercera prueba.

5. Herramientas para la evaluación del producto

Una buena manera de ir comprobando en cada etapa el funcionamiento del equipo y buscar fuentes de posibles errores en nuestro desarrollo se ha empleado principalmente nuestro compilador que se ha incorporado en nuestra máquina virtual de Lubuntu. Gracias a esto podemos comprobar los errores de código que tenemos a la hora de desarrollarlo debido a que si no solucionamos estos errores la simulación en el programa GTKWave no comienza. Pasamos a comentar las principales funcionalidades que hemos utilizado durante el desarrollo de las pruebas analizadas previamente.

5.1 Gtkwave

Es uno de los mejores visores de onda libres es parte de la distribución Debian y en nuestro desarrollo ha sido utilizado principalmente tras la simulación de nuestras pruebas para comprobar que las formas de onda que estamos generando en el código son las correctas y que se cumplen correctamente las temporizaciones de las señales que hemos definido y además se cumplen los periodos que tenemos para las distintas implementaciones realizadas.

Este software da una gran libertad a la hora de navegar por las distintas ondas que hemos generado, además de poder hacer zoom en zonas que es muy necesario, debido a que a simple vista no podríamos ver el resultado correcto y también nos permite la utilización de marcadores para poder ver valores de periodos o lo que queramos.

En la parte izquierda del programa aparece la jerarquía de programas que tenemos dentro de nuestra simulación y debajo de estos están situadas las señales que podemos ir introduciendo

en el visor de ondas las señales que se quieran, como ya hemos podido observar en capturas anteriores que hemos añadido de las simulaciones de este programa.

5.2 Instrumentación electrónica

El osciloscopio ha sido de gran utilidad a la hora de comprobar que las señales que hemos obtenido en el visor de señales GTKWave a la hora de realizar la síntesis con nuestra FPGA, poseen la misma forma a la hora de ejecutarlo en nuestro diseño. Como hemos podido ver en las pruebas anteriores nos ha ayudado a comprobar la forma de onda que poseen las componentes de color del VGA.

A su vez el multímetro ha sido también un instrumento utilizado a la hora del montaje de los componentes de nuestro dispositivo para comprobar que no se haya generado ningún cortocircuito en nuestro diseño y pueda afectar al correcto funcionamiento del prototipo.

6. Conclusiones

Para finalizar con el desarrollo de este proyecto, vamos a poner sobre la mesa distintos puntos como los objetivos que se han alcanzado en este proyecto, problemas encontrados y posibilidades de mejora para un futuro.

Para comenzar los objetivos que han sido alcanzados en el prototipo al final han sido en menor medida de los que se esperaba, debido a que inicialmente se realizó una estrategia primaria que se pensaba que se iba a tener el tiempo suficiente para la realización completa de nuestro simulador de videojuegos, pero al final, no ha podido ser así y únicamente lo hemos dejado como el diseño hardware y corroborar que las distintas partes más importantes de nuestro diseño funcionan como deben. No es para nada ningún fracaso debido a que este proyecto, aunque se realizó esa estrategia inicial, a medida que pasaba el tiempo se iban tomando nuevas decisiones que apuntaban hacia otras metas en función de la disponibilidad y la dificultad de los problemas planteados.

Los problemas que han sido encontrado a la hora de realizar el prototipo han sido en la medida de lo posible ínfimos que han podido ser corregidos al momento en el que se percataba de la situación ocurrida, para que el producto pudiera seguir saliendo a la luz como debía.

Finalmente, ya para cerrar este trabajo de fin de grado, las posibilidades de mejora para un futuro, pueden ser la implementación con la posterior simulación de algunos videojuegos que estaban pensados para este dispositivo y poder sincronizar un joystick con el que podamos disfrutar de la experiencia de juego del producto.

7. Documentación

7.1 Bibliografía

[1] *Circuit Digest. 2021. Introduction to FPGA and It's Programming Tools. [online] Available at: https://circuitdigest.com/tutorial/what-is-fpga-introduction-and-programming-tools*

[2] *FPGA | Field Programmable Gate Array | Introduction, Structure. (2021). From https://www.electronicshub.org/introduction-to-fpga/*

[3] *VGA Signal Timing. (2021). from http://www.tinyvga.com/vga-timing*

[4] *Definitions, V., & Hope, C. (2021). What is VGA (Video Graphics Array). From https://www.computerhope.com/jargon/v/vga.htm*

[5] *LLC, U. (2021). RS232 Pinout | DB9 pinout. From http://www.usconverters.com/index.php?main_page=page&id=61*

[6] 2021. *The PS/2 Mouse/Keyboard Protocol*. [ebook] Available at: http://www.computer-engineering.org/ps2protocol/

[7] *2021. iCE40 LP/HX Family Data Sheet. [ebook] Available at: <http://www.latticesemi.com/>*

[8] *Electronicavm.files.wordpress.com. 2021. [online] Available at: https://electronicavm.files.wordpress.com/2011/03/oscilador-a-cristal.pdf*

[9] *Ieec.uned.es. 2021. [online] Available at: <http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Tecnologia_SRAM.pdf>*

[10] *Pages.hmc.edu. 2021. [online] Available at: http://pages.hmc.edu/harris/class/e158/04/lect13.pdf*

[11] *2021. Datasheet CY7C1021DV33 1-Mbit (64 K × 16) Static RAM.pdf*

[12] *2021. LD1117 series Low drop fixed and adjustable positive voltage regulators. p.www.st.com.*

[13] *2021. Datasheet 10 mm carbon potentiometer PT10.*

[14] *2021. PROTEUS DESIGN SUITE. Primeros pasos con la pestaña Diseño PCB. [ebook] Available at: <http://www.labcenter.com>*

[15] *Nasir, S., 2021. Complete Guide on Proteus ISIS & ARES - The Engineering Projects. [online] The Engineering Projects. Available at: <https://www.theengineeringprojects.com/2013/03/a-complete-tutorial-on-how-to-use-proteus-isis-ares.html>*