



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

DESARROLLO E IMPLEMENTACIÓN DE UN
SISTEMA PARA PROCESAMIENTO DE
IMÁGENES MEDIANTE EL USO DE UN
DISPOSITIVO FPGA

TESIS

Que para obtener el título de

INGENIERO EN TELECOMUNICACIONES SISTEMAS Y
ELECTRÓNICA

PRESENTAN:

José Alejandro Maya Camacho

Rodrigo Santiago Rocha Varela

ASESOR:

Ing. José Luis Barbosa Pacheco

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO, 2020



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
ASUNTO: VOTO APROBATORIO

M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLÁN
PRESENTE

ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: Trabajo de Tesis

Desarrollo e implementación de un sistema para procesamiento de imágenes mediante el uso de un dispositivo FPGA.

Que presenta el pasante: JOSÉ ALEJANDRO MAYA CAMACHO
Con número de cuenta: 31201567-7 para obtener el Título de la carrera: Ingeniería en Telecomunicaciones, Sistemas y Electrónica

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 12 de marzo de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Ing. Noemi Hernández Dominguez	
1er. SUPLENTE	Mtro. Leopoldo Martín del Campo Ramírez	
2do. SUPLENTE	Ing. Jorge Alberto Vázquez Maldonado	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).



UNIVERSIDAD NACIONAL
AUTÓNOMA DE
MÉXICO

**FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
SECRETARÍA GENERAL
DEPARTAMENTO DE EXÁMENES PROFESIONALES**

U.N.A.M.
FACULTAD DE ESTUDIOS
ASUNTO: VOTO APROBATORIO

**M. en C. JORGE ALFREDO CUÉLLAR ORDAZ
DIRECTOR DE LA FES CUAUTITLÁN
PRESENTE**

**ATN: I.A. LAURA MARGARITA CORTAZAR FIGUEROA
Jefa del Departamento de Exámenes Profesionales
de la FES Cuautitlán.**

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos comunicar a usted que revisamos el: Trabajo de Tesis

Desarrollo e implementación de un sistema para procesamiento de imágenes mediante el uso de un dispositivo FPGA.

Que presenta el pasante: RODRIGO SANTIAGO ROCHA VARELA
Con número de cuenta: 31218871-7 para obtener el Título de la carrera: Ingeniería en Telecomunicaciones, Sistemas y Electrónica

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 21 de enero de 2020.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	Mtro. Jorge Buendía Gómez	
VOCAL	Ing. José Luis Barbosa Pacheco	
SECRETARIO	Ing. Noemi Hernández Dominguez	
1er. SUPLENTE	Mtro. Leopoldo Martín del Campo Ramírez	
2do. SUPLENTE	Ing. Jorge Alberto Vázquez Maldonado	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

AGRADECIMIENTOS

A mis padres Ma. Soledad Varela y José Pablo Rocha Jiménez:

Les agradezco por todo el amor incondicional y la confianza que me otorgaron. Gracias ustedes, soy la persona que hoy en día puede enfrentar todo tipo de problemas que se llegaran a presentar.

Espero que puedan observar a primera fila, todos los planes que se tienen para el futuro. Porque esos logros no solo serán míos, sino también suyos. Yo sé que aún me falta mucho por aprender. Pero sus enseñanzas, me guiarán por el buen camino.

A Humberto Rocha Varela y José Pablo Rocha Varela:

A mis hermanos, que han sido mis compañeros de toda la vida. Les agradezco por su cariño, experiencias y su protección que siempre me compartieron. Les tengo mucho respeto y admiración.

A mi profesor José Luis Barbosa Pacheco:

Siempre nos mostró el camino del conocimiento, de la curiosidad y de ir más allá de lo que nos ofrecen en el colegio. Alimentando nuestras ideas con sus sabios consejos que nos guiaron por la senda del estudio.

A mis compañeros:

Junto a los que emprendí el feliz camino de la vocación y a los cuales agradezco la estimación que me brindaron.

A mi Facultad de Estudios Superiores Cuautitlán:

Para todas aquellas personas que de alguna manera participaron en la elaboración de este trabajo.

Rodrigo Santiago Rocha Varela

Primeramente, quiero agradecer a mi familia, ya que sin ellos este trabajo no hubiera sido posible. A mi madre quien siempre ha estado allí cuando la he necesitado, por haber trabajado arduamente para que no me faltara nada, por todos esos consejos y el tiempo que dedicó para hacerme sentir mejor en mis peores momentos. A mi padre que siempre ha sido mi guía y un ejemplo que seguir, por inculcarme el gusto a estudiar y por todos los regaños que en su momento no entendía, pero hoy, cobran sentido al ver lo que he logrado. Por supuesto a mi abuelita, por haberme cuidado mientras mis padres trabajaban, por enseñarme modales y valores, además de siempre haberme mostrado un cariño incondicional. No hay palabras para agradecer todo lo que han hecho por mí y por hacerme el hombre que soy hoy.

A mis amigos, ya que puedo decir que estoy rodeado de personas increíbles, he aprendido mucho de todos ustedes, les agradezco por apoyarme cuando tuve algún problema, por darme un consejo cuando no tenía la más mínima idea de que hacer, por impulsarme y por creer en mí, estoy seguro de que el camino habría sido muy distinto si no hubiesen estado ahí. Gracias por todo.

A mi asesor el Ingeniero José Luis Barbosa por todo su apoyo, paciencia y sobre todo sus enseñanzas, que no solo se limitaron al ámbito académico, gracias por su dedicación a la enseñanza y por transmitirme ese gusto por la investigación y el desarrollo que me ayudaron a crecer como ingeniero.

Por último, quiero agradecer a la Universidad Nacional Autónoma de México y a la Facultad de Estudios Superiores Cuautitlán por todo lo aprendido en sus aulas y por todas esas experiencias vividas en sus instalaciones que me ayudaron a desarrollarme tanto en el ámbito profesional como en el personal, es un orgullo y un honor ser parte de la UNAM. “Por mi raza hablará el espíritu”.

José Alejandro Maya Camacho

CONTENIDO

Índice de figuras	III
Índice de tablas.....	V
Índice de Pseudocódigos.....	V
Introducción	VI
Justificación	VIII
Objetivo.....	XI
Objetivos particulares	XI
Capítulo 1 Marco teórico.....	1
1.1 Breve historia del lenguaje de descripción de hardware.....	1
1.1.1 Diseño en VHDL	3
1.1.2 Elementos básicos de VHDL.....	4
1.2 Antecedentes.....	5
1.2.1 Memorias programables de solo lectura (PROM).....	6
1.2.2 Dispositivos Lógicos Programables (PLD).....	7
1.2.3 Arreglos Lógicos Programables (PLA).....	7
1.2.4 Lógica de Arreglos Programables (PAL).....	8
1.2.5 Arreglos Lógicos Genéricos (GAL).....	9
1.2.5 Dispositivos Lógicos Programables Complejos (CPLD).....	9
1.2.6 Arreglo de Compuertas Programables en Campo (FPGA)	11
1.3 Procesamiento de imágenes en FPGA	11
1.3.1 Videojuegos en un dispositivo FPGA.....	13
1.4 Diseño con FPGA	15
Capítulo 2 Conceptos para el diseño del sistema de procesamiento de imágenes	19
2.1 Diagrama general de la solución	19
2.2 Imágenes digitales y audio.	21
2.2.1 Las imágenes digitales.....	21
2.2.2 El sonido.....	23
2.2.3 Conversión Analógico-Digital.....	23
2.3 Estándares y protocolos de comunicación para los puertos de entrada y salida.	25
2.3.1 Estándar VGA.....	25
2.3.2 Protocolo I2S.	28
Capítulo 3 Diseño de una aplicación para videojuegos	32

3.1 Planteamiento de la aplicación	32
3.2 Diseño de la aplicación	33
3.2.1 Definición de los módulos del sistema.....	35
3.2.1 Diseño conceptual (Top Level Design).....	36
3.3 Núcleo de la aplicación	39
3.4 Controles	41
3.5 Audio.....	44
3.6 Control de las imágenes	45
Capítulo 4 Implementación en la tarjeta de desarrollo.....	50
4.1 Tarjeta de desarrollo BASYS 3.	50
4.2 Implementación y desarrollo.	52
4.2.1 Módulo Divisor de frecuencia	52
4.2.2 Módulo VGA 640x480.....	55
4.2.3 Despliegue de figuras en pantalla (Módulo Laberinto).....	58
4.2.4 Efecto del movimiento de imágenes (Módulo de los Personajes)	62
4.2.5 Mandos de control.....	65
4.2.6 Despliegue de Imágenes	66
4.2.7 Implementación del audio	70
4.2.8 Interacción con el Puerto VGA (Módulo de control VGA)	72
Capítulo 5 Resultados.....	76
Conclusiones	84
Apéndice	86
Apéndice 1.- Código del sistema - Divisor de frecuencia.	86
Apéndice 2.- Código del sistema - VGA640x480.....	90
Apéndice 3.- Código del sistema - Laberinto.....	92
Apéndice 4.- Código del sistema - ArticaleDisplay.	97
Apéndice 5.- Código del sistema - Acceso a la ROM de Articale.	108
Apéndice 6.- Código del sistema - Control VGA.	109
Apéndice 7.- Código del sistema - Núcleo del sistema.	112
Apéndice 8.- Código del sistema - Controlador del Audio.....	115
Apéndice 9.- Código del sistema - Controlador de los mandos.	119
Bibliografía	123

ÍNDICE DE FIGURAS

Capítulo 1

Figura 1. 1 Proceso de diseño en VHDL.....	4
Figura 1. 2 Entidad en VHDL.....	5
Figura 1. 3 Configuración de una PROM.....	7
Figura 1. 4 Configuración de un PLA.....	8
Figura 1. 5 Configuración de un PAL.....	9
Figura 1. 6 Macrocela de una GAL.[3].....	10
Figura 1. 7 Estructura básica de un CPLD.....	11
Figura 1. 8 Estructura de una FPGA.....	12
Figura 1. 9 Tabla de patrones.....	14

Capítulo 2

Figura 2. 1 Diagrama general del sistema de procesamiento de imágenes.....	20
Figura 2. 2 Ejemplo de baja (Imagen 1) y alta resolución (Imagen 2) en una misma imagen.....	22
Figura 2. 3 Muestreo de una señal de audio.....	24
Figura 2. 4 Flujo de conversión Analógico-Digital.....	24
Figura 2. 5 Diagrama de conexiones del puerto VGA con la FPGA.[18].....	25
Figura 2. 6 Diagrama de señales síncronas en pantalla.....	26
Figura 2. 7 Diagrama general de tiempos para las señales síncronas.....	27
Figura 2. 8 Señales de sincronización horizontal y vertical.....	28
Figura 2. 9 Diagrama básico de conexión por medio del protocolo I2S.....	29
Figura 2. 10 Diagrama de tiempos de transmisión I2S.....	31
Figura 2. 11 Transmisión I2S justificada a la izquierda.....	31

Capítulo 3

Figura 3. 1 Modelo general del sistema completo.....	32
Figura 3. 2 Diagrama de flujo general del proceso de la aplicación.....	34
Figura 3. 3 Diseño de alto nivel (Top level design).....	38
Figura 3. 4 Integración por punto a punto e integración por sistema centralizado.....	40
Figura 3. 5 Máquina de estados para el sistema integrador central.....	41
Figura 3. 6 Mando para puerto de entrada de datos.....	42
Figura 3. 7 Diagrama de tiempos para el envío de datos por los mandos.....	42
Figura 3. 8 Máquina de estados de los mandos de entrada.....	43
Figura 3. 9 Máquina de estados para reproducir sonido.....	44
Figura 3. 10 Visualización en pantalla de los tipos de imágenes.....	45
Figura 3. 11 Jerarquía de imágenes.....	46
Figura 3. 12 Privilegios de imágenes de presentación y de finalización.....	47

Capítulo 4

<i>Figura 4. 1 Tarjeta BASYS 3 indicando los puertos utilizados.[22].</i>	51
<i>Figura 4. 2 Interacción con el módulo de divisor de frecuencia.</i>	54
<i>Figura 4. 3 Despliegue en pantalla del laberinto.</i>	58
<i>Figura 4. 4 Interacción entre el módulo Laberinto.</i>	59
<i>Figura 4. 5 Despliegue del laberinto en pantalla.</i>	60
<i>Figura 4. 6 Máquina de estados del módulo Laberinto.</i>	61
<i>Figura 4. 7 Máquina de estados del personaje principal.</i>	62
<i>Figura 4. 8 Máquina de estados del personaje principal.</i>	63
<i>Figura 4. 9 Máquina de estados del personaje principal.</i>	64
<i>Figura 4. 10 Módulos involucrados con Controles.</i>	65
<i>Figura 4. 11 Puertos disponibles de la tarjeta. [22].</i>	66
<i>Figura 4. 12 Diseño de pines de entrada para los mandos.</i>	66
<i>Figura 4. 13 Bloque de memoria para el resguardo de las imágenes.</i>	67
<i>Figura 4. 14 Generación de los archivos tipo .coe con herramientas de uso libre para almacenamiento de imágenes.</i>	68
<i>Figura 4. 15 Entidad generada para el bloque de memoria.</i>	69
<i>Figura 4. 16 Definición del puerto de salida y conexión con del dispositivo.</i>	71
<i>Figura 4. 17 Generación de los archivos tipo .coe.</i>	71
<i>Figura 4. 18 Pines definidos por la tarjeta y por el dispositivo PMODE AMP3.</i>	72
<i>Figura 4. 19 Controlador de señales hacia el puerto VGA.</i>	73
<i>Figura 4. 20 Conexiones entre el puerto VGA y el FPGA.</i>	75

Capítulo 5

<i>Figura 5. 1 Recursos utilizados en la Tarjeta BASYS 3.</i>	76
<i>Figura 5. 2 Indicadores de las señales enviadas por el núcleo del sistema.</i>	77
<i>Figura 5. 3 Indicadores del estado “fin de juego”.</i>	79
<i>Figura 5. 4 Estado de presentación y Nivel 1.</i>	80
<i>Figura 5. 5 Estado Nivel 2 y Nivel 3.</i>	81
<i>Figura 5. 6 Estado Nivel 4 y Nivel 5.</i>	82
<i>Figura 5. 7 Estado Fin del juego y Juego Ganado.</i>	83

ÍNDICE DE TABLAS

<i>Tabla 1 Comparación entre dispositivos de diseño digital.</i>	<i>VIII</i>
<i>Tabla 2 Jerarquía de las imágenes.</i>	<i>49</i>
<i>Tabla 3 Tabla comparativa de dispositivos con FPGA en el mercado.</i>	<i>50</i>
<i>Tabla 4 Frecuencias de movimiento de los enemigos.</i>	<i>54</i>
<i>Tabla 5 Colores a 12 bits para cada nivel del juego.</i>	<i>61</i>
<i>Tabla 6 Tamaño y gestión de datos para los bloques de memoria.</i>	<i>78</i>

ÍNDICE DE PSEUDOCÓDIGOS

<i>Pseudocódigo 1 - Divisor de frecuencia de 100 MHz a 25 MHz que requiere el módulo VGA640x480.</i>	<i>53</i>
<i>Pseudocódigo 2 - Señal síncrona horizontal.</i>	<i>56</i>
<i>Pseudocódigo 3 - Señal síncrona vertical.</i>	<i>57</i>
<i>Pseudocódigo 4 - Señal de habilitación en pantalla.</i>	<i>57</i>
<i>Pseudocódigo 5 - Algoritmo para obtener la dirección de memoria.</i>	<i>70</i>
<i>Pseudocódigo 6 - Algoritmo para definición de la jerarquía.</i>	<i>74</i>

INTRODUCCIÓN

Durante la década del 2010, se han realizado investigaciones por parte de las empresas dedicadas al diseño de microprocesadores para dispositivos electrónicos, buscando desarrollar e innovar nuevos modelos arquitectónicos para la implementación en nuevas áreas de la electrónica. Una de las empresas que ha estado trabajando en ello es Microsoft©, la cual lanzó una iniciativa para realizar investigaciones, desarrollos e innovaciones combinando las nuevas tecnologías (en especial el uso de dispositivos de silicio como son los Arreglos Genéricos Programables en Campo o también conocidos como FPGA) ciertas áreas de desarrollo como son: la manipulación del proceso paralelo en la utilización de *big data*¹, desarrollo y utilización de la nube como lo es Bing®, aplicaciones de inteligencia artificial, desarrollo en conjunto de aplicaciones con los procesadores Intel® para aumento de velocidad y flexibilidad con reducción de costos y para aplicaciones de tipo *machine learning*².

Varias de estas investigaciones han desarrollado formas de aprendizaje sobre las nuevas tecnologías como son los dispositivos de lógica programable (como los FPGA por mencionar alguna de ellas), donde se realizan desarrollos de sistemas de procesamiento basados en videojuegos[1], aportando un mejor entendimiento de los sistemas basados en lógica programable de descripción de hardware. Dando como resultado una mejora en el entendimiento de los lenguajes basados en descripción de hardware, diseños, delimitaciones y conceptualizaciones que se pudieran llegar durante el desarrollo. Además del apoyo en la búsqueda de nuevas herramientas que pudiesen complementar los proyectos como son dispositivos de entrada y salida (el comprender e implementar protocolos y estándares necesarios para la utilización de éstos).

La utilización de dispositivos de entrada y salida conlleva la búsqueda de información referente a éstos, la visualización en pantalla a través del puerto VGA es una buena opción para la implementación de aplicaciones basadas en dispositivos FPGA. Varios artículos presentan implementaciones y desarrollos sobre diversas tarjetas con dispositivos FPGA para el procesamiento de imágenes desplegadas en pantallas a través del puerto VGA[2], obteniendo buenos resultados en rendimientos y ventajas sobre el espacio en memoria y el tamaño en pantalla de las imágenes desplegadas por medio del estándar VGA con resolución 640x480.

Dado el interés generado por lo anterior, este trabajo buscará desarrollar y evaluar una aplicación de procesamiento de imágenes, a través de un sistema simple basado en un videojuego diseñado e implementado con varios sistemas interactuando al mismo tiempo sobre una misma tarjeta de desarrollo. El diseño propuesto hace uso de varios recursos, por ejemplo:

¹ Big data se refiere a la utilización manipulación de grandes cantidades de datos. Lo caracteriza por el volumen, variabilidad, velocidad, captura y análisis de datos.

² Machine learning es considerada un área de la inteligencia artificial donde los sistemas aprenden los patrones para realización de predicciones de comportamientos futuros.

desplegar imágenes, reproducir audio y administrar registros, por lo que un FPGA podría realizar todas esas tareas de forma independiente y agruparlas en un solo diseño, además de ofrecer flexibilidad para implementar sobre este dispositivo varios de los protocolos de comunicación y estándares en forma de lenguaje descriptivo.

Para implementar dicho sistema es necesario utilizar algunas herramientas para el desarrollo de la aplicación como son: lógica programable (una forma de expresar circuitos lógicos por medio de un lenguaje de programación) ofreciendo un procesamiento paralelo aplicado en un FPGA entre los módulos internos del sistema, bloques de memoria (celdas independientes que tienen el propósito de guardar datos) que serán utilizados para almacenar las imágenes de la aplicación y por último los puertos de entrada y salidas (túneles que interactúan con el exterior del sistema) con el fin de ofrecerle al usuario una interacción que pueda comprender durante la ejecución de la aplicación.

Dichos dispositivos se tornan ideales y flexibles en el proceso de diseño porque permiten ser desarrollados por medio de un lenguaje de descripción de hardware, de esta manera se pueden dividir las tareas entre cada uno de los sistemas involucrados en el procesamiento de imágenes.

Este trabajo consta de 5 capítulos, el primer capítulo explica el marco teórico para poner en contexto el tema de la lógica programable, abordando de forma general su funcionamiento, componentes y características que lo hacen destacar para cumplir el objeto de estudio en el procesamiento de imágenes y sus aplicaciones en los videojuegos; en el segundo capítulo se explicarán los conceptos necesarios para el diseño de la aplicación de procesamiento de imágenes, tanto los sistemas involucrados, la forma de interpretar los datos de entrada, la manipulación de los datos y el procesamiento de la salida de datos requeridos para el diseño, abarcando los protocolos de comunicación y algunos estándares; el tercer capítulo se describirá el diseño de la aplicación del sistema de procesamiento de imágenes aplicado en un sistema de videojuegos; en el cuarto capítulo se mostrará la implementación de los sistemas descritos sobre la tarjeta de desarrollo; y en el quinto capítulo se mostrarán los resultados obtenidos de la implementación del sistema.

JUSTIFICACIÓN

El desarrollo de este trabajo es derivado por el interés de conocer y evaluar las características de los dispositivos FPGA que están evolucionando junto con las nuevas tecnologías y cómo es el impacto que tienen en la manipulación de estos en sistemas aplicados, entre ellos el procesamiento de imágenes.

A nivel académico se busca comprobar las ventajas que tiene el trabajar con estos dispositivos en el área de la electrónica y su viabilidad en el manejo de imágenes con dispositivos con FPGA evaluando los recursos que disponen.

A nivel institución se muestra este trabajo basado en estudios con FPGA para destacar la importancia de estos dispositivos, debido a que cambiarán la forma de diseñar nuevas aplicaciones para las diferentes áreas de la ingeniería como son: bioingeniería, ingeniería aeroespacial, telecomunicaciones, etc.

El hecho de usar un dispositivo FPGA tiene ventajas frente a otro tipo de sistemas, por ejemplo, el procesamiento de algoritmos de forma concurrente ha acelerado los tiempos de ejecución respecto a una programación secuencial, también resulta ser más económicos respecto a otros dispositivos como se muestra en la tabla 1.

	<i>Rendimiento</i>	<i>Costo</i>	<i>Flexibilidad</i>
<i>ASIC</i>	Alto	\$482.41 USD [3]	Baja
<i>Procesador</i>	Bajo	\$20.00 USD [4]	Alta
<i>FPGA</i>	Medio	\$149.00 USD [5]	Alta

Tabla 1 Comparación entre dispositivos de diseño digital.

De la tabla 1 se puede observar que los FPGA son una alternativa muy atractiva para el desarrollo de proyectos al no ser tan costosas como un ASIC (circuitos integrados diseñados a la medida) y ser igual de flexibles como un microprocesador, pero con un mayor rendimiento.

Una de las características que puede ofrecer los dispositivos FPGA con respecto a un CPU es la baja latencia³. Con un FPGA es factible obtener una latencia aproximadamente de 1

³ Latencia es la velocidad de respuesta que se tiene cuando el emisor transmite un dato y el remitente lo responde.

microsegundo a comparación de un CPU con 50 microsegundos[6], con estos tiempos se puede hacer la diferencia en la respuesta para un sistema programado del piloto automático de un avión de combate. Además, la latencia de un FPGA es mucho más determinista porque no depende de un sistema operativo para poder realizar algún envío de datos entre otros dispositivos, como lo podría ser la comunicación por medio de los buses de entrada y salida (el puerto USB por mencionar alguno).

La conectividad que ofrece los FPGA es mucho más diversa. Esto se debe a que puede conectarse con cualquier fuente de datos directamente a través de los pines del dispositivo. Contrastando contra los CPU que deben realizar sus conexiones a través de los buses estandarizados y depender directamente de la respuesta del sistema operativo (ofrece un mayor ancho de banda para el envío de datos). Un ejemplo de este caso es el sistema LOFAR⁴ que genera una gran cantidad de datos y que deben ser reducidos para poder ser enviados de tal forma que sean manejables. Por lo que el instituto ASTRON⁵ diseñó un tablero con 4 dispositivos FPGA que manipulan más datos por segundo que el intercambio de internet. [6]

Otra característica de los FPGA es la eficiencia energética, eso se debe a que las tarjetas con FPGA tienen sus propios puertos de entrada y salida y no requieren una computadora de tipo Host para funcionar, ahorrando así energía y dinero.

Se debe observar cuales son las medidas que las empresas toman para ir actualizando y mejorando los nuevos sistemas como lo son: el desarrollo de información en la nube (o Cloud); implementación de nuevas tecnologías de IoT (Internet de las cosas o *Internet of Things*); la mejora de procesadores (como lo es el dispositivo *Intel Stratix 10 DX*[7] por mencionar alguna), donde ofrecen un procesador con un dispositivo FPGA que aumenta la aceleración de las cargas de trabajo.

Un ejemplo de lo anterior, es la empresa INTEL® con la compra de ALTERA®[8], donde tiene una visión en el futuro utilizando dispositivos FPGA para desarrollar nuevos productos con mejores características y rendimientos. Otra de ellas fue Microsoft©, lanzando

⁴ LOFAR (por sus siglas en inglés Low Frequency Array) es una red distribuida de sensores multipropósito utilizado principalmente como radiotelescopio para las áreas de la astronomía, geofísica y agronomía.

⁵ ASTRON es un Instituto para la radio astronomía en Países Bajos (Netherlands).

el proyecto CATAPULT[9] donde se desarrolla nuevas tecnologías en base a dispositivos FPGA para mejorar áreas mencionadas anteriormente.

OBJETIVO

Diseñar e implementar un sistema de procesamiento de imágenes con un sistema embebido en FPGA, utilizando el lenguaje de descripción de hardware VHDL.

OBJETIVOS PARTICULARES

- Desarrollar una plataforma de procesamiento de imágenes haciendo uso eficiente de los recursos de hardware de una FPGA.
- Implementar sobre la tarjeta FPGA los protocolos de comunicación necesarios para el sistema.
- Implementar el diseño del sistema de procesamiento de imágenes usando lenguaje VHDL por medio de una tarjeta FPGA BASYS 3.

CAPÍTULO 1 MARCO TEÓRICO

En este capítulo se fundamentarán los conceptos teóricos necesarios para la comprensión y desarrollo del presente trabajo de tesis. Se dará una idea general de los conceptos principales utilizados y sus características más importantes. Primero se va a definir el lenguaje de descripción de hardware con el que se realizará el código del proyecto, más adelante se abordarán las características de un FPGA y las ventajas de utilizarlas para el diseño de sistemas digitales.

1.1 BREVE HISTORIA DEL LENGUAJE DE DESCRIPCIÓN DE HARDWARE

El desarrollo de circuitos integrados digitales programables y la necesidad de desarrollar aplicaciones cada vez más complejas hizo que las herramientas tradicionales se volvieran ineficientes. Como consecuencia, las empresas desarrollaron los lenguajes de descripción de hardware (HDL por sus siglas en inglés *Hardware Description Language*).

Hay diferentes formas de describir en HDL:

- a) Funcional: en esta forma se pueden representar los circuitos combinatoriales como pueden ser sumadores, multiplicadores, codificadores, decodificadores o conversores de código utilizando funciones elementales como AND, OR, NOT, XOR, XNOR.
- b) Procedimental: los valores asignados se describen mediante un procedimiento, el cual se ejecuta ante el cambio de señales, mismos que pueden representar circuitos combinatoriales o secuenciales utilizando sentencias o instrucciones como *if-else*, *case*, *for* o *while*.
- c) Estructural: en este modelo se realiza el diseño de todos los módulos y conexiones que se utilizarán obteniendo al final un diagrama que contiene los componentes, procesos y

lazos de los que se compone el sistema conocido como diagrama de alto-nivel (Top-Level).

Algunos tipos de HDL que existen son:

- ABEL-HDL que es un tipo de lenguaje de descripción que soporta una variedad de entradas incluyendo ecuaciones de alto nivel, diagramas de estados y tablas de verdad.
- SystemC es un conjunto de librerías y macros implementadas para ser utilizadas como extensión de C++, esta puede agregar modelos de constructores⁶ como: la concurrencia, relojes, módulos e interconexiones en base al núcleo de C++.
- Verilog es uno de los lenguajes de descripción de hardware más usados, permite usar diferentes niveles de descripción de sistemas digitales en un mismo ambiente, admite la descripción estructural del diseño en base a los componentes básicos como transistores, compuertas lógicas, multiplexores, etc., los cuales se enfocan principalmente en la conducta del sistema.

Con el fin de estandarizar los sistemas desarrollados, el IEEE⁷ y el Departamento de Defensa de los Estados Unidos iniciaron el desarrollo de un lenguaje llamado VHDL, su nombre viene de VHSIC que significa *Very High Speed Integrated Circuit* y HDL de las siglas *Hardware Description Language*, es decir Lenguaje de Descripción de Hardware para Circuitos Integrados de Velocidad Muy Alta.

El lenguaje VHDL fue concebido para cubrir varias necesidades que surgen durante el proceso de diseño. Permite realizar una descripción funcional o de comportamiento del circuito, utilizando técnicas procedimentales. Asimismo, permite describir la estructura del diseño y declarar las entidades y subentidades que lo forman especificando una jerarquía entre las mismas y sus interconexiones. Por último, permite simular el diseño con herramientas de síntesis especiales, para poder abstraer la descripción de un circuito, hasta obtener un diseño solamente estructural.

⁶ Un constructor son variables, métodos y funciones que pueden ser utilizados cuando se inicializa o se crea el objeto.

⁷ El IEEE (por sus siglas en inglés *The Institute of Electrical and Electronic Engineers*). Es una organización dedicada a la normalización de procesos en el área de ingeniería a nivel mundial.

Las características más importantes de VHDL son las siguientes:

- Sintaxis parecida al Lenguaje ADA⁸.
- Interfaz única que permite ser conectada a otros elementos.
- Comportamiento preciso.
- Estructura jerárquica.
- Se puede simular cualquier operación lógica y de temporización.

1.1.1 DISEÑO EN VHDL

VHDL permite describir de forma estructurada un circuito, es decir, diseñarlo a partir de pequeños circuitos más sencillos, además, para desarrollar estos circuitos se debe seguir un proceso de diseño que va a permitir elaborar de forma estructurada cada módulo planteando varias etapas en el proceso, donde, de producirse un error, debe ser corregirlo antes de proceder al siguiente paso. Las etapas de diseño son las siguientes:

- Análisis: recopilación de todos los requerimientos funcionales obtenidos durante una etapa de descubrimiento.
- Delimitación: parte del proceso donde se definen los procesos de investigación, tiempos de desarrollo y recursos disponibles.
- Conceptualización: plasmar una idea generalizada con base al análisis realizado y delimitaciones definidas previamente.
- Definición del problema: esta etapa es considerada crucial, porque puede decidir el tipo de preguntas y la calidad de resultados que se desean obtener durante todo el proceso. Además de plantear una solución para generar una hipótesis que definirá el rumbo que tomará el diseño.
- Diseño general de la solución (Top Level): en esta etapa se trata al circuito como una caja negra, donde solo se observan las entradas y salidas de los diferentes módulos que conforman el sistema y cómo se comunican entre sí.

⁸ ADA es un lenguaje de programación orientado a objetos utilizado principalmente para aplicaciones donde se requiera mucha seguridad.

- **Codificación:** una vez definidas las interfaces del proyecto se tienen que describir en VHDL, preferentemente en un editor especializado, el cual verifique que toda la sintaxis sea la correcta.
- **Simulación:** una vez que el código es correcto se definen entradas de prueba y se verifica que las salidas correspondan a los valores esperados.
- **Síntesis:** convierte el modelo descrito en el conjunto de conexiones de circuitos necesarios para la implementación sobre el FPGA y la tarjeta de desarrollo que se utilizará.
- **Implementación:** mapeo de las conexiones generadas en la síntesis de acuerdo con los recursos disponibles de la tarjeta de desarrollo, generando un archivo tipo bit⁹.
- **Programación de la tarjeta de desarrollo:** si se cumplieron exitosamente todas las etapas anteriores, incluyendo la generación del archivo bit, se debe descargar dicho archivo hacia el FPGA para observar el comportamiento real del diseño.

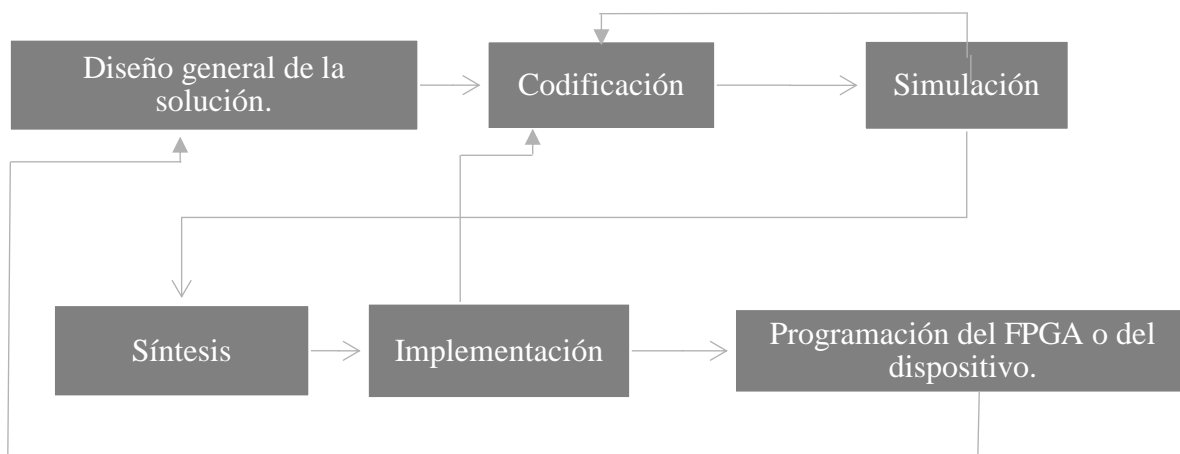


Figura 1. 1 Proceso de diseño en VHDL.

1.1.2 ELEMENTOS BÁSICOS DE VHDL

Cualquier diseño que se realice en VHDL debe ser descrito por sus señales y como interactúan entre ellas, indicando cuántos puertos de entrada y salida se asumen, a esto se le conoce como la entidad, la cual se muestra en la figura 1.2. Una entidad puede definirse como

⁹ Archivo generado por el editor especializado para programar el dispositivo FPGA el cual contiene todo el mapeo de las interconexiones de ruteo que determinará el comportamiento del circuito dentro del FPGA.

una representación de un sistema electrónico en la cual se describen únicamente los elementos externos que deben tener un nombre, si son entrada o salida y el tipo de datos que van a transmitir o recibir.



Figura 1. 2 Entidad en VHDL.

Una vez descrita la entidad, se debe especificar cómo deberán funcionar los puertos de salida con respecto a las entradas, a este elemento se le conoce como arquitectura, donde, por medio de funciones propias de VHDL, se va a describir el comportamiento de la entidad, por lo que una arquitectura siempre irá asociada a una entidad. Las principales características de una arquitectura son las siguientes:

- Describir el comportamiento del circuito.
- Declarar las señales internas, los procedimientos variables y constantes necesarios en el diseño.
- Describir de forma estructural o por comportamiento.

1.2 ANTECEDENTES

Numerosos circuitos integrados (CI) utilizan circuitos lógicos de uso común, por ejemplo, compuertas lógicas, codificadores, convertidores o sumadores, estos componentes son de los años 70, por esta razón, actualmente son utilizados únicamente con fines didácticos. Dependiendo de la cantidad de transistores, compuertas o funciones de las que se compone el CI, es decir, de su escala de integración; se clasifican en pequeña (SSI por sus siglas en inglés Small Scale Integration) las cuales se consideran las más pequeñas con solo algunos transistores. Mediana (MSI por sus siglas en inglés Medium Scale Integration) estos integrados fueron utilizados en 1970 por los primeros ordenadores. Gran (LSI por sus siglas en inglés Large Scale Integration) este tipo de escala es utilizado en los primeros microprocesadores. Muy grande

(VLSI por sus siglas en inglés Very Large Scale Integration) utilizados para procesadores más potentes en marcas reconocidas. Ultra (ULSI por sus siglas en inglés Ultra Large Scale Integration) utilizado en microprocesadores complejos a principios de los 80s para el desarrollo de la serie 8086 de Intel®.

Un gran problema fue que, si el diseñador requería diseños de circuitos lógicos más complejos, se necesitaban más circuitos integrados, dando como resultado alterar el diseño original y modificarlo en un circuito más complejo y en un PCB¹⁰ más grande, dejando de optimizar espacio. Por esta razón, los nuevos circuitos tienen una mayor escala de integración, es decir, contienen más componentes por circuito integrado. Además, el hecho de modificar el diseño en el software implica alteraciones en el hardware, cableado, distribuciones de los circuitos, etc. Por estos hechos, la lógica programable atenúa estos cambios porque solamente se realizan modificaciones dentro del código.

1.2.1 MEMORIAS PROGRAMABLES DE SOLO LECTURA (PROM).

Los circuitos programables, también conocidos como dispositivos programables, pueden ser considerados como un circuito de propósito general, donde el usuario puede modificar la estructura interna. Las características antes mencionadas describen perfectamente a un dispositivo de memoria PROM (*Programmable Read-Only Memory*).

La memoria PROM utilizan bits de direccionamiento donde cada una de estas direcciones tiene un tamaño fijo de memoria que depende del dispositivo utilizado. Las memorias PROM sólo pueden ser programadas una vez.

En la figura 1.3 se puede observar la estructura interna de una PROM como dispositivo lógico programable. La PROM se compone de una matriz AND fija y una OR programable.

¹⁰ Un PCB por sus siglas en inglés *Printed Circuit Board* es la placa donde se conectan los dispositivos electrónicos de forma permanente.

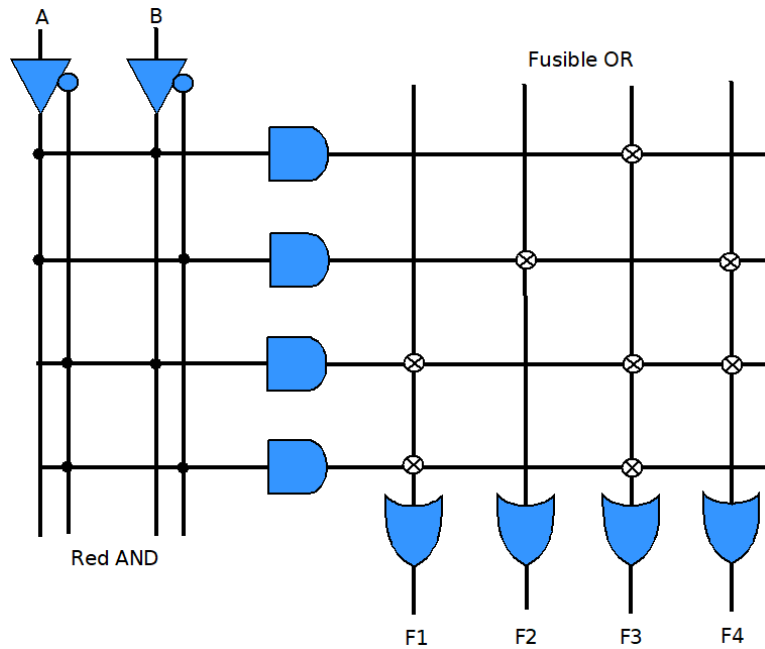


Figura 1. 3 Configuración de una PROM.

1.2.2 DISPOSITIVOS LÓGICOS PROGRAMABLES (PLD)

El **PLD** (Programmable Logic Device) es un dispositivo que consiste en una matriz de compuertas AND conectada con otra matriz de compuertas OR. Este diseño permite al circuito ser modelado como una suma de productos. Una de las ventajas es que este circuito puede ser muy eficiente, solamente si se implementa en diseños que no superan a unos cientos de compuertas. Un problema que se puede ver es que la arquitectura es muy rígida y está muy limitada a un número de entradas y salidas.

1.2.3 ARREGLOS LÓGICOS PROGRAMABLES (PLA)

Los **PLA** (por sus siglas en inglés Programmable Logic Array) consisten en dos matrices que son programables, una AND y otra OR. Tienen la característica de que en las entradas y las salidas hay compuertas NOT para obtener mayor versatilidad en la salida (ver figura 1.4). Los PLA pueden ser conectadas externamente a Flip-Flop's para poder diseñar circuitos secuenciales.

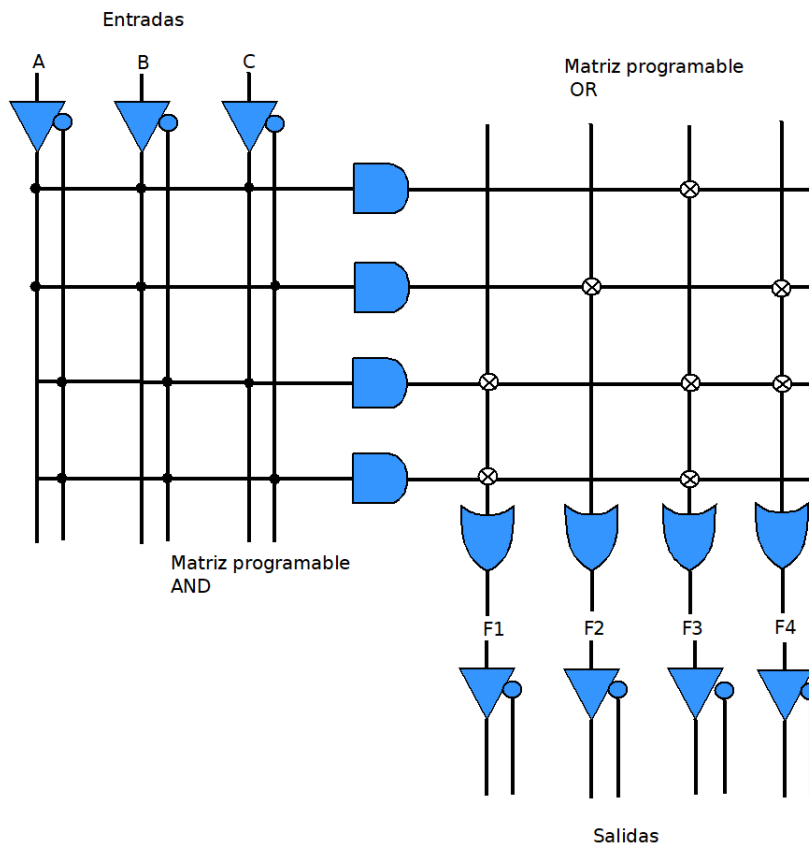


Figura 1. 4 Configuración de un PLA.

1.2.4 LÓGICA DE ARREGLOS PROGRAMABLES (PAL)

Los PAL (por sus siglas en inglés Programmable Array Logic) son considerados una variante de las PLA que consisten en dos planos:

1. Un plano AND programable.
2. Un plano OR fijo.

En la figura 1.5, se incluyen inversores en la entrada con las que se pueden implementar funciones con variables y su complemento (como en el dispositivo lógico anterior PLA). En las salidas se añade un arreglo fijo OR, donde se limita mucha de la funcionalidad y flexibilidad que ofrecía el circuito anterior (PLA).

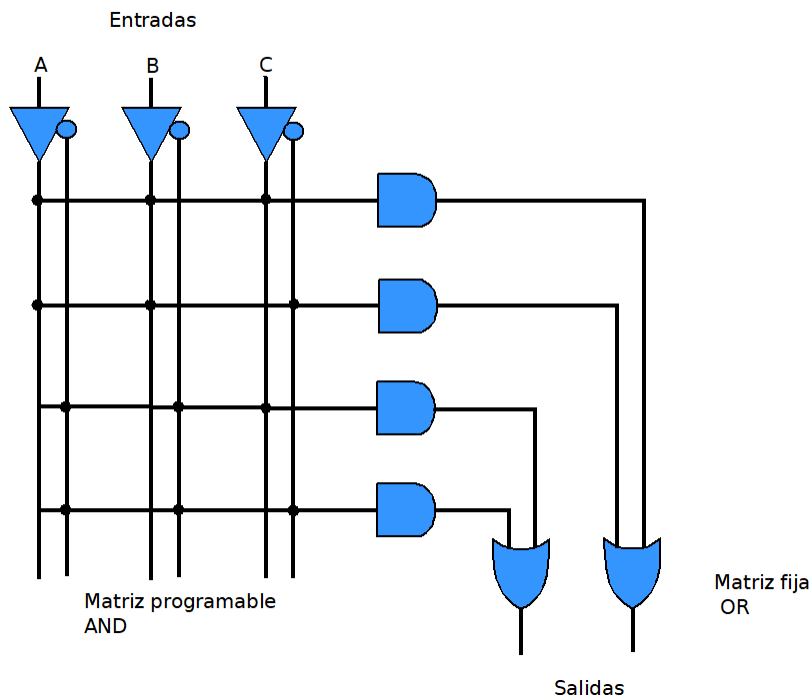


Figura 1. 5 Configuración de un PAL.

1.2.5 ARREGLOS LÓGICOS GENÉRICOS (GAL)

La **GAL**, por sus siglas en inglés *Generic Array Logic*, es básicamente un PLA (como se muestra en la figura anterior 1.4) pero contiene una Macrocelda que es un conjunto de Flip-Flop's, todo esto para cambiar el estado lógico de la salida además de realimentar las salidas de los Flip-Flop's (se puede observar en la figura 1.6). Utiliza una matriz de memoria EEPROM, dando la oportunidad a la GAL de ser programada varias veces.

1.2.5 DISPOSITIVOS LÓGICOS PROGRAMABLES COMPLEJOS (CPLD)

Un **CPLD** (por sus siglas en inglés *Complex Programmable Logic Device*) es un PLD a un mayor nivel de integración y tiene las siguientes características.

- Permiten implementar sistemas más eficientes.
- Utilizan menos espacio.
- Mejoran la confiabilidad en el circuito.
- Reducen costos.

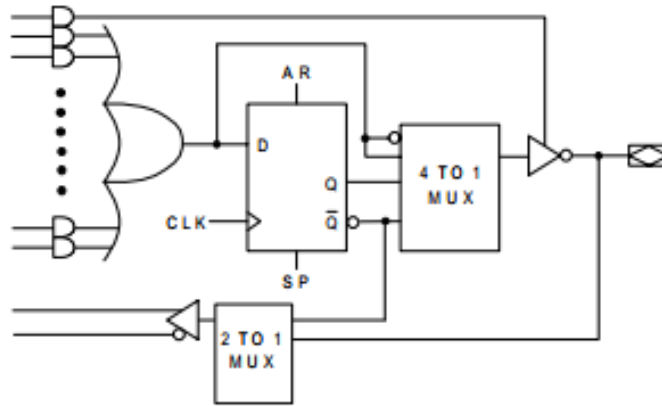


Figura 1. 6 Macrocelda de una GAL.[3]

El CPLD se forma con múltiples bloques lógicos, donde cada uno es similar a un PLD. Los bloques lógicos se comunican entre sí utilizando una matriz programable de interconexiones lo cual hace más eficiente el uso del silicio y tiene un mejor desempeño. La arquitectura de un CPLD se compone de tres bloques principales:

- 1) Bloque Lógico (Logic Block): un Bloque Lógico es muy parecido a un PLD donde cada uno de ellos poseen generalmente una matriz de compuertas AND, una matriz de compuertas OR y una configuración para la distribución de los productos en las diferentes macroceldas del bloque. La cantidad de bloques lógicos que puede tener un CPLD dependerá de la familia y del fabricante del dispositivo.
- 2) Matriz de Interconexión Programable (Programmable Interconnect Matrix): permite unir los pines de I/O con las entradas del bloque lógico o sus salidas a las entradas del bloque lógico.
- 3) Bloque de Entrada/Salida (Input Output Block): La función del Bloque de I/O es permitir el paso de la señal hacia adentro o al exterior del dispositivo.

Se pueden observar los bloques correspondientes del diagrama del circuito CPLD en la figura 1.7.

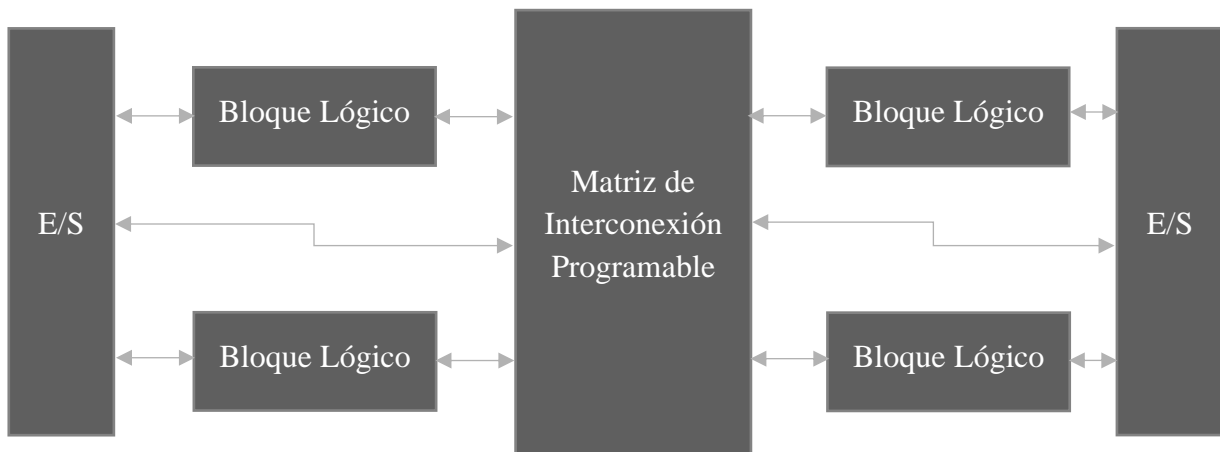


Figura 1. 7 Estructura básica de un CPLD.

1.2.6 ARREGLO DE COMPUERTAS PROGRAMABLES EN CAMPO (FPGA)

El Arreglo de Compuertas Programables en Campo o Field Programmable Gate Array, son circuitos de alta densidad programables por el usuario con un tiempo de respuesta mucho más rápido en comparación de otros sistemas. *Los FPGA presentan líneas de interconexión, las cuales son agrupadas en canales verticales y horizontales, donde disponen de células de memoria de configuración que son distribuidas a lo largo de todo el chip*[10] (como se puede observar en la figura 1.8), éstas almacenan toda la información necesaria para programar los dispositivos mencionados anteriormente.

1.3 PROCESAMIENTO DE IMÁGENES EN FPGA

El procesamiento digital de imágenes ha tomado mucha relevancia en diversas áreas de investigación, por ejemplo, aplicaciones industriales, ciencias médicas, satélites o videojuegos, estas aplicaciones usualmente requieren el suavizado de la imagen, la eliminación del ruido o realzar y detectar bordes, por mencionar algunos. Generalmente los algoritmos empleados requieren de mucha velocidad de procesamiento, flexibilidad y tiempo de desarrollo para ser

completados exitosamente. Existen herramientas de software que pueden solventar estas necesidades haciendo uso de una computadora, esto hace a la implementación del algoritmo mucho más sencilla, pero se pierde velocidad de ejecución.[2]

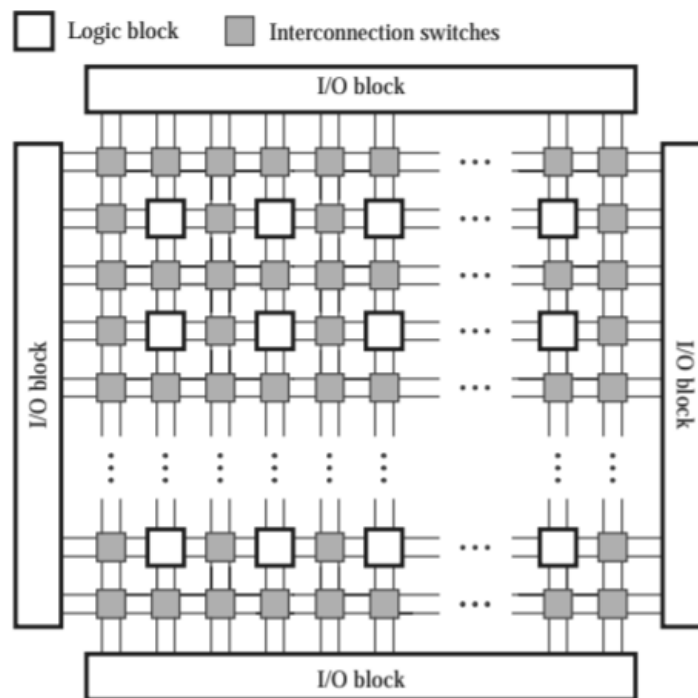


Figura 1. 8 Estructura de una FPGA¹¹.

La solución que se le ha dado a este tipo de problemas es el uso de arquitecturas de procesadores en paralelo, sin embargo, están limitadas a pocas aplicaciones por sus altos costos de diseño ya que hay sistemas que pueden alcanzar un valor de más de \$4000 dólares.[11] Los FPGA logran hacer frente a las exigencias de estos procesos ofreciendo grandes ventajas en cuanto a velocidad de procesamiento y costos del proyecto, ya que son capaces de realizar operaciones completas o reprocesar los datos antes de utilizarlos sin perder velocidad en el proceso[12].

¹¹ Imagen tomada de: S. D. Brown y Z. G. Vranesic, Fundamentals of digital logic with VHDL design, 3rd ed. New York, NY. McGraw-Hill, 2009.

1.3.1 VIDEOJUEGOS EN UN DISPOSITIVO FPGA

Los videojuegos han formado parte de la vida diaria de la sociedad desde hace mucho tiempo, se puede considerar al OXO¹² como el primer videojuego de la historia en 1952. En los años ochenta la industria de los videojuegos tuvo un crecimiento exponencial debido a las máquinas recreativas y a las primeras consolas domésticas. [13]

En las consolas antiguas, los recursos de hardware no eran muchos, se requería de un gran procesamiento y mucha memoria, esto hizo que los desarrolladores emplearan métodos muy ingeniosos para la creación de sus juegos. Los cartuchos de esa época guardaban los gráficos en una memoria separada del código del juego, por otra parte, la tarjeta madre de la consola tenía el CPU donde se ejecutaba toda la lógica del juego y la parte gráfica era procesada por la unidad de procesamiento de imágenes conocida como PPU (por sus siglas en inglés *Picture Processing Unit*). Este dispositivo tenía acceso directo a la memoria donde estaban almacenadas las imágenes y las desplegaba en la pantalla generando cada cuadro a la vez.

Una sola imagen en pantalla requería de 61440 píxeles[14], que para un cartucho de esa época era imposible de almacenar en ese espacio. Para solventar eso, se creó el concepto de “tiles” el cual consiste en almacenar pequeños cuadros con fragmentos de una imagen y reutilizarlos una y otra vez a lo largo del juego y guardarlos en una sola imagen conocida como la tabla de patrones similar a la mostrada en la figura 1.9.

Una plataforma de videojuegos, en su forma más simple, solo requiere una interfaz que interactúe con el jugador, otra que lo haga con las salidas y un mecanismo que permita implementar la lógica del juego[15]. Como se puede ver en la figura 1.10, la estructura simple de un videojuego consta de esas partes esenciales. Pueden agregarse muchas más, pero eso dependerá de los requerimientos y alcances que permita el juego.

¹² Videojuego desarrollado por Alexander Douglas, era una versión computarizada del juego del gato que permitía enfrentar a un jugador contra la máquina.

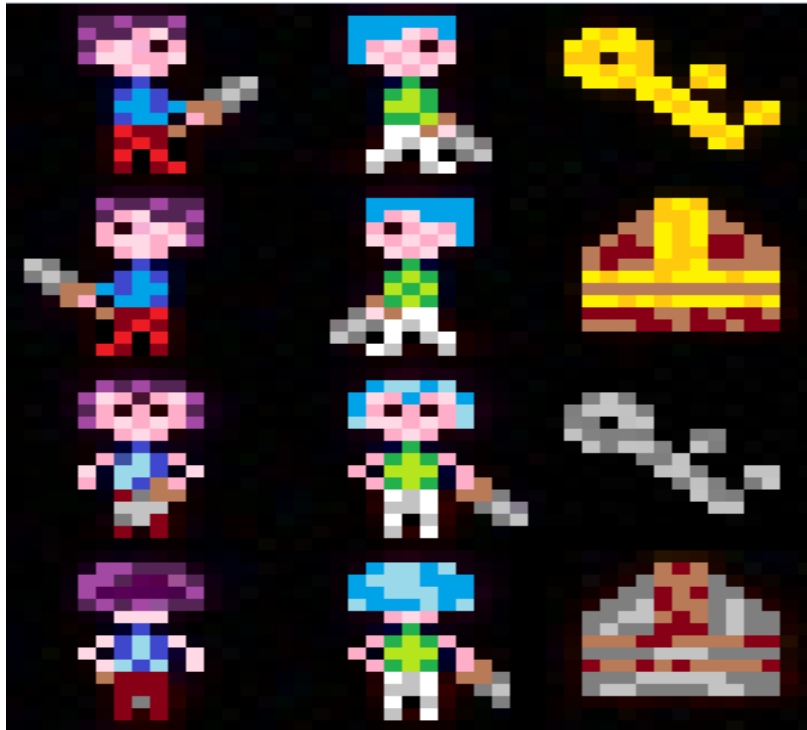


Figura 1.9 Tabla de patrones.

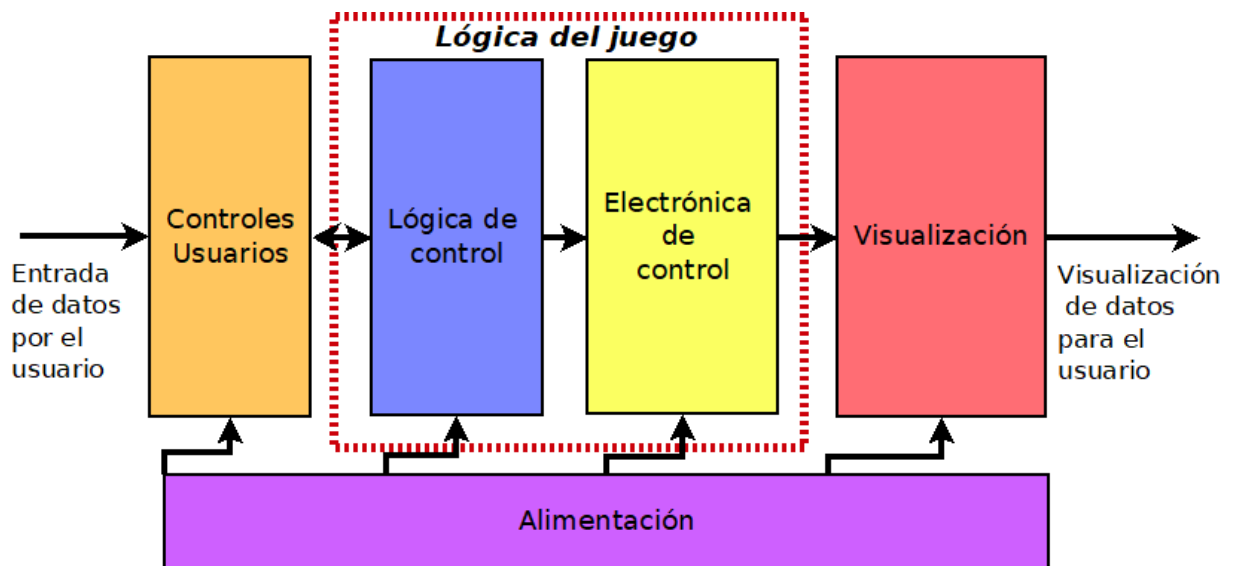


Figura 1.10 Diagrama de una estructura simple de un videojuego. [16]

1.4 DISEÑO CON FPGA

De forma física, el proceso de un circuito digital utilizando una matriz lógica programable puede descomponerse en dos etapas básicas:

1. Dividir el circuito en bloques básicos, asignándolos a los bloques configurables del dispositivo.
2. Conectar los bloques de lógica mediante los conmutadores necesarios.

A continuación, se mencionan los bloques principales de los que se componen los FPGA:

1. **Bloques lógicos**, cuya estructura y contenido se denomina arquitectura. Suelen incluir biestables para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de Entrada/Salida.
2. **Recursos de interconexión**, cuya estructura y contenido se denomina arquitectura de ruteo.
3. **Bloques de memoria**, son los que se carga, durante el RESET para configurar bloques y conectarlos.

Existen muchas ventajas que puede proporcionar el uso de los FPGA como:

- **Hardware a la medida.**

Cuando se hace referencia al termino de “hardware a la medida” es cuando el diseñador no tiene que buscar los productos en el mercado que mejor se adapten a sus diseños, sino que se los diseña a la medida de sus necesidades, o reutiliza o modifica los diseños ya existentes. En un sistema que incorpore un FPGA, es el diseñador que implementa solamente los controladores necesarios para nuevas posibilidades de diseño, por ejemplo, el crear una CPU específica para una aplicación determinada junto con sus propios controladores de periféricos.

- **Reducción del ciclo de diseño.**

El modelo de diseño de hardware basado en HDL contiene muchas de las ventajas del diseño software. El circuito es ahora un archivo de texto, que se puede editar, simular, modificar y finalmente sintetizar. Se pueden crear repositorios de hardware, con colecciones de diseños

ya probados: controladores de puerto VGA, UART, temporizadores, CPU, etc., donde el diseñador puede crear prototipos de manera muy rápida, probarlos, medirlos y modificarlos.

- **Flexibilidad.**

Con el mismo hardware, se puede conseguir un sistema con comportamientos diferentes. En la misma tarjeta, se puede tener sintetizada una CPU además de poder probar un puerto con un protocolo UART.

- **Posibilidad de transferir algoritmos al hardware.**

En los diseños mixtos de microcontrolador con FPGA, se puede decidir el implementar una solución de hardware frente a una puramente de software, consiguiéndose una mayor velocidad. Por ejemplo, la implementación de algoritmos de cifrado. En un microcontrolador de 8 bits esto sería un proceso lento, pero se vuelve viable si se añade un hardware que haga el cifrado.

- **Diseños de hardware libre.**

Posibilidad de realizar diseños que se compartan dentro de una comunidad y que cualquier diseñador pueda utilizarlos, modificarlos y distribuir las modificaciones. Esto es especialmente útil en el campo de la docencia y la investigación, donde se potencia si el hardware en el que se prueban los diseños también es libre.

- **Paralelismo**

Una de las ventajas más notorias de usar un FPGA es la programación en paralelo. Ésta ofrece el beneficio de realizar 2 o más procesos secuenciales de forma concurrente, dando la oportunidad de implementar diversos sistemas independientes funcionando en línea o módulos que dependen de la señal procesada de otro sector.

Las aplicaciones de los FPGA pueden brindar más de una solución y actúan en muchos campos, esto se debe al amplio manejo de datos y señales que brindan estos dispositivos. Dentro de las áreas de aplicación se encuentran[10]:

Aerodefensa espacial: son utilizados en radares y sistemas de guía, además pueden ser implementados en diferentes áreas de estudio como: industrial o comercial a través de las tecnologías de la FPGA.

Prototipos para ASIC: este tipo de aplicación es una de las más utilizadas, ya que en una FPGA se pueden crear rápidamente muchos sistemas que cumplan las funciones propias de un ASIC con la ventaja de poder ser probados de forma inmediata.

Audio y video: en el manejo de una gran gama de señales que permiten que los FPGA brinden varias soluciones de costo reducido y con gran confiabilidad.

Automotriz: los sistemas de navegación y asistencia al conductor, monitoreo del correcto funcionamiento de un vehículo, sistemas de información y entretenimiento pueden ser creados con estos dispositivos.

Informática de alto rendimiento: con la integración de microprocesadores dentro del chip FPGA se puede lograr una aceleración de hardware, lo que conduce a un alto rendimiento en el procesamiento de datos y señales.

Centro de datos: diseñado para grandes anchos de banda, donde los FPGA se pueden implementar como servidores de baja latencia, sistemas de redes informáticas y aplicaciones de control de datos.

Seguridad: ya se habló sobre el procesamiento de señales de audio y video, así como la posibilidad de tener un microprocesador dedicado, si a esto se agregan sistemas biométricos como sensores de huellas digitales o de voz y de control de datos, se tienen los elementos fundamentales para la implementación de sistemas de seguridad y vigilancia.

Área médica: la gran capacidad de procesamiento, visualización y muestreo de datos permite que se puedan crear aplicaciones para equipos de monitoreo, diagnóstico y aplicación de terapias.

Aplicaciones inteligentes: en la búsqueda de la comodidad, los sistemas inteligentes han tomado un gran auge, un FPGA puede adaptarse a las exigencias que demandan estos sistemas.

Equipos medidores y de pruebas: se pueden diseñar toda una gran gama de equipos, como ejemplo se tienen: osciloscopios, generadores de señal y analizadores, analizadores lógicos, multímetros, equipos de medición para industria automotriz, probadores analógicos de radiofrecuencia, probadores de señales mixtas, probadores de memoria, etc.[10]

Como se puede deducir, el campo de aplicación de estos dispositivos programables es muy amplio. El trabajo se centra en desarrollar un sistema de procesamiento de imágenes complementado con una aplicación para mostrar el proceso de diseño e implementación. Para lo cual, en los siguientes capítulos se abordan las bases de la aplicación y los detalles de dicho proceso.

CAPÍTULO 2 CONCEPTOS PARA EL DISEÑO DEL SISTEMA DE PROCESAMIENTO DE IMÁGENES

En este capítulo se presentará la propuesta de solución para el sistema de procesamiento de imágenes, se definirán las interfaces a utilizar y los protocolos necesarios para establecer una comunicación eficiente entre los dispositivos de entrada y salida y la tarjeta de desarrollo. Posteriormente, se explicará los conceptos teóricos relacionados a cada una de las interfaces, como son el estándar VGA¹³, el protocolo I2S¹⁴ y el proceso para generar y guardar datos en la memoria de la FPGA y poder acceder a ellos cuando sea necesario.

2.1 DIAGRAMA GENERAL DE LA SOLUCIÓN

En la figura 2.1, se muestran las interfaces involucradas en la solución propuesta para el sistema de procesamiento de imágenes, como se observa, el sistema es controlado totalmente por el FPGA de la tarjeta de desarrollo. Primeramente, se puede ver en la parte superior de la figura 2.1, que el diseño cuenta con un par de controles para permitir a dos usuarios el manipular las imágenes por medio de la transmisión de datos. Una vez que se obtienen dichos datos de los controles serán enviados a un módulo que determinará el movimiento de la imagen, el cual va a estar conectado a un bloque de memoria para obtener la información de la imagen y enviarla al controlador VGA. De igual forma, existe un bloque que va a interactuar algunas imágenes de forma independiente al usuario en la pantalla accediendo a su propio bloque de memoria y, de la misma forma, será enviado al módulo VGA para ser desplegado en pantalla con las otras imágenes.

Como más de un bloque será mostrado en pantalla, el sistema cuenta con un controlador VGA el cual seleccionará que imagen será desplegada en un determinado instante de tiempo, dependiendo del píxel que se esté mostrando en ese momento.

¹³ VGA (como en sus siglas en inglés Video Graphics Array) la matriz de gráficos de video es un estándar que se implementó para la transmisión de datos de imágenes.

¹⁴ I2S (como en sus siglas en inglés Integrate Interchip Sound) es una interfaz de tipo serial para interconectar dispositivos digitales de audio entre sí.

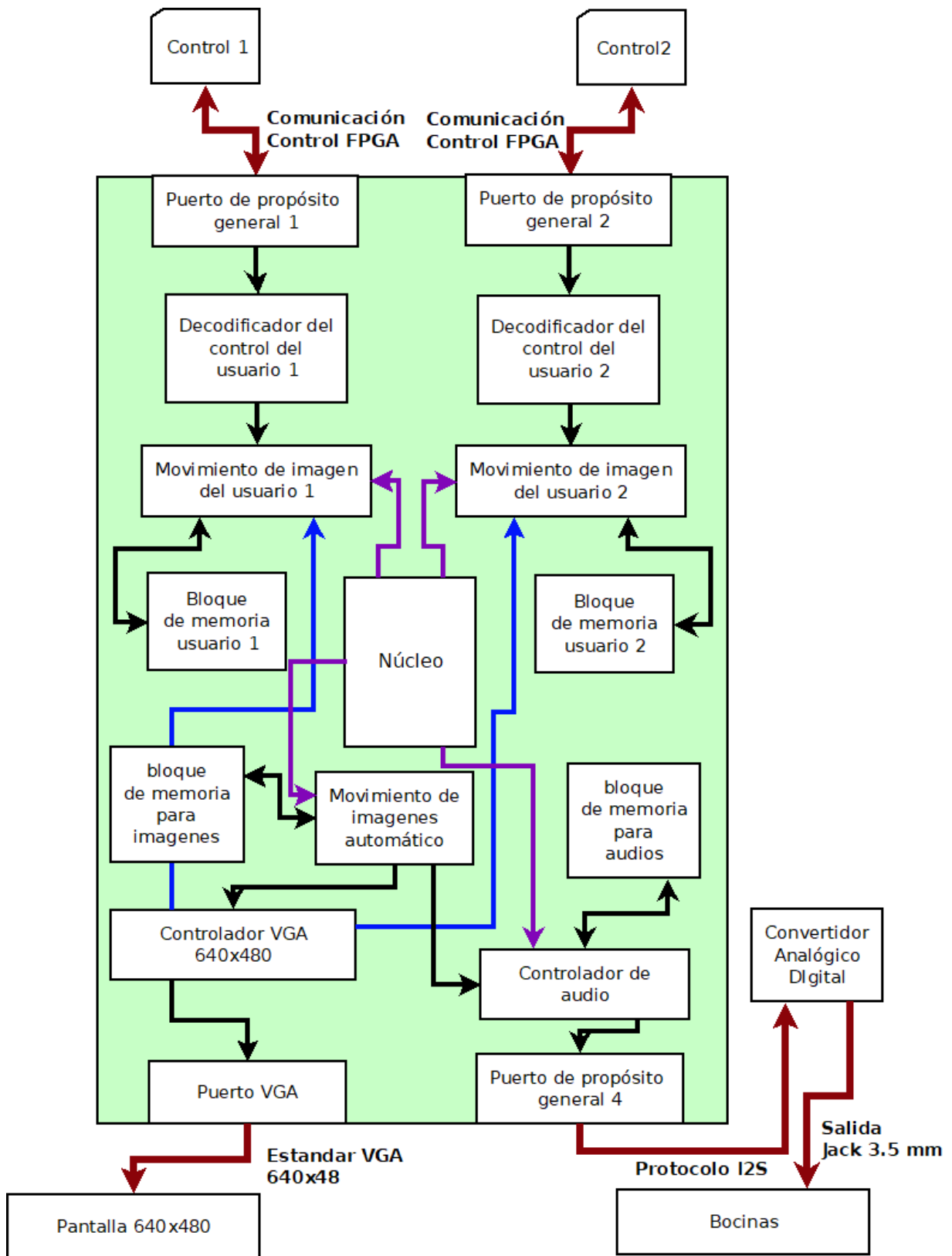


Figura 2. 1 Diagrama general del sistema de procesamiento de imágenes.

El controlador de audio va a recibir información en tiempo real para poder seleccionar que audio va a ser enviado por medio del protocolo I2S al convertidor digital-analógico, para ser reproducido por medio de un par de bocinas a través de un conector tipo jack de 3.5mm. Los detalles de todos los protocolos y estándares utilizados se van a explicar más a detalle a lo largo de este capítulo.

2.2 IMÁGENES DIGITALES Y AUDIO.

2.2.1 LAS IMÁGENES DIGITALES.

Las imágenes pueden manifestar la apariencia que pudiera tener un objeto real o imaginario, dándole al espectador la capacidad de observar y entender lo que el expositor trata de mostrar. Con el avance de la tecnología, se han obtenido mejores formas de generar y mostrar figuras digitales.

Una imagen digital es definida como una representación bidimensional de una ilustración a partir de un conjunto de valores del mismo tipo llamado matriz, el cual puede ser de tipo binario (1 pixel¹⁵ = 1 bit, donde 0 es negro y 1 es blanco), escala de grises (1 pixel = 1 byte, donde 0 es negro y 255 blanco, es decir, permite 256 niveles de gris) y a color (1 pixel > 1 byte, debido a que se requieren 3 valores de colores: rojo, verde y azul).

Al ser una imagen considerada como una figura matricial, se puede observar como un mapa de bits. Un factor que influye directamente en la estructura de las imágenes es su resolución, por esta razón, podría cambiar su tamaño y la cantidad de espacio en bits de memoria.

El tamaño de la imagen se representa con medidas bidimensionales (*largo * ancho*), donde se proporciona el dato total de píxeles que contiene la imagen (*píxeles = largo * ancho*).

¹⁵ El píxel es la unidad más pequeña de una imagen digital y está representado por un número para formar una imagen completa

Ejemplo:

La cantidad de pixeles contenidos en las imágenes 1 y 2 de las figuras 2.2 son las siguientes:

$$\text{Píxeles}_{\text{imagen 1}} = 150 * 267 = 40,050 \text{ píxeles}$$

$$\text{Píxeles}_{\text{imagen 2}} = 720 * 1280 = 921,600 \text{ píxeles}$$



(Imagen 1)



(Imagen 2)

Figura 2. 2 Ejemplo de baja (Imagen 1) y alta resolución (Imagen 2) en una misma imagen.

El espacio en memoria de la imagen 2 (921,600 píxeles) será mucho mayor al espacio designado para la imagen 1 (40,050 píxeles).

Una imagen digital puede ser guardada en celdas de memoria, siempre y cuando los recursos lo permitan. Para obtener el dato deseado, es necesario conocer una dirección donde fue asignada dentro de la memoria.

2.2.2 EL SONIDO

El sonido tiene diversas definiciones dependiendo el área que se estudie, en su sentido elemental se puede describir como la sensación producida en el oído por medio de vibraciones transmitidas por un medio elástico, por ejemplo, el aire o el agua.

Como cualquier fenómeno físico, las ondas de sonido tienen características propias que son necesarias para poder entenderlas y utilizarlas. Algunas de estas características son:

Frecuencia: el número de ciclos completos que puede realizar una onda durante un segundo y se mide en Hertz (Hz). Para el ser humano el rango audible se encuentra entre los 20Hz a los 20KHz.

Amplitud: es la altura máxima de la onda y corresponde a la intensidad del sonido, comúnmente es expresada en la cantidad de energía transportada.

Longitud de onda: distancia recorrida por una onda en un instante de tiempo.

El sonido puede almacenarse de forma analógica o digital. La primera consiste en convertir, por medio de un transductor, los cambios en la presión de aire a cambios de tensión eléctrica, estos cambios pueden almacenarse en cintas magnéticas como un casete o un disco de vinilo, sin embargo, estos sistemas tienden a desgastarse muy pronto y con el tiempo pierden su calidad.

El segundo tipo de almacenamiento es el digital. El audio digital divide la señal en pequeños instantes de tiempo denominados muestras y se les asigna un valor binario, este proceso se le llama muestreo del audio. Existen varios tipos de dispositivos que se pueden almacenar audio digital, algunos de ellos son: discos CD-ROM, discos CD-audios, DVD, disquetes, tarjetas de memoria tipo flash, etc.

2.2.3 CONVERSIÓN ANALÓGICO-DIGITAL

El proceso de muestreo es posible por medio de los convertidores analógicos-digitales, estos sistemas toman una muestra de la señal analógica y le asignan un valor binario que puede ser almacenado en una memoria. Un instante después, se toma una nueva muestra y se almacena

de nuevo, la frecuencia con la que se realiza este proceso debe ser al menos el doble de la máxima frecuencia de la señal original (Teorema de Nyquist).

Además del muestreo de la señal, el proceso de conversión analógico-digital del audio tiene otros dos pasos para completarlo correctamente los cuales son los siguientes:

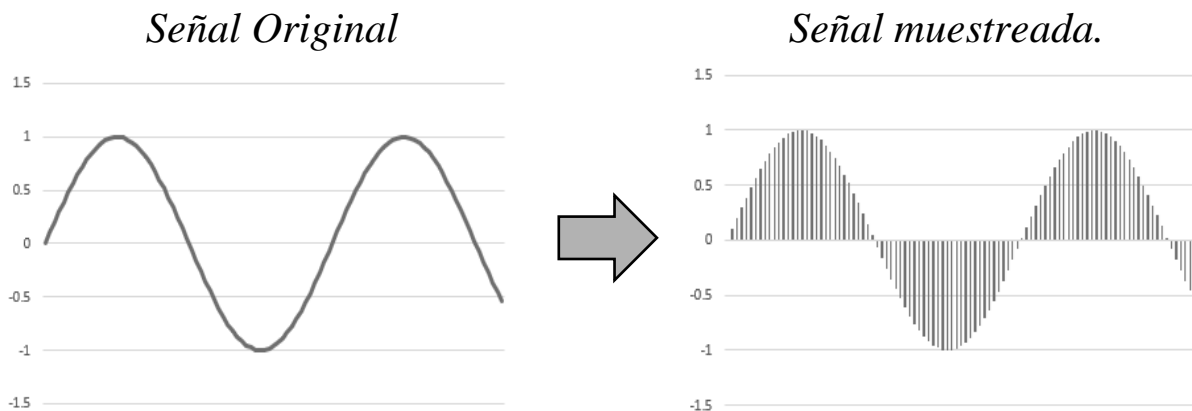


Figura 2. 3 Muestreo de una señal de audio.

Cuantización: cuando el proceso de muestreo terminó las muestras tomadas tienen valores continuos (voltajes) por lo tanto deben ser convertidos a valores discretos.

Codificación: ya que se tiene cuantizada la señal, cada muestra debe ser convertida a su valor binario. La resolución del audio dependerá de cuántos bits se utilicen para codificar la señal y después almacenarla.



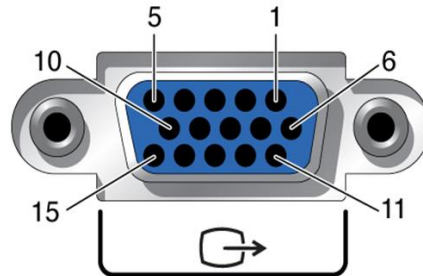
Figura 2. 4 Flujo de conversión Analógico-Digital.

2.3 ESTÁNDARES Y PROTOCOLOS DE COMUNICACIÓN PARA LOS PUERTOS DE ENTRADA Y SALIDA.

2.3.1 ESTÁNDAR VGA.

El estándar VGA[17] es un sistema de transmisión analógica de imágenes por medio de los 3 componentes RGB (Red, Green and Blue). Creado por IBM® para transmitir imágenes hacia las pantallas o proyectores a través de un conector de 15 pines.

El puerto VGA (Video Graphics Array) es un conector disponible en la tarjeta de desarrollo como un puerto de entrada y salida de información. El cual tiene sus conexiones físicas con el FPGA como se muestra en la figura 2.5.



Pin 1: Canal Rojo	Pin 6: Retorno Rojo	Pin 11: N/C
Pin 2: Canal Verde	Pin 7: Retorno Verde	Pin 12: SDA
Pin 3: Canal Azul	Pin 8: Retorno Azul	Pin 13: HSync
Pin 4: N/C	Pin 9: + 5V CD	Pin 14: VSync
Pin 5: GND Tierra	Pin 10: GND Tierra	Pin 15: SCL

N/C = No conectado.

Figura 2. 5 Diagrama de conexiones del puerto VGA con la FPGA.[18]

La emisión de energía en el espectro RGB es proporcional al voltaje correspondiente a las señales de colores Rojo, Verde y Azul. Cada punto coloreado en la pantalla es un píxel. La pantalla despliega pixeles comenzando en la esquina superior izquierda, desplazándose a la derecha, línea por línea, hasta la parte inferior de la pantalla. Un pulso de sincronía horizontal se encarga de sincronizar cada nueva línea. Una vez que se alcanza la parte inferior de la

pantalla, un pulso de sincronía vertical provocará un nuevo comienzo desde la parte superior izquierda como se observa en la figura 2.6.

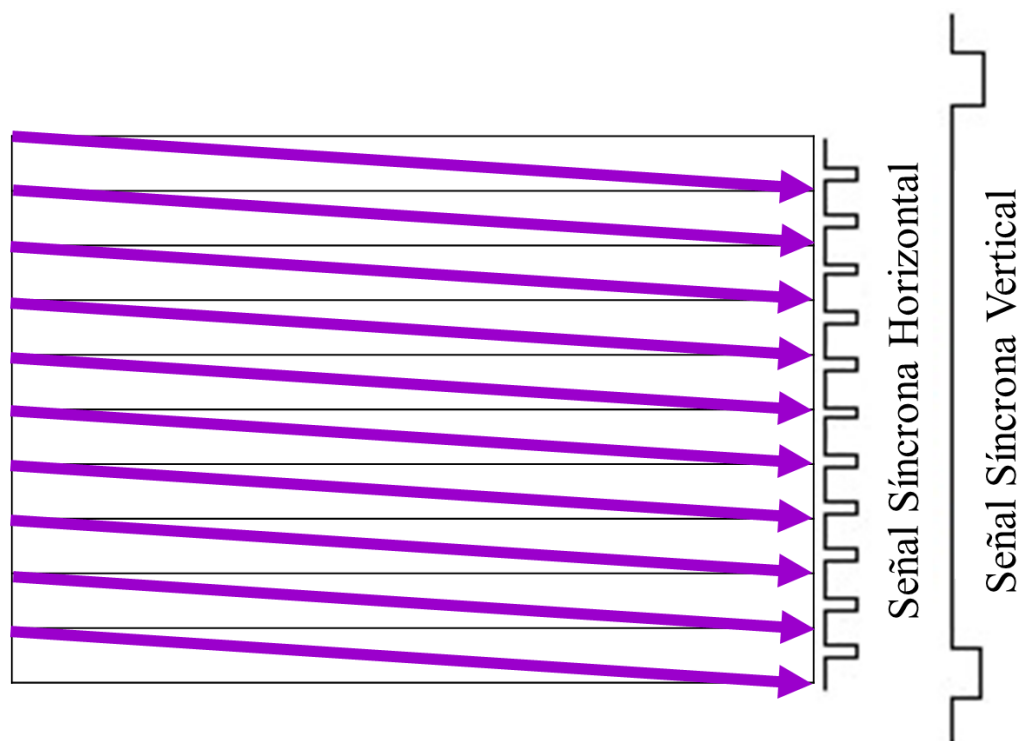


Figura 2. 6 Diagrama de señales síncronas en pantalla.

Una vez explicada la importancia de las señales síncronas, la figura 2.7 muestra todas las regiones que contiene una señal síncrona general:

- T_s es el periodo de la señal *Synchronous* (Síncrona).
- T_{disp} es la señal *Display Timing* (Tiempo en pantalla (HV para la señal síncrona horizontal y VV para la señal síncrona vertical).
- T_{fp} es la señal *Front Porch* (Porción Frontal no Visible).
- T_{bp} es la señal *Back Porch* (Porción Posterior no Visible).
- T_{pw} es el *Pulse Width* para identificar las señales síncronas (Ancho del Pulso).

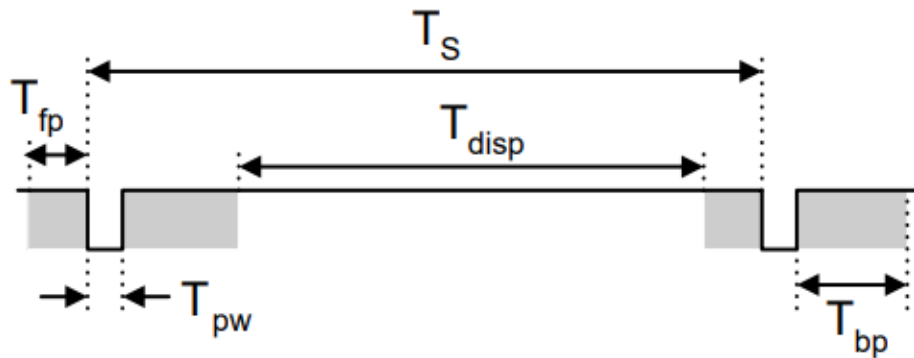


Figura 2. 7 Diagrama general de tiempos para las señales síncronas.

Los pulsos síncronos se basan en los requerimientos técnicos para la resolución 640x480, [17] para su buen funcionamiento se debe tener presente lo siguiente:

1. **Señal de sincronía horizontal (HS):** se compone de 4 regiones, como se muestra en la figura 2.8:
 - Pulso de sincronía (SP).
 - Back Porch (BP).
 - Video horizontal (HV).
 - Front Porch (FP).
2. **Señal de sincronía vertical (VS):** se compone de 4 regiones como se muestra en la figura 2.8:
 - Pulso de sincronía (SP).
 - Back Porch (BP).
 - Video Vertical (VV).
 - Front Porch (FP).
3. **Las entradas al monitor Rojo, Verde y Azul:** son señales de tipo analógicas, Algunas tarjetas de desarrollo ya poseen un convertidor digital analógico como un convertidor digital analógico con resistencias ponderadas.

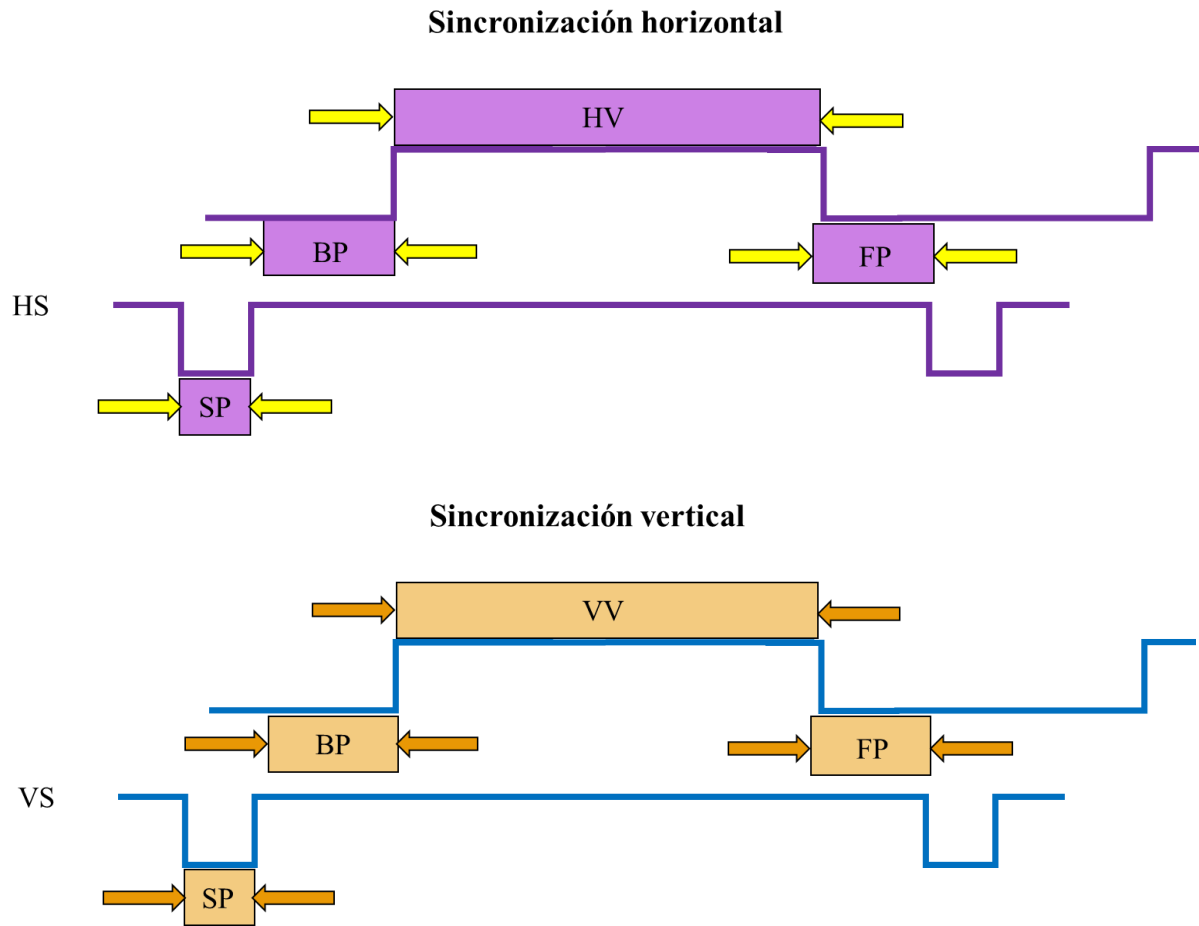


Figura 2. 8 Señales de sincronización horizontal y vertical.

2.3.2 PROTOCOLO I2S.

El protocolo I2S (por sus siglas en inglés *Integrated Interchip Sound*) es un estándar de transferencia de audio, ampliamente utilizado para transmitir datos de audio desde un microcontrolador, DSP o FPGA hacia un decodificador de audio para reproducir su contenido. Las especificaciones de este protocolo fueron descritas en el documento “*I2S bus specification*[19]” por Philips Semiconductor en febrero de 1986 y tuvo su última revisión en junio de 1996. Al utilizar este protocolo se minimiza el número de conexiones requeridas para entablar la comunicación entre dos dispositivos utilizando únicamente 3 líneas para la sincronización. El diagrama básico de la conexión de entre dos dispositivos comunicados por I2S se muestra en la figura 2.9.

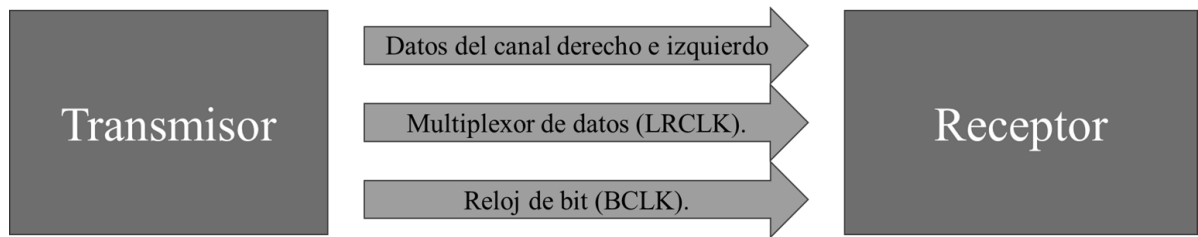


Figura 2. 9 Diagrama básico de conexión por medio del protocolo I2S.

En el protocolo I2S se puede transmitir audio entre 8 y 32 bits de resolución por canal (esta cantidad puede variar dependiendo el módulo con el que se esté trabajando), esta información va a ser enviada a través de la primera línea de datos que se llama SDATA (Serial Data).

En el caso de esta transmisión, la línea SDATA está multiplexada en el tiempo, por lo que primero se enviarán todos los bits de datos del canal izquierdo comenzando por el bit más significativo (MSB) y hasta que se haya enviado su bit menos significativo (LSB) comenzará a enviarse el MSB del canal derecho y al enviarse el LSB de este canal terminará la transmisión de esa muestra para repetir el proceso con la siguiente.

Hay dos formas de sincronizar el canal izquierdo y derecho por medio del protocolo I2S, sus diagramas de tiempos se explican a continuación.

- **Transmisión I2S.**

Para sincronizar los canales de audio se tiene el reloj que va a multiplexar los canales en el tiempo (LRCLK), este reloj va a tener la frecuencia a la cual se haya muestreado la señal de audio que se esté transmitiendo. Si el LRCLK se encuentra en un nivel alto se va a transmitir el canal izquierdo, de lo contrario se tiene que transmitir el canal derecho. Cada bit, de cada canal, se va a transmitir durante un determinado tiempo, que va a ser igual al inverso de la frecuencia de muestreo dividida entre los bits de resolución de la muestra por 2. Por ejemplo, si se tiene una frecuencia de muestreo de 44.1 KHz y cada canal de audio tiene una resolución de 16 bits el tiempo de transmisión de cada uno de los 16 bits enviados será:

$$Tiempo\ de\ bit = \frac{1}{\frac{44.1KHz}{16(2)}} = 708.6168ns$$

El tiempo de bit servirá para calcular la frecuencia que va a tener el reloj de bit (BCLK), para obtenerla únicamente se calcula el inverso del tiempo obtenido:

$$F(BCLK) = \frac{1}{708.6168nS} = 1.4112MHz.$$

Una vez que se tienen las dos señales que van a sincronizar la transmisión de audio, en el caso del protocolo I2S, se debe asegurar que el LRCLK haga el cambio de un estado a otro cuando el BCLK haga el cambio de nivel alto a nivel bajo (flanco de bajada). El MSB del canal izquierdo va a comenzar a enviarse en el primer flanco de bajada del BCLK después de que el LRCLK haya cambiado de un nivel alto a un nivel bajo y el LSB del canal izquierdo va a transmitirse en el flanco de bajada del BCLK el mismo que indica el flanco de subida de la señal LRCLK. Por lo tanto, el canal derecho va a comenzar su transmisión en el primer ciclo de bajada después de este cambio, es decir, el inicio del envío de información de cada canal tiene un flanco de reloj de retardo[20]. El diagrama de tiempos de este proceso se observa en la figura 2.10.

- **Transmisión I2S justificado a la izquierda.**

Para el modo I2S justificado a la izquierda la frecuencia de las señales LRCLK Y BCLK se mantienen, la diferencia entre este modo y el anterior es el tiempo de inicio de transmisión. El bit más significativo del canal izquierdo va a transmitirse al comenzar el flanco de subida del LRCLK y durante el flanco de bajada del BCLK correspondiente por lo que el paquete de datos de ese canal terminará de transmitirse en el flanco de bajada del LRCLK para que el canal derecho se transmita durante el tiempo que dure la señal en nivel bajo[20]. La imagen 2.11 muestra el diagrama de tiempos que describe el comportamiento de este tipo de transmisión.

Los dos modos de transmisión I2S mostrados anteriormente, pueden enviar audio con la misma calidad, queda a elección del usuario decidir cuál de los dos usar, por lo tanto, de aquí en adelante cuando se mencione la transmisión I2S se va a referir al modo justificado a la izquierda.

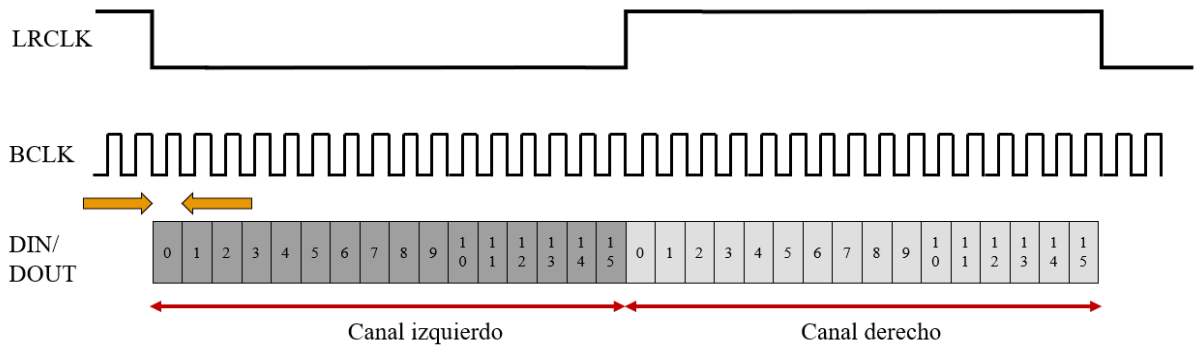


Figura 2. 10 Diagrama de tiempos de transmisión I2S.

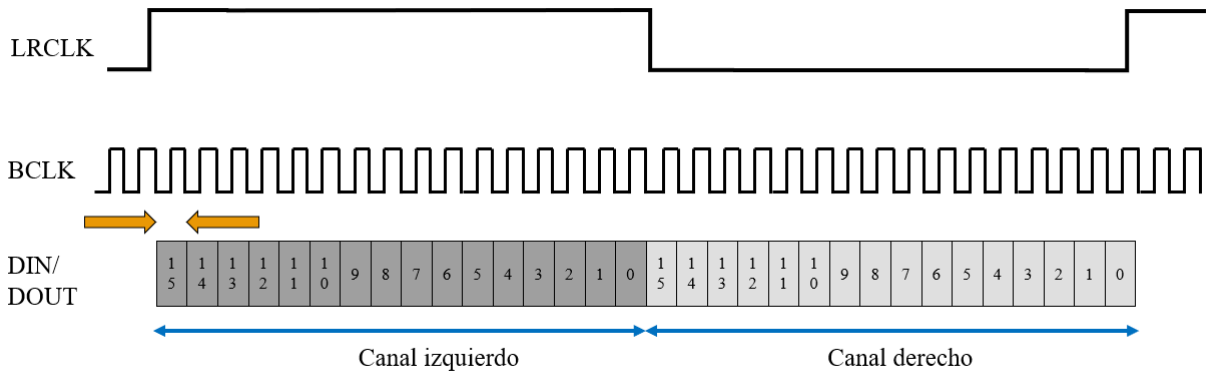


Figura 2. 11 Transmisión I2S justificada a la izquierda.

Como se puede observar, los protocolos y estándares están delimitados a las características que los rigen. El trabajo se centra en desarrollar un sistema de procesamiento de imágenes que cumpla con esas peculiaridades, considerando complementarlo con la aplicación. Para lo cual, en los siguientes capítulos se abordan definiciones, diseños e implementación de la aplicación como tal.

CAPÍTULO 3 DISEÑO DE UNA APLICACIÓN PARA VIDEOJUEGOS

En el capítulo anterior se describió el diseño general del sistema, conceptos necesarios para el procesamiento de imágenes, algunos estándares y protocolos de comunicación, que servirán como base para la implementación de la aplicación.

En este capítulo se definirán y explicarán las funcionalidades generales del sistema, la justificación de la implementación de la aplicación, el despliegue de las imágenes y la jerarquía que tiene la imagen sobre la pantalla, definiciones generales sobre algunos módulos y las máquinas de estados correspondientes del flujo de la aplicación.

3.1 PLANTEAMIENTO DE LA APLICACIÓN

El diseño de la aplicación está implementado como un sistema de videojuegos de los años 80, el cual cuenta con un sistema de mandos de entrada y 2 tipos de salida de datos; la parte visual que está desplegado en una pantalla de 640x480 de resolución y la auditiva que se puede apreciar sobre una bocina conectada por medio de un conector Jack de 3.5 mm como se puede ver en la figura 3.1. Por el momento solamente se hablará de la funcionalidad, en el siguiente capítulo se detallará sobre las características físicas.

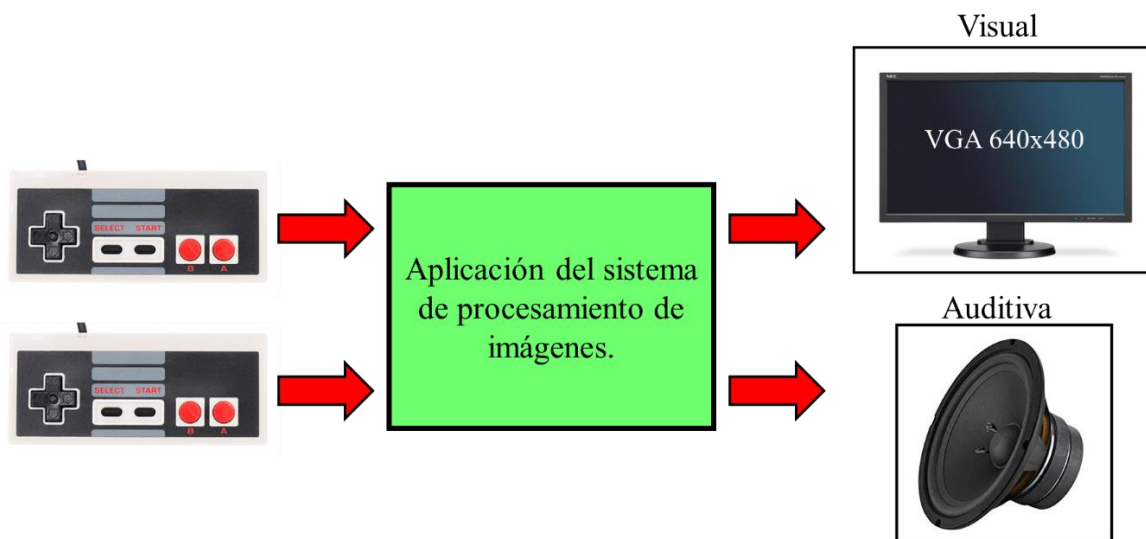


Figura 3.1 Modelo general del sistema completo.

El funcionamiento de la aplicación está basado en un sistema apoyado en el concepto de microservicios¹⁶, donde se separan todas las funcionalidades que requiere la aplicación en pequeños módulos independientes para la funcionalidad del sistema. Otro concepto en el que se basa el sistema es la orquestación¹⁷ de los módulos, este es utilizado para organizar y dirigir a los módulos en un mismo proceso, proporcionando señales a los estados para que los módulos funcionen sobre el mismo flujo de la aplicación.

El flujo que mantendrá la aplicación se muestra descrito en la figura 3.2, donde se coloca el proceso general del seguimiento que realiza el sistema. El flujo sigue el ciclo mostrando la presentación, realizar la validación del inicio del sistema, preparación de todos los módulos para iniciar sus procesos, lectura de datos de entrada y bifurcación de 2 procesos; desplegar imágenes correspondientes en el proceso de cada módulo y enviar los audios de los respectivos sonidos. La segunda vía consulta los estados de cada nivel y entra en un proceso de validación donde verifica si ha finalizado, de ser así muestra si el estado de la finalización fue exitoso o no, de lo contrario seguirá realizando los procesos hasta que finalice.

3.2 DISEÑO DE LA APLICACIÓN.

El diseño propuesto es un sistema de procesamiento de imágenes aplicado a un sistema de videojuegos del tipo de los años 80. El diseño se definió en 3 etapas:

- La presentación del sistema: este proceso se encarga de proporcionarle al usuario el nombre del sistema y el tiempo necesario para que decida iniciar el proceso.
- Flujo definido del proceso: en este punto el usuario deberá seguir el flujo presentado por el sistema, pasando por diversos niveles. Durante este proceso estarán funcionando todos los módulos necesarios para mostrar las imágenes

¹⁶ Los sistemas basados en microservicios es un estilo de arquitectura donde se dividen los componentes y son independientes entre sí, donde cada módulo funciona en conjunto para llevar a cabo las tareas predeterminadas por el desarrollador.

¹⁷ La orquestación es un proceso que lleva el control de varios servicios involucrados entre sí para la realización de una tarea, donde se le delega la mayoría de la responsabilidad a un coordinador central y los demás servicios solamente se dedican a sus procesos individuales.

correspondientes, indicadores del seguimiento del proceso y revisión constante de estados del sistema.

- Finalización del sistema: durante el proceso, este puede ser interrumpido o finalizado exitosamente. Mientras se realizan validaciones de estados del proceso, el sistema decide si continúa o finaliza, indicándole al usuario si cumplió con los objetivos o fallo en los intentos.

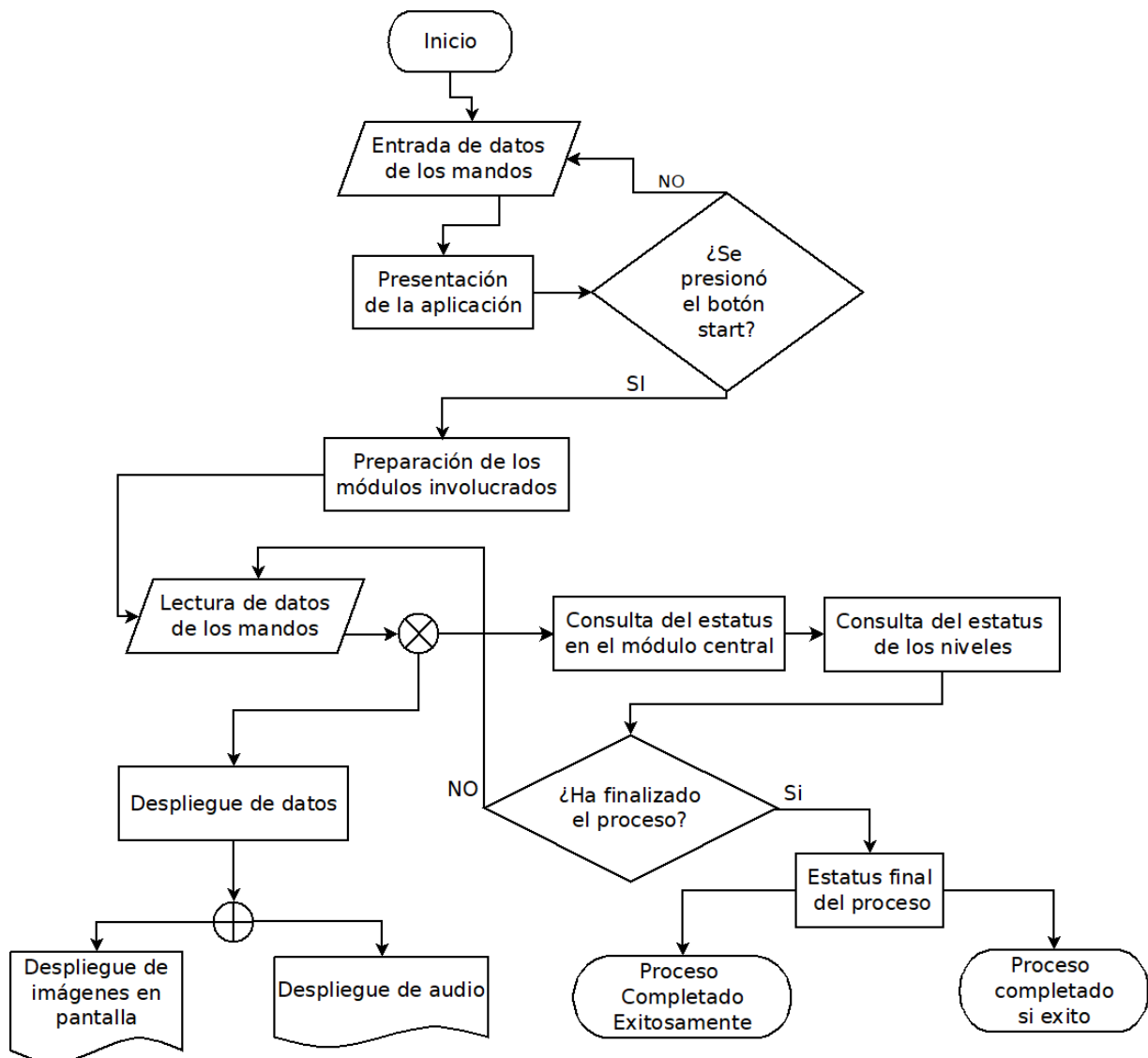


Figura 3. 2 Diagrama de flujo general del proceso de la aplicación.

Como se mencionó anteriormente, el sistema está basado en dos conceptos importantes: el uso del modelo de microservicios y de un sistema central orquestador. El modelo de microservicios fue retomado para el desarrollo de sistemas independientes que se utilizan dentro de la aplicación que estuviesen trabajando al mismo tiempo. Para el desarrollo del flujo fue necesario separar cada proceso en un módulo individual que estuviese a cargo de ciertas tareas, por ejemplo: el despliegue de las imágenes, el controlador del VGA, el controlador encargado del envío del audio por protocolo I2S, el control de los divisores de frecuencia, etc. Al manejar este modelo es necesario que todos los módulos estén funcionando al mismo tiempo, así que la ayuda de lenguajes de descripción de hardware permite usar el paralelismo, dándole oportunidad de utilizar este modelo de microservicios.

El otro concepto utilizado es el sistema central orquestador, que tiene como finalidad obtener todas las señales necesarias de los módulos implementados y enviar señales de sincronía que definen el proceso que llevará la aplicación como: el estado actual del proceso o si ha finalizado exitosamente o no.

En los siguientes subtemas se describirá en detalle el sistema central orquestador, conocido como el núcleo del juego, la funcionalidad de algunos módulos para el proceso de la aplicación, el despliegue de las imágenes y el funcionamiento de la obtención de los datos de entrada y de la salida del audio.

3.2.1 DEFINICIÓN DE LOS MÓDULOS DEL SISTEMA

La aplicación implementará sistemas encargados de procesar ciertas tareas, las cuales son esenciales para el proceso y duración de la aplicación. Para lograrlo es necesario definir algunos módulos que estarán involucrados con la funcionalidad del sistema:

1. *Divisor de frecuencia*: este sistema se encargará de manejar las frecuencias necesarias para cada módulo que requiera una frecuencia diferente a la principal.
2. *Señales VGA*: este módulo tiene como tarea el proporcionar las señales síncronas correspondientes al estándar VGA640x480 en tiempo real.
3. *Laberinto*: la funcionalidad que tiene este sistema es delimitar el área permitida donde los usuarios puedan desplazarse y cumplir los objetivos de la aplicación.

4. *Jugadores y enemigos*: estos sistemas tienen la tarea de desplazarse sobre el área permitida en pantalla, los primeros son los que estarán directamente en contacto con los usuarios, en cambio los segundos estarán impidiendo a los usuarios cumplir con los objetivos.
5. *Puntaje*: considerado uno de los indicadores de información en pantalla, mostrándoles a los usuarios el progreso obtenido durante la duración de la aplicación.
6. *Temporizador*: otro indicador de información para el usuario, mostrando el tiempo restante que tiene el usuario para finalizar exitosamente el proceso delimitado por el sistema.
7. *Indicador de oportunidades*: este indicador de información le comparte a los usuarios las oportunidades que tienen para finalizar exitosamente el proceso de la aplicación.
8. *Controlador de VGA*: este sistema tiene como tarea el gestionar el despliegue de las imágenes según su jerarquía.
9. *Presentación*: este sistema se encarga de mostrarle al usuario el nombre de la aplicación, esta será desplegada hasta que el usuario inicie el proceso del sistema.
10. *Finalización de la aplicación*: tiene como tarea el estar pendiente si el sistema ha finalizado para mostrar una imagen de terminación.
11. *Núcleo del sistema*: este sistema orquestador tiene como propósito el recibir información necesaria de otros módulos, interpretarla y compartir a todos los módulos restantes el flujo que debe seguir el sistema sobre los estados de la aplicación, respondiendo si ha finalizado exitosamente o no.
12. *Audio*: su tarea principal es obtener los datos guardados dentro de un repositorio y transmitirlos mediante el protocolo I2S.

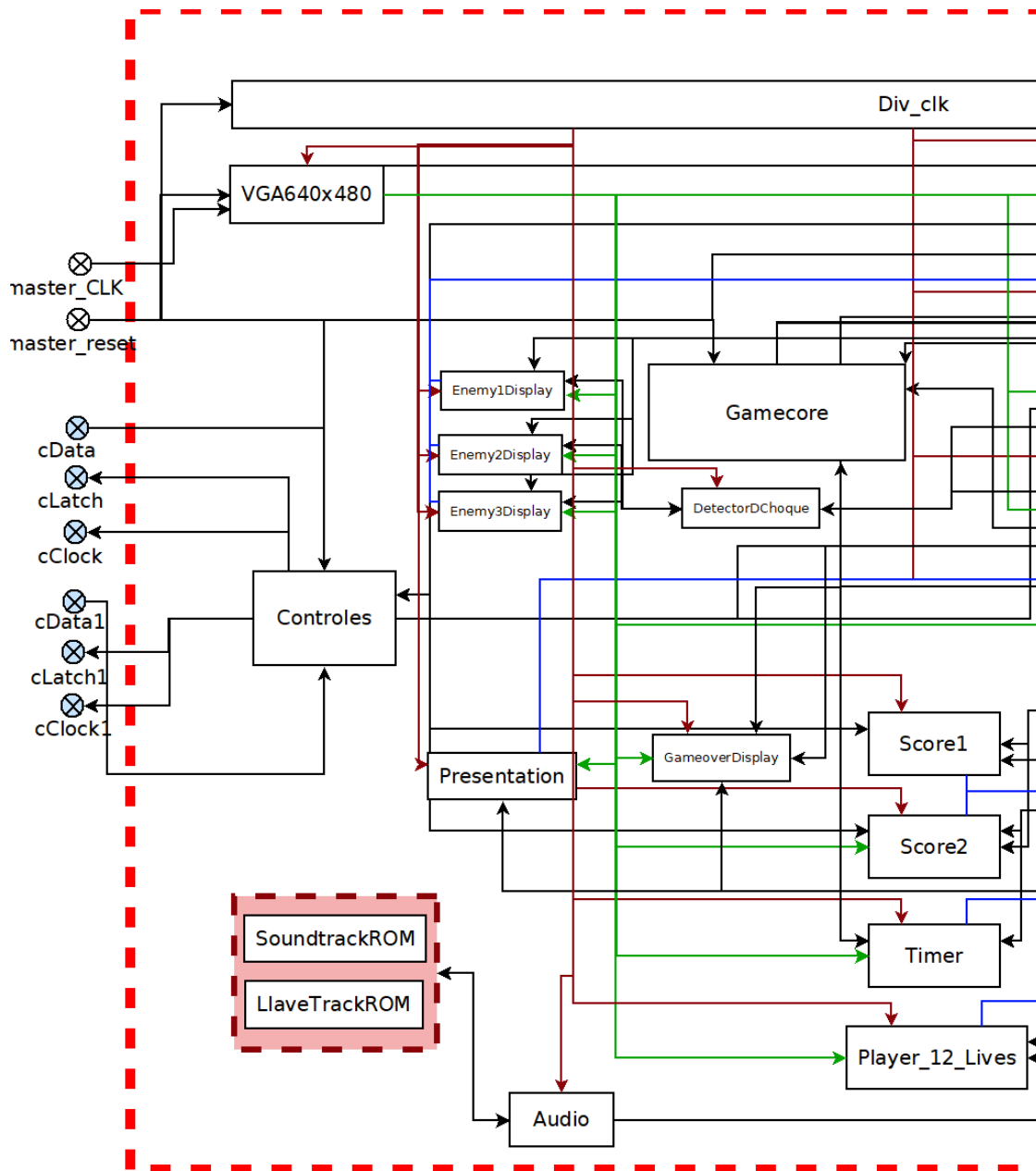
Como se puede observar, existen varios módulos especializados en una tarea, por mencionar los módulos del número 3 al 10, también es parte del proceso el obtener los datos correspondientes de cada imagen desde un repositorio e indicarle al controlador de imágenes su momento de despliegue en pantalla.

3.2.1 DISEÑO CONCEPTUAL (TOP LEVEL DESIGN)

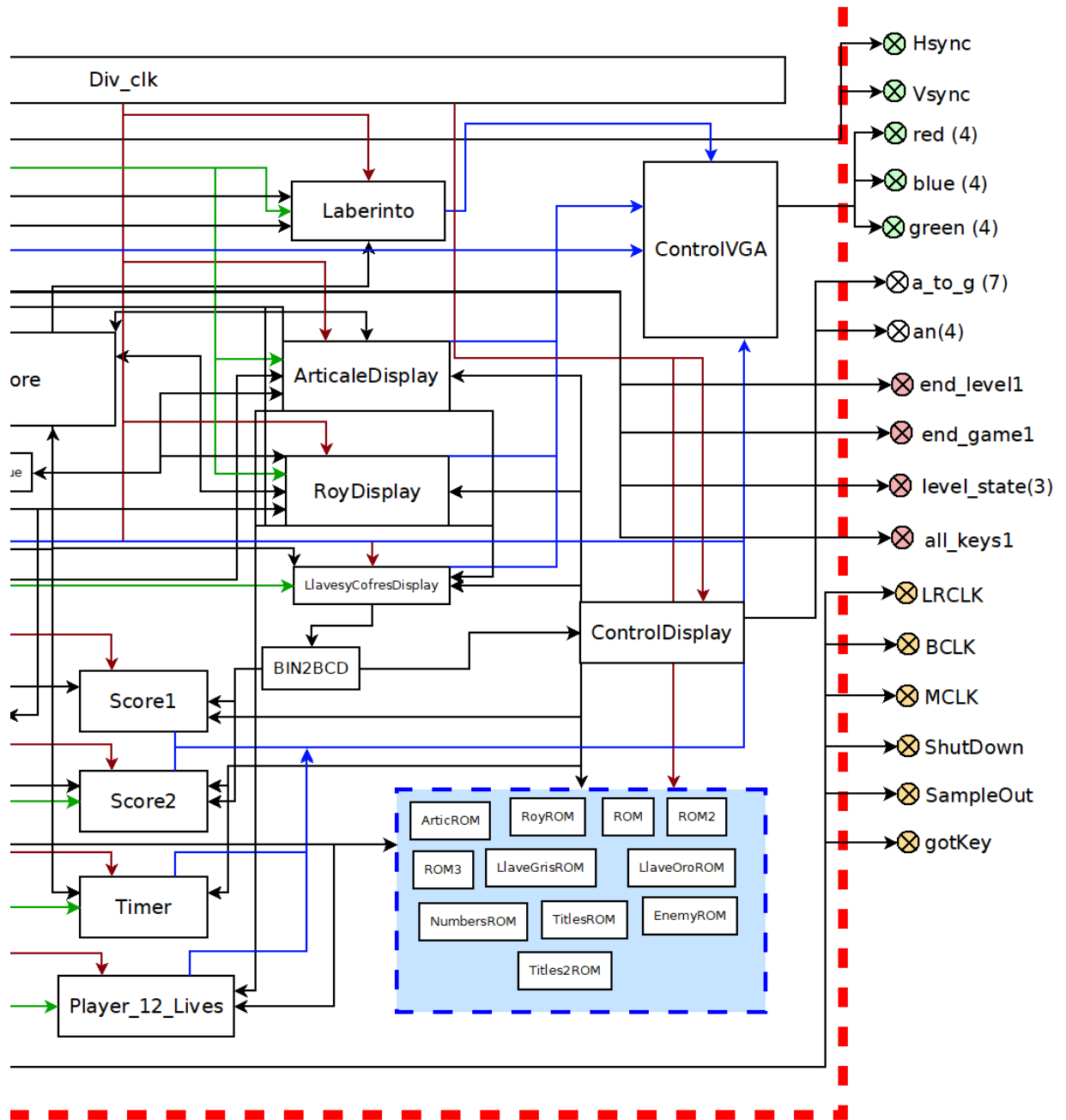
Como se mencionó anteriormente, el diseño abarca los 2 conceptos base que ayudarán para una mejor implementación sobre el sistema. Los módulos que se implementarán sobre el

sistema serán los encargados de cada imagen, alguna funcionalidad para el despliegue, los estándares y protocolos, el controlador de imágenes y el sistema orquestador.

Una vez definidas las funciones de cada módulo, se realiza un diseño de nivel superior (Top Level Design) en donde se representa todo el sistema, en el caso de la aplicación de procesamiento de imágenes como varios bloques que desempeñan la tarea de recibir, procesar y enviar señales. Este diseño se puede observar en la figura 3.3, donde se muestran los módulos



a)



b)

Figura 3. 3 Diseño de alto nivel (Top level design).

involucrados, sus interconexiones entre sí, los bloques de memoria y los puertos de entrada y salida de la aplicación (donde las señales de color rojo son señales de reloj provenientes del módulo de divisor de frecuencia, las de color verde son las señales síncronas provenientes del módulo de VGA y las de color azul son las señales dirigidas al módulo de controlador VGA).

3.3 NÚCLEO DE LA APLICACIÓN

Un sistema de este tipo involucra muchos datos para el funcionamiento correcto del sistema. El problema que se tiene es el envío de datos a diversos módulos del sistema con el objetivo de realizar un proceso. Para este caso, el sistema de procesamiento de imágenes funciona por sí solo, registra el puntaje cuando se obtienen las llaves, muestra el tiempo restante y despliega cuantas veces se ha perdido una vida. Revisando hasta este punto se tiene un proceso simple, por lo que se buscó generar un flujo donde se tengan involucrados los niveles, condiciones y estados para formar un nuevo proceso que le permitiera al jugador ir avanzando y al mismo tiempo subir la dificultad. Lo primero que se debe considerar son los recursos de los que se dispone, después definir el proceso que se desea realizar, es decir, la cantidad de niveles que se desean ingresar, las condiciones que tomará el sistema para considerar si el juego finaliza o continúa, etc. Y como último punto, considerar qué tan flexible es el modelo base para ser modificado al nuevo sistema final.

La primera etapa del desarrollo cuenta con pequeños módulos trabajando de forma concurrente, lo que se busca es un sistema central que se encargue de orquestar o dirigir el proceso de la aplicación final. Las ventajas de utilizar un sistema que realice un proceso central es delegar gran parte de la responsabilidad del nuevo proceso orquestado que se busca, es decir, dirigir a los demás sistemas para cumplir un cierto objetivo que es realizar el proceso de la aplicación. De esta manera no se altera drásticamente al sistema base, solamente se crean nuevas puertas de entradas y salidas de datos necesarios para orquestar el proceso. Los módulos actúan como emisores y receptores de información mientras que el sistema central se encarga de coordinar todos esos datos y comunicar al resto de los sistemas entre sí. Si no se manejara un sistema que realice el proceso central, cada uno de los módulos tendrían que ser modificados de tal manera que, si se requiere una señal de otro sistema, se tendrían que levantar una conexión directa con el otro módulo. Generando una red de conexiones punto a punto, también conocida como integraciones de sistemas tipo *espaguetti*. En la figura 3.4 se puede ver una comparación de ambos tipos de integraciones de sistemas.

Una vez realizadas ciertas modificaciones adecuando a los módulos, de tal manera que se puedan conectar con el sistema integrador central enviando y recibiendo señales para poder

identificar en pantalla los estados de los niveles creados, como por ejemplo el color de algunas imágenes. En efecto, la aplicación podrá realizar modificaciones o anexos de nuevos módulos al sistema, por lo siguiente, se considerarán como requerimientos esenciales de este sistema central para poder operar correctamente.

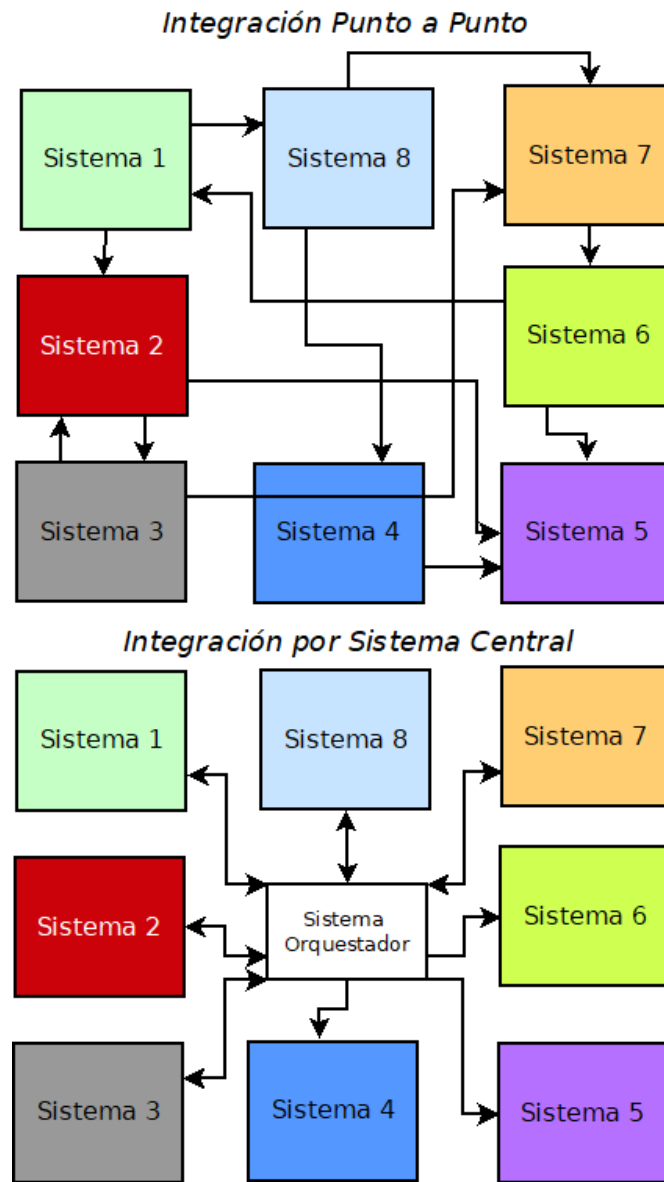


Figura 3. 4 Integración por punto a punto e integración por sistema centralizado.

El funcionamiento del sistema central orquestador se muestra en la siguiente máquina de estados de la figura 3.5, donde se obtienen algunos datos de otros módulos implementados en el sistema y se valida si cambia de estado del proceso o finaliza la aplicación. Como se

muestra en la máquina de estados, el sistema sigue un flujo y, permaneciendo en algún estado, son enviadas señales a los demás módulos indicándoles el estado o el proceso de la aplicación. Durante la revisión esta puede ser interrumpida por ciertos indicadores y finalizar de forma incompleta. En caso contrario, llegará al fin del flujo finalizando de forma exitosa.

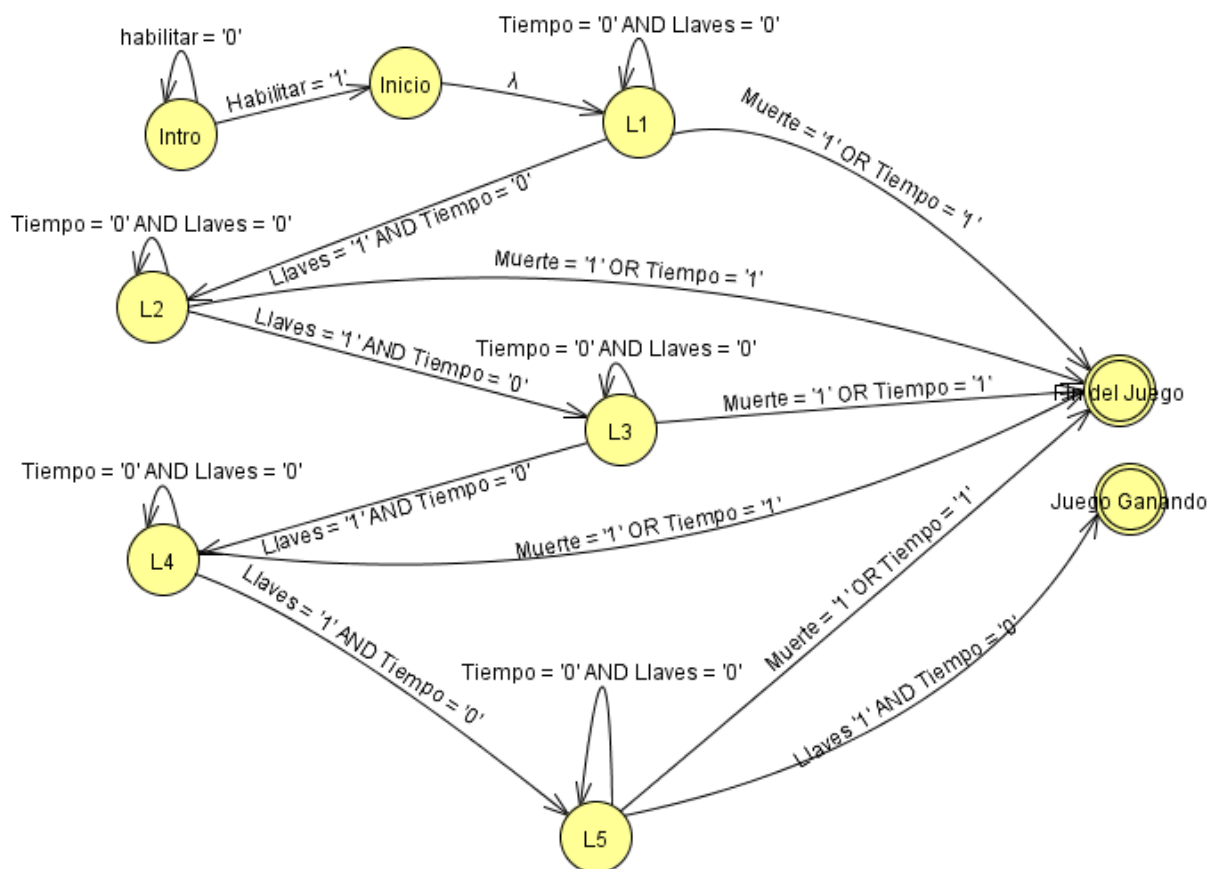


Figura 3. 5 Máquina de estados para el sistema integrador central.

3.4 CONTROLES

Para darle más realismo a la interacción con el jugador, se añadió un módulo que interactúe con los mandos de Nintendo® de su famosa consola Nintendo Entertainment System®.

Se hizo uso de los controles mostrados en la figura 3.6 [21] por las características que tienen: la cantidad de botones es la suficiente, la forma de obtener los datos de los mandos no

requiere muchos recursos, le da al usuario la experiencia que está utilizando una aplicación de los años 80.

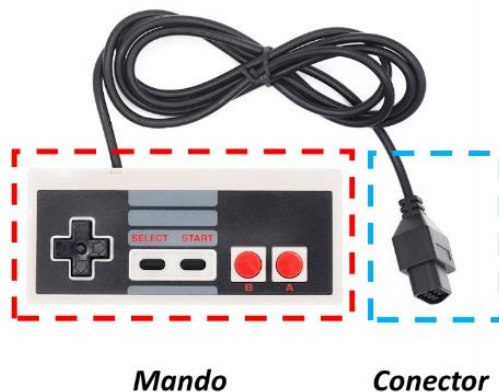


Figura 3. 6 Mando para puerto de entrada de datos.

El diagrama de tiempos que se muestra en la figura 3.7, corresponde con la señal enviada por los mandos, obteniendo los botones presionados durante el proceso. El módulo controles se encarga de enviar las señales *Latch* y *Pulse*. El mando procesa esas señales y envía la señal de *Data* (proceso visto en el capítulo anterior).

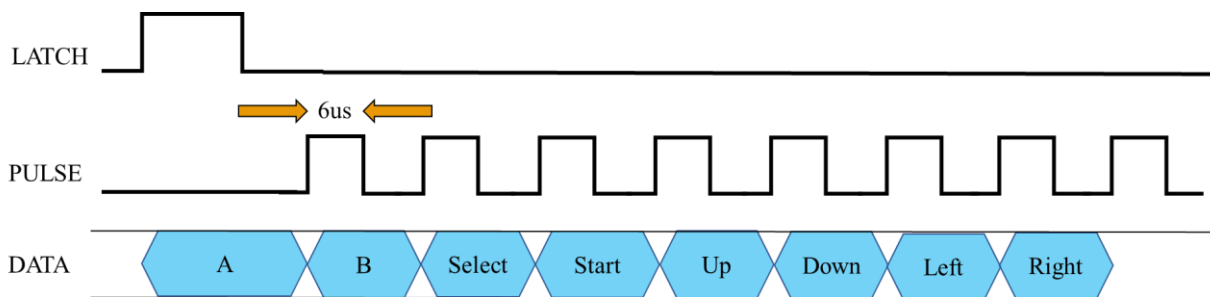


Figura 3. 7 Diagrama de tiempos para el envío de datos por los mandos.

Conociendo el funcionamiento del mando, se utilizaron los botones ubicados del lado izquierdo (Up, Down, Left y Right) que le permiten mover al usuario las imágenes sobre la aplicación y el botón de Start que se utiliza para iniciar la aplicación. Con base en lo establecido anteriormente, se diseñó la máquina de estados que se muestra en la figura 3.8, la cual sigue el siguiente proceso: para iniciar en el estado *Asegurar*, se envía una señal en nivel alto hacia los

mandos, únicamente cambiará hacia el estado *Listo* si el contador de tiempo es mayor a la cantidad de 11 conteos. El contador de tiempo incrementará cada $1\mu s$, hasta que haya acumulado 12 conteos. Se emplea una frecuencia de 1Mhz, un contador de 0 a 11 (12 conteos), al momento que el contador sea mayor de 12 conteos, se reinicie y vuelva a contar. Obteniendo el periodo de $12\mu s$ que requiere los mandos.

Durante el estado *Listo*, únicamente se obtiene el valor del botón presionado del mando, donde será guardado en el bit menos significativo del vector. Inmediatamente, al llegar el siguiente bit, el anterior será desplazado hacia la izquierda dentro del vector. Permanecerá en este estado únicamente si el contador de tiempo es mayor a 3 conteos.

Una vez desplazado el primer bit del valor obtenido en el estado *Listo*, este se cambiará hacia el estado *Leer y desplazar*, donde realizará un proceso que validará si se han obtenido los 7 dígitos restantes del mando. Además, este mismo proceso esperará el tiempo necesario, enviando una señal de reloj que cambia de nivel cada $6\mu s$. Una vez que el contador de bits sea mayor a 7, regresará al estado *Asegurar*, indicando que se obtuvieron correctamente los valores correspondientes de los 8 botones, y este reiniciará el proceso de la máquina de estados para obtener nuevamente los siguientes 8 valores correspondientes del mando.

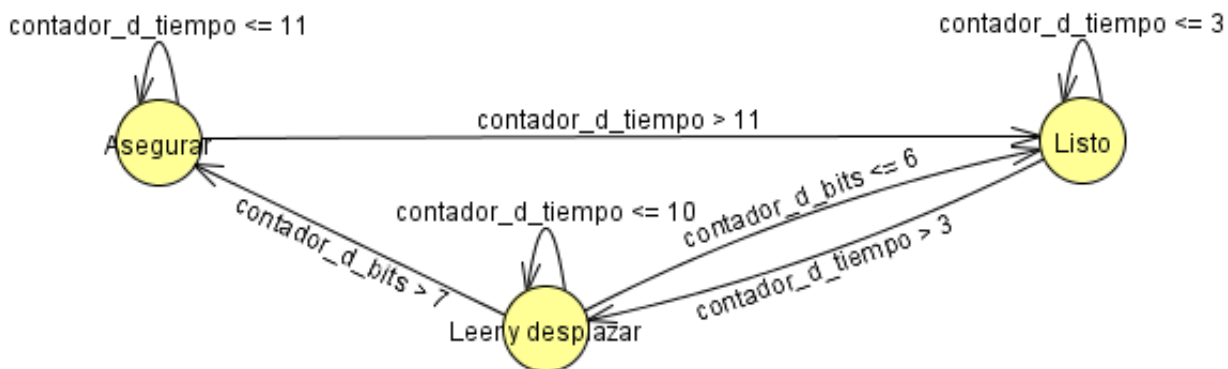


Figura 3. 8 Máquina de estados de los mandos de entrada.

3.5 AUDIO

En esta sección se describe el proceso para reproducir el audio de la aplicación durante toda la partida por medio de la máquina de estados mostrada en la figura 3.9. El siguiente proceso describe el funcionamiento de la transmisión de los datos por medio del protocolo I2S.

La máquina comienza en el estado de *Inicio*, donde va a permanecer hasta que las señales BCLK Y LRCLK, explicadas en el capítulo 2, estén sincronizadas de acuerdo con su diagrama de tiempos. Al coordinarse los flancos de subida y bajada de ambas señales se hace la transición al estado *Cambio*, donde enviará el primer bit de la señal de audio y posteriormente hacer un corrimiento a la señal, después pasará al estado de *Retardo* donde va a mantenerse hasta el próximo flanco de bajada del BCLK y repetir el proceso para enviar los 16 bits que componen la muestra y aumentará en uno la dirección de memoria solicitando la siguiente parte de la señal y repetir el proceso con sus respectivos 16 bits.

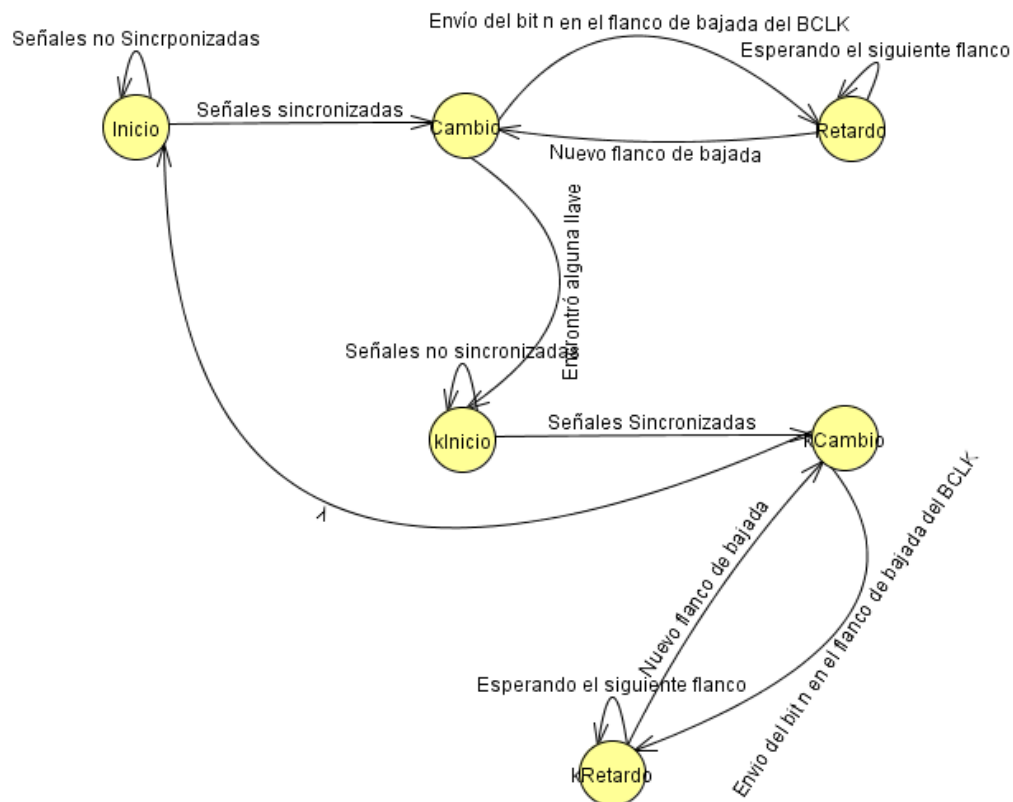


Figura 3. 9 Máquina de estados para reproducir sonido.

Cuando todas las muestras que componen a la señal hayan sido enviadas, se le dará un reset a los contadores, por lo tanto, el sonido de fondo estará reproduciéndose en un bucle durante toda la partida. Si en el estado *Cambio* recibe una señal de que una de las llaves fue capturada, cambiará al estado *Inicio*, donde reiniciará el proceso descrito anteriormente, con la diferencia de que el sonido solo va a reproducirse una vez. Una vez finalizada este audio, se volverá a la máquina de estados principal reproduciendo la música de fondo de forma continua.

3.6 CONTROL DE LAS IMÁGENES

Durante el proceso de despliegue de imágenes, se tienen contemplados varios módulos que están involucrados en mostrar una imagen (con sus valores RGB respectivos) sobre un puerto VGA.

En una pantalla no pueden convivir 2 imágenes sobre una misma área delimitada, esto restringe cómo serán desplegadas las imágenes sobre la pantalla. Se tienen considerados 2 tipos de imágenes, como se muestra la figura 3.10: las imágenes estáticas (son consideradas aquellas que se mantendrán en la misma posición en pantalla) y las imágenes dinámicas (la posición de estas imágenes será modificadas y se desplazarán sobre una zona delimitada en pantalla).

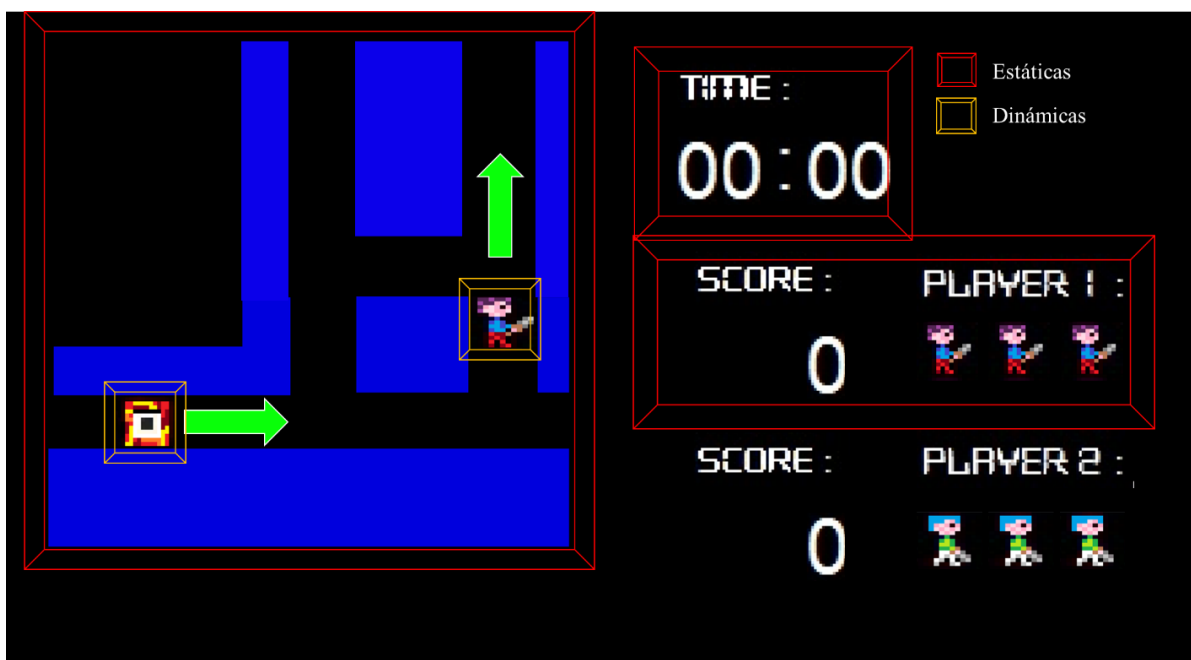
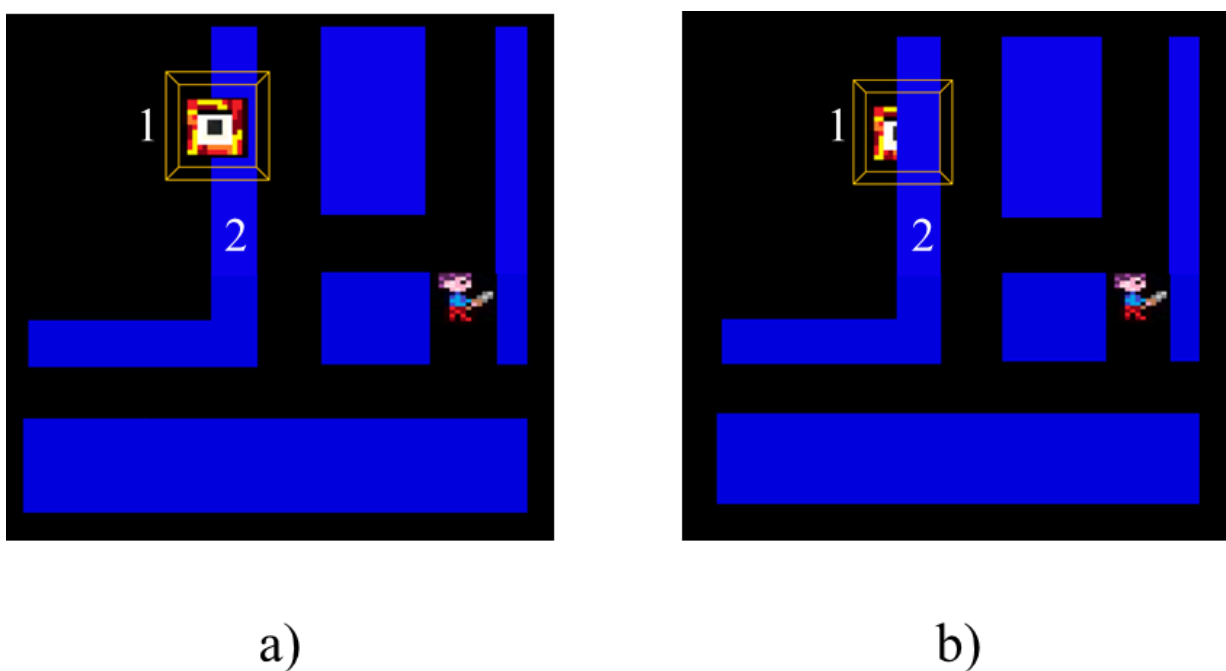


Figura 3. 10 Visualización en pantalla de los tipos de imágenes.

Para la aplicación del sistema de procesamiento de imágenes, cada imagen debe tener una jerarquía, donde una tiene mayor prioridad de ser mostrada en pantalla sobre otra. Como ejemplo: se puede dar el caso donde se despliega el laberinto y un enemigo. En la figura 3.11 se puede observar la figura a) donde la imagen 1 tiene un nivel más alto de jerarquía que la imagen 2. En cambio, la figura b) muestra que la imagen 2 tiene una mayor jerarquía sobre la imagen 1.



- a) Imagen 1 con mayor jerarquía respecto a la imagen 2.
b) Imagen 2 con mayor jerarquía respecto a la imagen 1.

Figura 3. 11 Jerarquía de imágenes.

Otra característica importante para definir el diseño de la jerarquía del despliegue de las imágenes está definida por la importancia de las imágenes al momento de desplegarse en pantalla. Esto puede ser observado en el ejemplo anterior con la imagen del laberinto, además puede observarse en la parte de la presentación y conclusión de la aplicación. En la figura 3.12 pueden observarse 2 imágenes; la imagen a) muestra que la presentación tiene mayor privilegio de ser mostrada en pantalla que todas las demás imágenes (el diseño define que la presentación debe ser la primera imagen que debe ver el usuario). En cambio, la imagen b) es la siguiente imagen que continúa en el nivel de jerarquía (esto se debe a que, si finaliza la aplicación, esta debe de estar por encima de las demás imágenes).

La tabla 2 muestra la jerarquía que tiene cada imagen sobre pantalla al momento de ser desplegada, este diseño es definido por las características descritas anteriormente, donde la columna 3 indica el nivel de jerarquía que la imagen tiene sobre las demás.



a)



b)

- a) Presentación con mayor jerarquía en la aplicación.
 b) Imagen de juego terminado con mayor jerarquía dentro de la pantalla de juego.

Figura 3. 12 Privilegios de imágenes de presentación y de finalización.

<i>Imagen</i>	<i>Figura</i>	<i>Nivel de jerarquía</i>	<i>Observaciones</i>
<i>Presentacion1</i>		1	Estas imágenes tienen el mayor privilegio. Son desplegadas y desaparecen de la pantalla una vez que el usuario inicie el proceso principal de la aplicación.
<i>Presentacion2</i>		2	
<i>Gameover</i>		3	Esta imagen debe mostrarse al usuario

			una vez que finalice el proceso.
<i>Articale</i>		4	Estas imágenes pertenecen a cada usuario para poder ser utilizados durante el proceso de la aplicación.
<i>Roy</i>		5	
<i>Llaves</i>		6	Son el objeto de búsqueda para el usuario.
<i>Enemy1</i>		7	Estarán desplazándose sobre el área delimitada de la siguiente imagen.
<i>Enemy3</i>		8	
<i>Enemy2</i>		9	
<i>Laberinto</i>		10	Es el área delimitada de la aplicación, donde el usuario puede desplazarse.




<i>Score1 y 2</i>		11	Son indicadores informativos para el usuario sobre su progreso.
<i>Timer</i>		12	
<i>Lives12</i>		13	

Tabla 2 Jerarquía de las imágenes.

Una vez definida la jerarquía que tiene cada imagen, el proceso de despliegue de imágenes llegará a un controlador que administrará el orden para que el dispositivo pueda desplegar las imágenes de forma correcta. En el siguiente capítulo se hablará sobre la implementación en la tarjeta con FPGA abordando conceptos más técnicos pero enfocados al diseño propuesto.

CAPÍTULO 4 IMPLEMENTACIÓN EN LA TARJETA DE DESARROLLO

Durante el capítulo anterior, se abordó el diseño de la aplicación, algunos conceptos generales y sobre todo alguna funcionalidad. Por lo que en este capítulo se hablará sobre la implementación de dicha aplicación sobre una tarjeta de desarrollo con FPGA.

Para la parte de la implementación, se describirán algunas características técnicas que contiene la tarjeta BASYS 3, el diseño conceptual implementado en dicha tarjeta y algunos módulos del funcionamiento.

4.1 TARJETA DE DESARROLLO BASYS 3.

Durante la fase de investigación se encontraron diversos dispositivos para poder implementar sistemas digitales, algunos de ellos tienen características que se requieren para implementar la aplicación del sistema de procesamiento de imágenes.

Como se muestra en la siguiente tabla, existe una relación FPGA – precio en el mercado actualmente¹⁸:

<i>Tarjeta</i>	FPGA	VGA	Memoria	Precio¹⁹
<i>BASYS 2</i>	Xilinx® ²⁰ Spartan – 3E FPGA	8-bit de color	72 Kbits block RAM	\$149.00 USD
<i>NEXYS A7</i>	Artix-7 FPGA	12-bit de color	4,860 Kbits block RAM	\$265.00 USD
<i>NEXYS 4</i>	Artix-7 FPGA	12-bit de color	4,860 Kbits block RAM	\$320.00 USD
<i>BASYS 3</i>	Artix-7 FPGA	12-bit de color	1,800 Kbits block RAM	\$149.00 USD

Tabla 3 Tabla comparativa de dispositivos con FPGA en el mercado.

¹⁸ Toda la información de estos dispositivos, se puede consultar en la página del proveedor Digilent® <https://store.digilentinc.com/fpga-development-boards-kits-from-digilent/>.

¹⁹ Los precios son en moneda *Dólar americano* (USD), sin gastos de envío incluidos.

²⁰ Empresa estadounidense. Su principal función es el desarrollar y proveer de dispositivos lógicos programables. Además, es el inventor del FPGA y de sistemas embebidos en un solo circuito.

Comparando características-precio se optó por implementarse el sistema sobre una tarjeta BASYS 3 con un FPGA de tipo Artix-7.

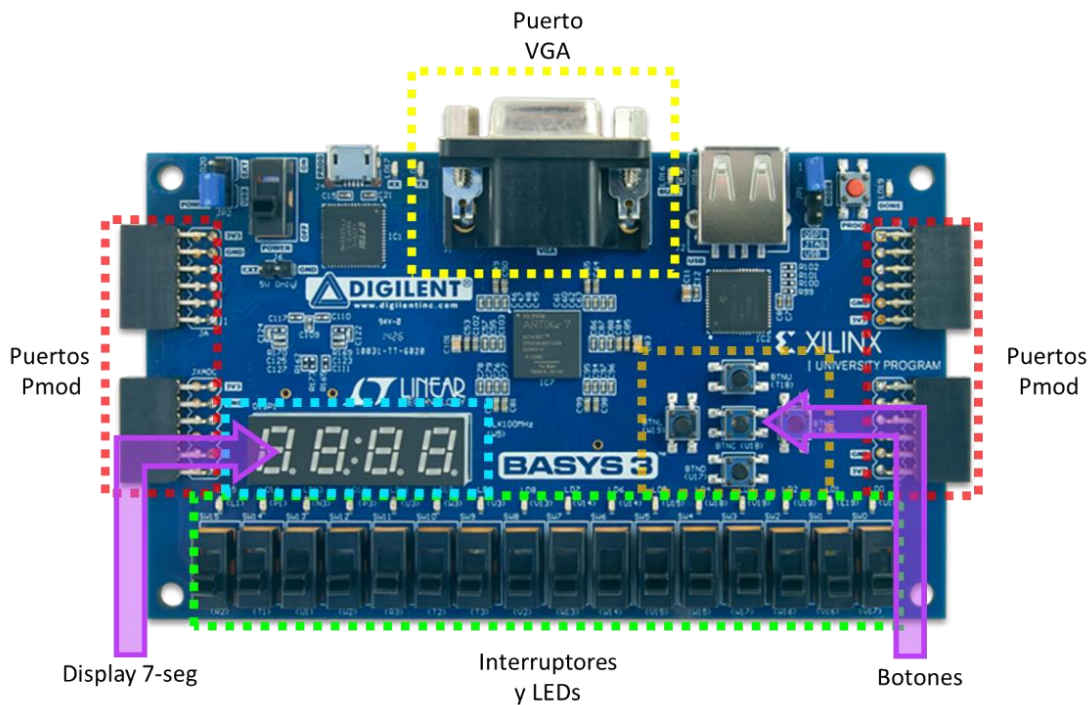


Figura 4. 1 Tarjeta *BASYS 3* indicando los puertos utilizados.[22]

La BASYS 3, además de contener el puerto VGA y los bloques de memoria, cuenta con un amplio número de periféricos, tanto de entrada como de salida. Cabe mencionar las características que contiene:

Chip Artix-7

- 33,280 celdas lógicas en 5200 porciones.
- 1,800 Kbits de bloques de RAM.
- Reloj interno que puede llegar hasta 450 MHz.

Puertos y periféricos

- 16 interruptores.
- Display de 4 dígitos de 7 segmentos.

- Puerto de salida VGA con 12-bit de color.
- Puerto USB para programación y comunicación de datos.
- 16 LEDs.
- 3 puertos para expansión Pmod.
- Puerto para conexión USB-UART.
- 5 botones.
- Entrada Pmod para convertidor analógico digital (XADC).

4.2 IMPLEMENTACIÓN Y DESARROLLO.

En el capítulo anterior, se discutió acerca del diseño del sistema completo y la funcionalidad de cada parte o módulo desarrollado. Durante este capítulo se describirá la implementación sobre las herramientas de desarrollo, la interacción y manejo de las señales utilizadas, dando los detalles de los resultados obtenidos.

Hablando acerca de las herramientas de desarrollo para el diseño e implementación de este sistema en la tarjeta BASYS 3 fueron:

- Lenguaje de Descripción de hardware: VHDL.
- VIVADO DESIGN SUITE (versión 2018.2)²¹. Entorno de desarrollo de licencia libre para FPGA desarrollado para dispositivos Xilinx con el cual se configura y describe el desarrollo en la tarjeta.

4.2.1 MÓDULO DIVISOR DE FRECUENCIA

El divisor de frecuencia es un módulo encargado de ofrecer a los otros módulos señales de sincronización, dependiendo del requerimiento de la funcionalidad será desplegada la frecuencia apropiada.

Con base en las características que ofrece la tarjeta BASYS 3, la señal de sincronización principal del sistema es de 100 MHz, esta señal es declarada en el sistema como *Master_clk*,

²¹ Se puede consultar toda la información acerca de la herramienta en: <https://www.xilinx.com/products/design-tools/vivado.html>

con ella se pueden desplegar diferentes frecuencias. Para generar dichas frecuencias, se basan en 2 puntos importantes: conocer la frecuencia base deseada y realizar la conversión de la frecuencia base a la frecuencia deseada.

Algunos módulos requieren una frecuencia diferente a la señal de sincronización que proporciona el dispositivo, de tal sentido, se deben realizar cálculos para reducir la frecuencia base a una frecuencia deseada como se muestra en el Pseudocódigo 1.

El módulo interactúa con otros sistemas que requieren de su participación como son: el módulo de las señales síncronas *VGA640x480*, el módulo específico para delimitar el espacio permitido para el usuario *Laberinto*, los sistemas encargados de interactuar y desplegar imágenes con el usuario, los repositorios de datos para resguardar las imágenes y los audios convertidos, los sistemas que manejan el puntaje, el temporizador del sistema, los módulos controladores del audio y el módulo orquestador del sistema *Gamecore*.

```
Algoritmo Reloj_25MHz  
  contador <- 0  
  señal<-0  
  mientras (master_reset = 0) hacer  
    Leer master_reset  
    Si master_reset = 1 Entonces  
      contador <- 0  
      señal<-0  
    SiNo  
      Leer master_reloj  
      Si master_reloj = 1 Entonces  
        Si contador >= 1 Entonces  
          señal <- 1  
          contador <- 0  
        SiNo  
          contador <- contador + 1  
          señal <- 0  
        Fin Si  
      FinSi  
    Fin Si  
    Clk25M <- señal  
    Escribir Clk25M  
  FinMientras  
FinAlgoritmo
```

Pseudocódigo 1 - Divisor de frecuencia de 100 MHz a 25 MHz que requiere el módulo *VGA640x480*.

En la figura 4.2 se observa la importancia que tiene ese sistema, ya que los diseños de los módulos anteriores no podrían funcionar secuencialmente (por consecuencia, no se podría utilizar el diseño con máquinas de estados), que son necesarios para poder interactuar todos de forma concurrente.

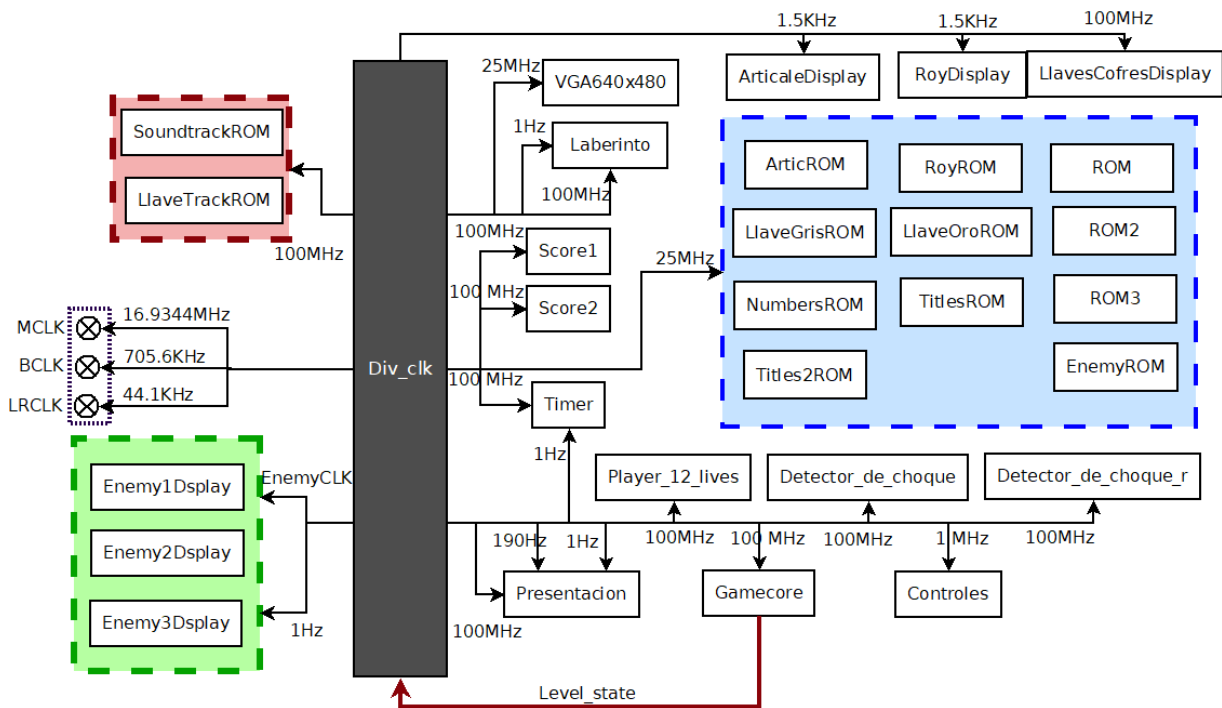


Figura 4. 2 Interacción con el módulo de divisor de frecuencia.

La interacción que se tiene con el módulo de orquestación es enviarle el estado del nivel, con el cual, para la señal de salida conocida como *EnemyCLK*, cambiará del nivel del proceso que se encuentre, esto se puede mostrar en la tabla 4, el cual indica la salida de frecuencia dependiendo de la señal de estado de nivel.

<i>Nivel</i>	Frecuencia
1	38.46 Hz
2	47.61 Hz
3	58.82 Hz
4	83.33 Hz
5	90.90 Hz

Tabla 4 Frecuencias de movimiento de los enemigos.

4.2.2 MÓDULO VGA 640X480

Observando el estándar VGA[17], se desarrolló un módulo que se encargará de generar todas las señales síncronas de video para la resolución de 640x480, donde se requerirá de una señal de reloj de 25 MHz (obtenido del módulo divisor de frecuencia), además de implementar valores definidos por el estándar como lo son:

Valores para la señal de sincronía horizontal:

- HBP (Horizontal Back Porch) = SP (Synchrony Pulse) + BP (Back Porch) = $128 + 16$ pixeles = 144 pixeles.
- HFP (Horizontal Front Porch) = HBP + HV (Horizontal Video) = $144 + 640$ pixeles = 784 pixeles.
- Total de Pixeles = $SP + BP + HV + FP$ (Front Porch) = $128 + 16 + 640 + 16 = 800$ pixeles.

Valores para la señal de sincronía Vertical:

- VBP (Vertical Back Porch) = SP (Synchrony Pulse) + BP (Back Porch) = $2 + 29$ líneas horizontales = 31 líneas horizontales.
- VFP (Vertical Front Porch) = VBP + VV (Vertical Video) = $31 + 480$ líneas horizontales = 511 líneas horizontales.
- Total de líneas = $SP + BP + VV + FP$ (Front Porch) = $2 + 29 + 480 + 10 = 521$ líneas horizontales.

El sistema replica las señales de video con base a los valores obtenidos anteriormente. Para generar dichas señales, la frecuencia base para las señales síncronas horizontales es de 25 MHz y la frecuencia base para las señales síncronas verticales será la que retorne la señal síncrona horizontal al recorrer todos los pixeles horizontales, esto se puede ver en Pseudocódigos 2 y 3.

Algoritmo Senales_sincronas_Horizontales

```
contador_horizontal<-0
pixeles_horizontal<-799
senal_sincrona<- 128
mientras(master_reset = 0) hacer
    Leer master_reset
    Si master_reset = 1 Entonces
        contador <- 0
        senal_vertical<-0
    SiNo
        Leer reloj_25M
        Si reloj_25M = 1 Entonces
            Si contador_horizontal =
                pixeles_horizontal Entonces
                    senal_vertical <- 1
                    contador_horizontal <- 0
            SiNo
                contador_horizontal <-
                    contador_horizontal + 1
                senal_vertical <- 0
            Fin Si
        FinSi
    Fin Si
    Si contador_horizontal >= senal_sincrona Entonces
        Senal_horizontal_sincrona<- 1
    SiNo
        Senal_horizontal_sincrona<- 0
    FinSi
    Escribir Senal_horizontal_sincrona
FinMientras
FinAlgoritmo
```

Pseudocódigo 2 - Señal síncrona horizontal.**Algoritmo Senales_sincronas_Verticales**

```
contador_vertical<-0
pixeles_vertical<-521
senal_sincrona<- 2
mientras(master_reset = 0) hacer
    Leer master_reset
    Si master_reset = 1 Entonces
        contador <- 0
        senal_vertical<-0
    SiNo
        Leer reloj_25M
        Leer Senal_horizontal_sincrona
        Si (Senal_horizontal_sincrona = 1) Y
            (reloj_25M = 1) Entonces
                Si contador_vertical =
                    pixeles_vertical Entonces
                        contador_vertical <- 0
```

```

                SiNo
                    contador_vertical <-
                        contador_vertical + 1
                Fin Si
            FinSi
        Fin Si
    Si contador_vertical >= senal_sincrona Entonces
        Senal_vertical_sincrona<- 1
    SiNo
        Senal_vertical_sincrona<- 0
    FinSi
    Escribir Senal_vertical_sincrona
FinMientras

```

FinAlgoritmo

Pseudocódigo 3 - Señal síncrona vertical.

Para indicar que las señales síncronas se encuentran en la zona visible en pantalla, se utiliza una señal de habilitación, donde, estarán las condiciones delimitadas por el estándar, esto se puede observar en el Pseudocódigo 4.

```

Algoritmo indicador_de_zona_visible
    señal_video <- 0
    hbp <- 144
    hfp <- 784
    hpixels <- 800
    vbp <- 31
    vfp <- 511
    vlínes <- 521
    Mientras (master_reset=0) hacer
        Leer master_reset
        Leer senal_vertical_sincrona
        Leer senal_horizontal_sincrona

        Si (((senal_horizontal_sincrona < hfp) Y
            (senal_horizontal_sincrona >= hbp)) Y
            ((senal_vertical_sincrona < vfp) Y
            (senal_vertical_sincrona >= vbp))) Entonces

            señal_video <- 1

        SiNo
            señal_video <- 0

        FinSi
        Escribir Senal_horizontal_sincrona
    FinMientras
FinAlgoritmo

```

Pseudocódigo 4 - Señal de habilitación en pantalla.

4.2.3 DESPLIEGUE DE FIGURAS EN PANTALLA (MÓDULO LABERINTO)

El módulo *Laberinto* tiene como función delimitar en pantalla el espacio disponible para desplazarse, en donde cada jugador deberá cumplir su objetivo de conseguir todas las llaves correspondientes de cada nivel, como se muestra en la imagen 4.3. El sistema se encuentra integrado con otros módulos, como lo son: el sistema orquestador *Gamecore*, el sistema encargado de las señales síncronas *VGA640x480* y el divisor de frecuencia *Div_clk*.

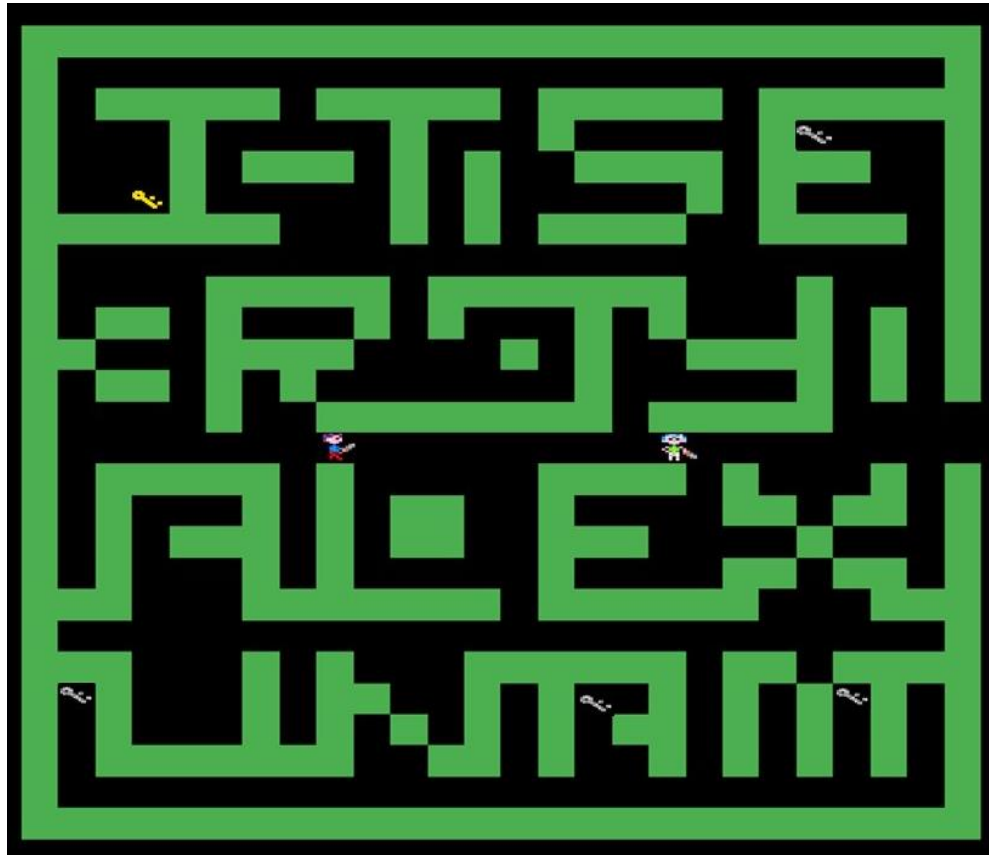


Figura 4. 3 Despliegue en pantalla del laberinto.

Una vez mencionados los módulos involucrados, el funcionamiento del *Laberinto* se debe contemplar como si fuese una imagen de un tamaño definido con los siguientes datos: el área que se desplegará en pantalla, la posición donde iniciará el despliegue de datos y el área considerada como las paredes que bloquearán el paso al jugador durante la obtención de las llaves a lo largo de la travesía.

El área definida que se desplegará en pantalla es de 420x480 píxeles. En este espacio se muestran los caminos donde se podrá desplazar el jugador. Para hacer esto posible, el módulo

trabaja al mismo tiempo con el divisor de frecuencia y con el módulo VGA640x480 como se muestra en la imagen 4.4.

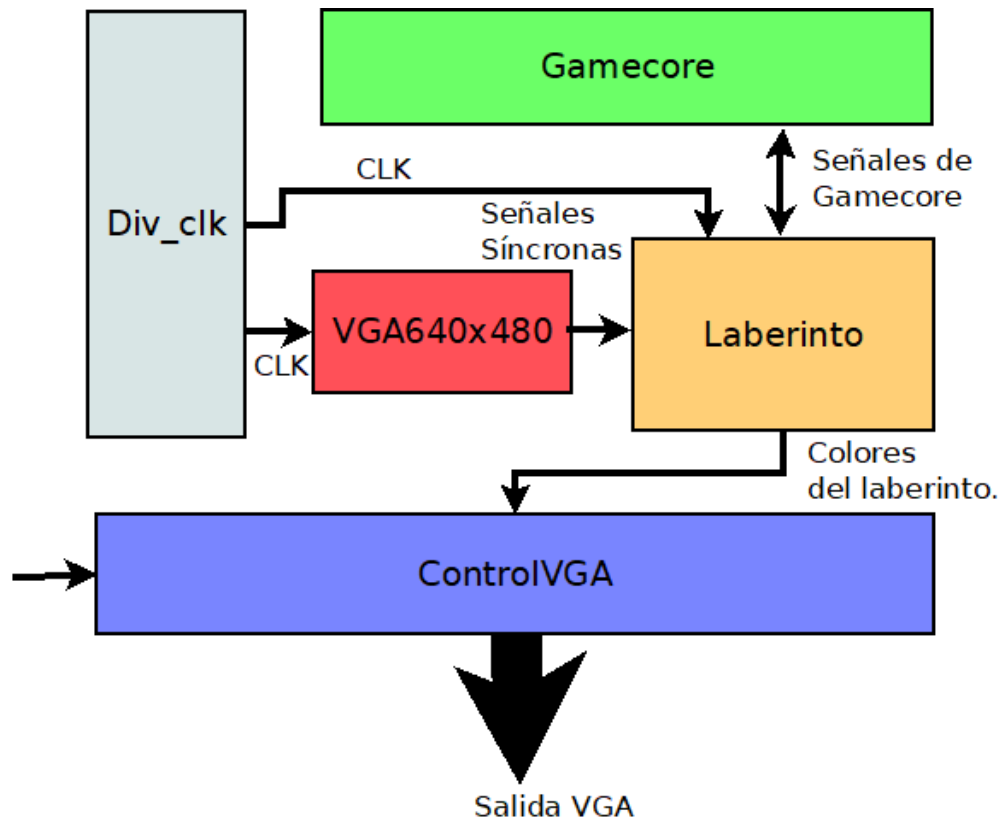


Figura 4. 4 Interacción entre el módulo Laberinto.

Previamente, se concretan los espacios donde se trazarán las paredes de las posiciones recibidas por el módulo VGA640x480, trabajando mutuamente recibiendo las señales síncronas, el módulo comparará en qué píxel se encuentra actualmente y así poder desplegar la imagen del laberinto correctamente. Para describir el funcionamiento se ubica la imagen 4.5, en la cual se muestra el laberinto de color azul y una pequeña parte del laberinto de color amarillo con 4 coordenadas que ubicarán el espacio que se desplegará en pantalla.

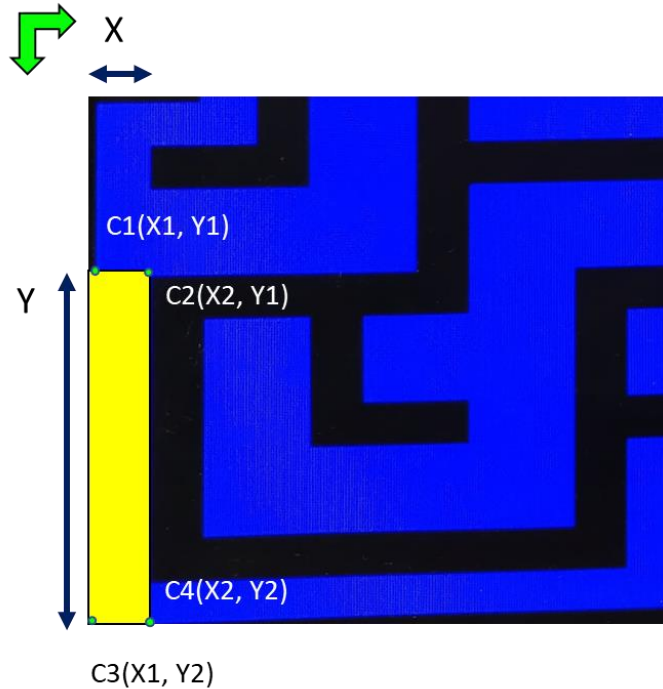


Figura 4. 5 Despliegue del laberinto en pantalla.

Una vez descrita la imagen anterior, debe ser desplegada en pantalla la sección en amarillo, donde se delimita las coordenadas de los extremos del área conformados por C1, C2, C3 y C4. El proceso comparará si las señales sincronas se encuentran desplegándose en esa zona, es decir, si la señal sincrona horizontal (HC) se encuentra entre X1 y X2, y la señal sincrona vertical (VC) que se encuentra entre Y1 y Y2. Si se detectan las señales sincronas dentro de las coordenadas, se despliega el laberinto de un color distinto. En el caso contrario se llenará con un campo de color negro, indicando que este es el espacio donde los jugadores podrán desplazarse.

Teniendo ubicadas todas las coordenadas requeridas para formar el laberinto, será desplegada con un color definido por el sistema. Este color será seleccionado por el estado donde se encuentre el nivel. Para esto, se definió dentro del módulo una máquina de estados que se encargue de este proceso, como se muestra en la imagen 4.6. El proceso es el siguiente: mientras la señal de inicio este en nivel bajo ('0') permanecerá en el estado *Inicio*. Cuando se detecta el cambio de nivel de bajo a alto ('1'), cambiará al estado *L1* (que viene siendo el nivel uno del juego). En este estado *L1*, se envía en la salida sazul el valor "1101", para la salida sverde el valor "0011" y la salida srojo el valor "0001". Para cambiar al siguiente nivel, el *gamecore*

deberá enviar la señal correspondiente de *level_state*, donde el estado corresponde al valor de dicha señal recibida. En la siguiente tabla 5 se pueden observar los valores enviados a la salida VGA del módulo con relación al nivel actual.

Nivel	Color	Estado_nivel	sazul	Srojo	Sverde
1	Azul	001	1101	0001	0011
2	Verde	010	0010	0001	1010
3	Amarillo	011	0011	1001	1100
4	Anaranjado	100	0001	1011	0010
5	Rojo	101	0000	1011	0010
Fin del juego	Gris oscuro	111	0010	0010	0010

Tabla 5 Colores a 12 bits para cada nivel del juego.

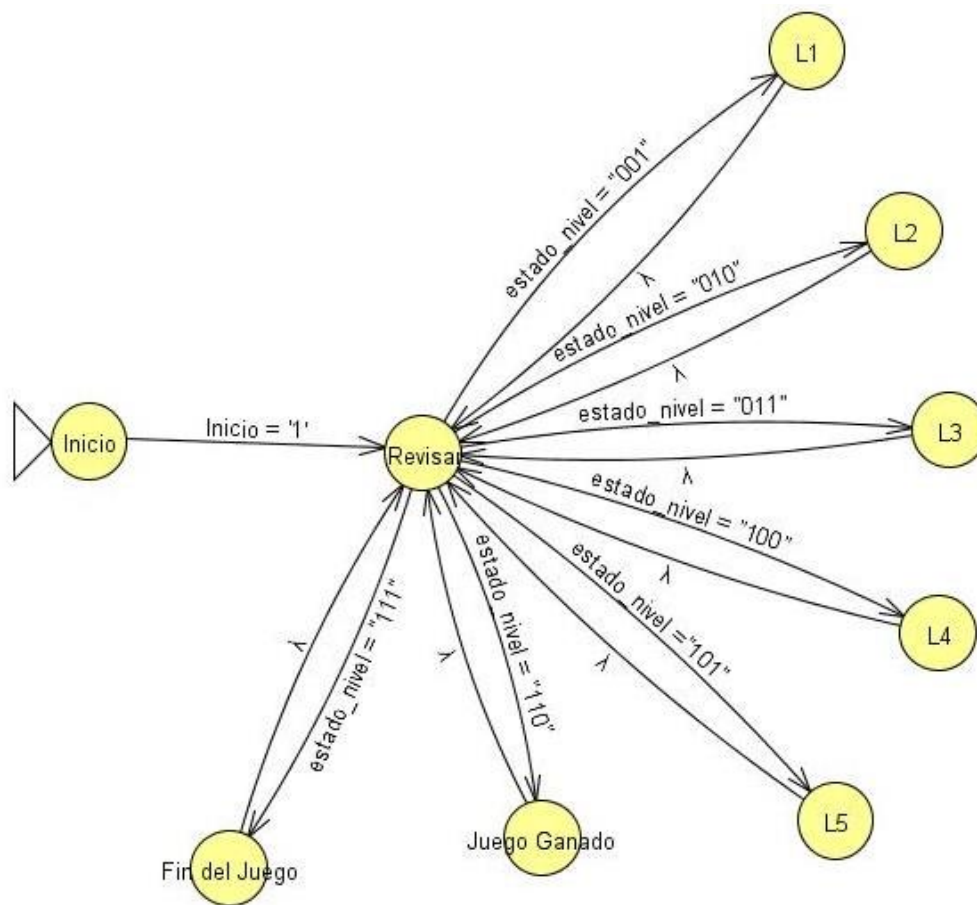


Figura 4. 6 Máquina de estados del módulo Laberinto.

4.2.4 EFECTO DEL MOVIMIENTO DE IMÁGENES (MÓDULO DE LOS PERSONAJES)

Los módulos de los personajes son sistemas con el que el jugador puede manipular la posición de la imagen del personaje correspondiente.

Los módulos de los personajes están integrados con otros sistemas como se ve en la imagen 4.7: con el sistema orquestador *Gamecore*, el módulo transductor de los mandos de entrada *Controles*, el generador de señales síncronas *VGA640x480*, con los módulos de memoria *ArticaleDisplay* o *RoyDisplay*, con el controlador del puerto VGA *ControlVGA*, con el divisor de frecuencia *Div_clk* y el módulo encargado de detectar los impactos con los enemigos *detector_de_choque*.

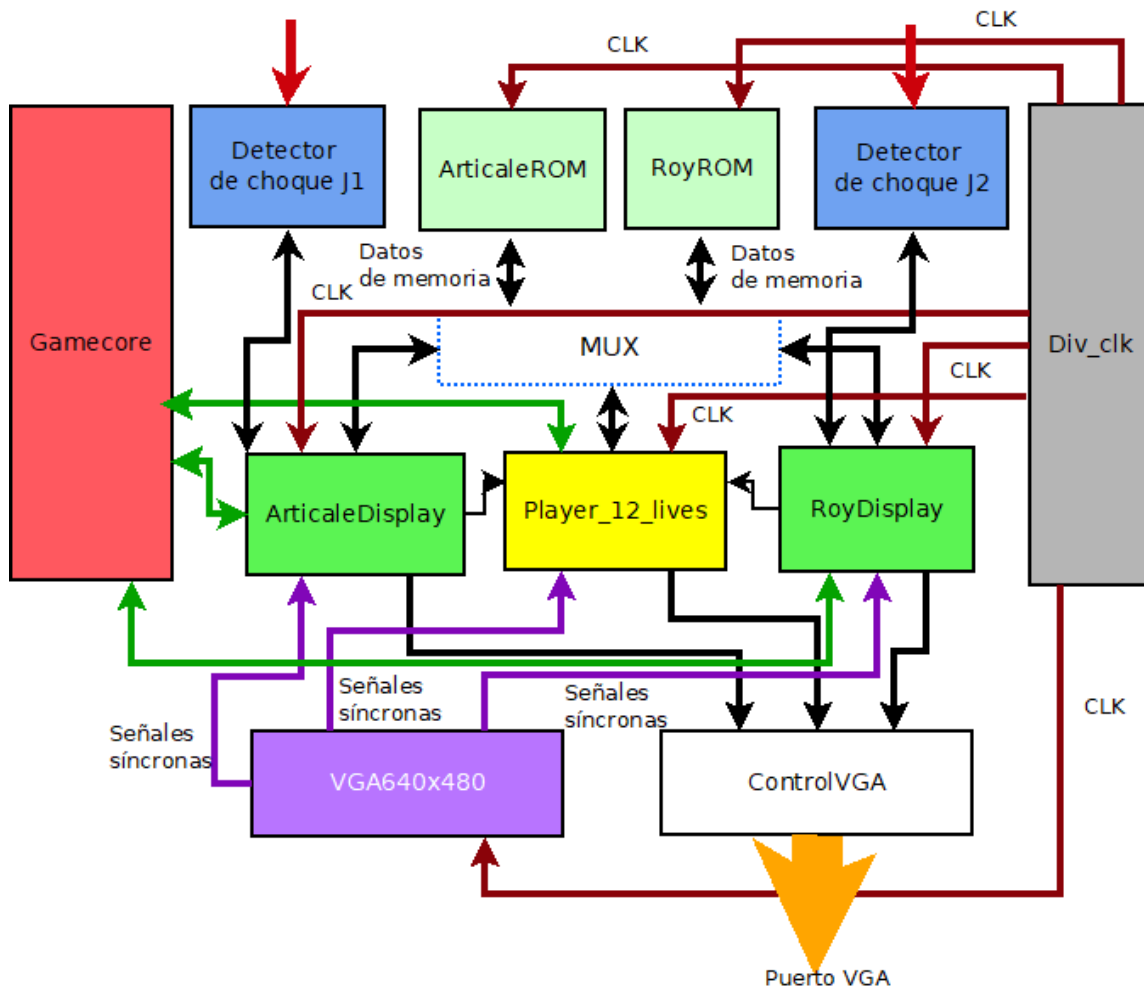


Figura 4. 7 Máquina de estados del personaje principal.

El jugador podrá manipular a uno de los personajes en pantalla, en la figura 4.8 se muestra la máquina de estados que permite el control del personaje por medio de un mando de control. El comportamiento del personaje del jugador 1 y el jugador 2 es el mismo, así que solo se explicará una máquina de estados.

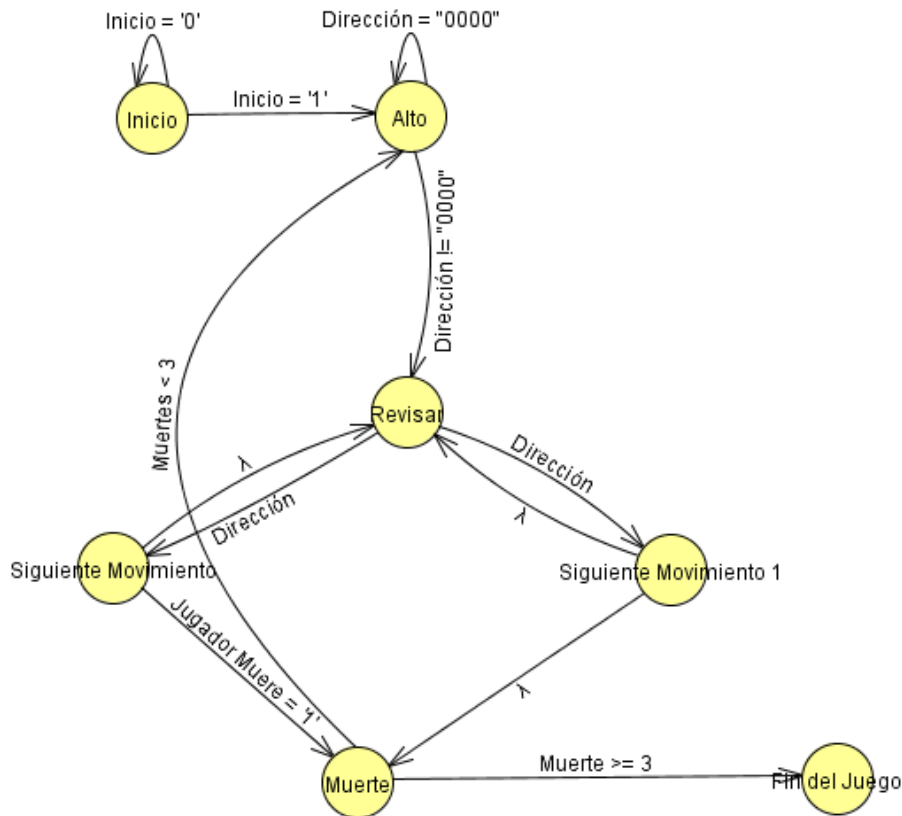


Figura 4. 8 Máquina de estados del personaje principal.

La máquina de la figura anterior comienza en el estado *Inicio*, donde permanecerá en este estado hasta que alguno de los jugadores haya presionado el botón de inicio. Una vez hecho esto, pasará al estado de *Alto*, donde esperará una señal de la dirección. Mientras no se reciba alguna señal, permanecerá en este estado, de lo contrario, cambiará al estado *Revisar*, donde si es una dirección inválida regresará al estado de *Alto*. El hecho de que una señal sea válida o no, dependerá de la posición del jugador en ese momento.

En el estado *Revisar* siempre va a verificarse que la señal a la que se pretende avanzar sea una dirección válida, de ser así se dirigirá a *Siguiente Movimiento* avanzando a la posición deseada y se mostrará en la pantalla la imagen correspondiente para después regresar a *Revisar*. Si la dirección deseada es la misma que en el instante anterior se dirigirá a *Siguiente Movimiento 1*, donde de igual forma se moverá hacia el punto indicado, sin embargo, se desplegará otra imagen con la misma dirección, pero con diferente postura. Se puede observar en la figura 4.9 algunas de las imágenes utilizadas para dar un efecto de movimiento similar al de los folioscopios (o flipbooks en inglés).



Figura 4. 9 Máquina de estados del personaje principal.

Durante todo el juego, se estará alternando entre estos tres estados, excepto cuando el jugador sea impactado por uno de los enemigos, de ser así se recibe una señal indicando al jugador que “murió”, por lo tanto, ya sea de *Siguiente Movimiento* o *Siguiente Movimiento 1* se dirigirá al estado de *Muerte*. En este estado, uno a uno se irá contabilizando todas las “muertes” del jugador hasta llegar a un total de 3 “muertes”. De ser así, regresará al estado *Alto* e inmediatamente cambiará a *Fin del juego*, indicando el fin de la partida.

4.2.5 MANDOS DE CONTROL

El sistema está integrado con: el módulo divisor de frecuencia *Div_clk* y los módulos de los personajes *ArticaleDisplay* y *RoyDisplay*. Como se ve en la figura 4.10, directamente existe un intercambio de datos con el mando como un transmisor y receptor de datos, el mando siempre está enviando 8 datos (los cuales son los botones del mando).

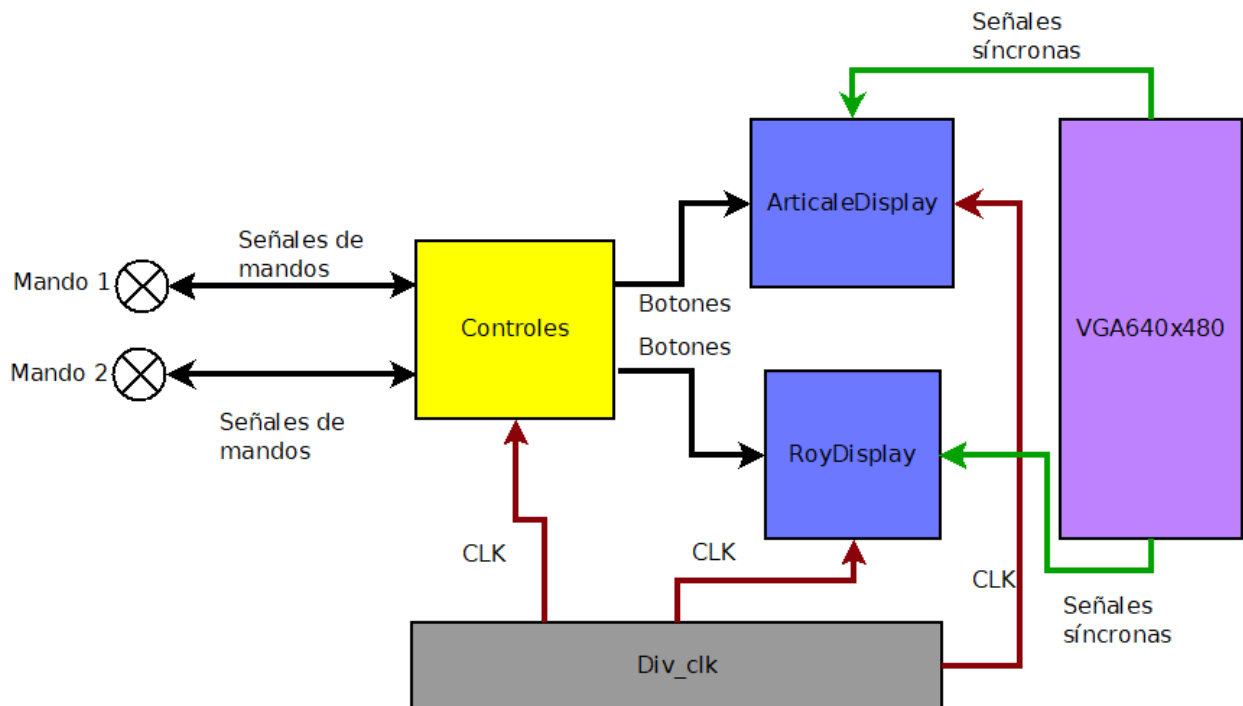


Figura 4. 10 Módulos involucrados con Controles.

La tarjeta cuenta con puertos de entrada interactivos (como lo son los interruptores y botones), también cuenta con pines de propósito general como se ve en la figura 4.11. Estos puertos están implementados con un estándar definido para dispositivos electrónicos que quisiesen añadir al sistema (como lo es el dispositivo amplificador de audio).

Basado en la figura 4.12, el diseño implementado para el control de los mandos es el siguiente: por los pines JA6 y JA12 son los pines de voltaje, JA5 y JA11 son los pines de GND, JA4 y JA10 para DATA, para los pines JA3 y JA9 es LATCH y los pines JA2 y JA8 es para PULSE.

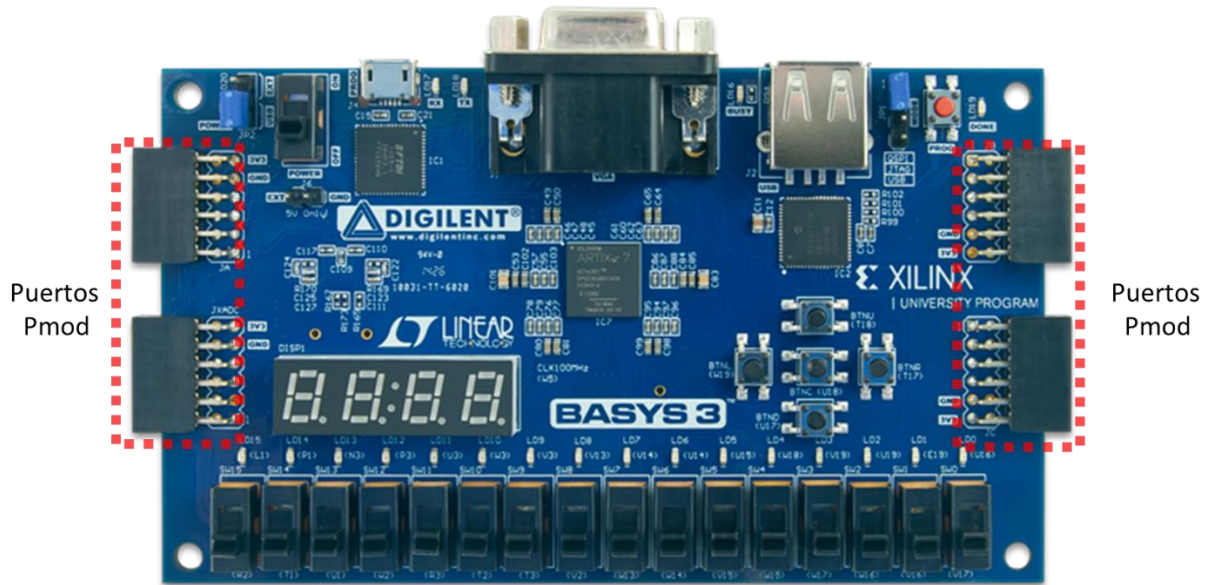


Figura 4. 11 Puertos disponibles de la tarjeta. [22]

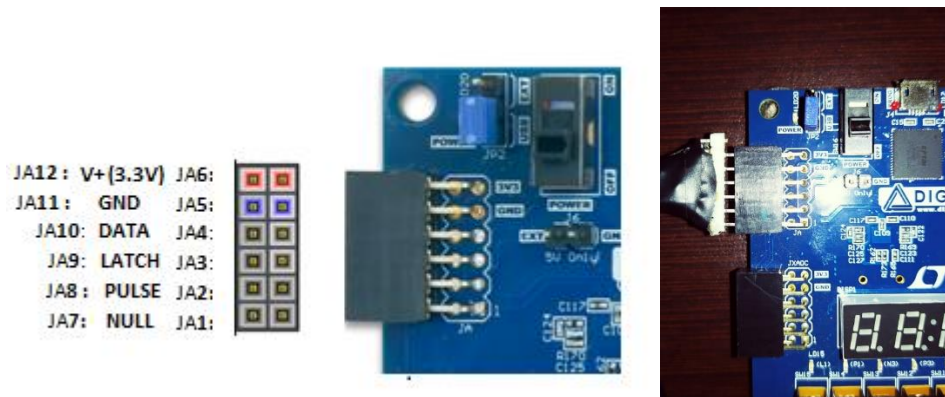


Figura 4. 12 Diseño de pines de entrada para los mandos.

4.2.6 DESPLIEGUE DE IMÁGENES

El proceso de despliegue de imágenes es llevado a cabo por 2 procesos:

- Definición de la dirección de memoria y obtención del dato resguardado en los bloques de memoria.
- Envío del dato correspondiente a la salida RGB del módulo de cada imagen.

Para definir el almacenamiento de las imágenes dentro la tarjeta, se debe conocer el tamaño de las imágenes y la cantidad de bits que ocupará cada celda de memoria. Con dichos datos se puede crear un bloque de memoria que cumpla con los requisitos.

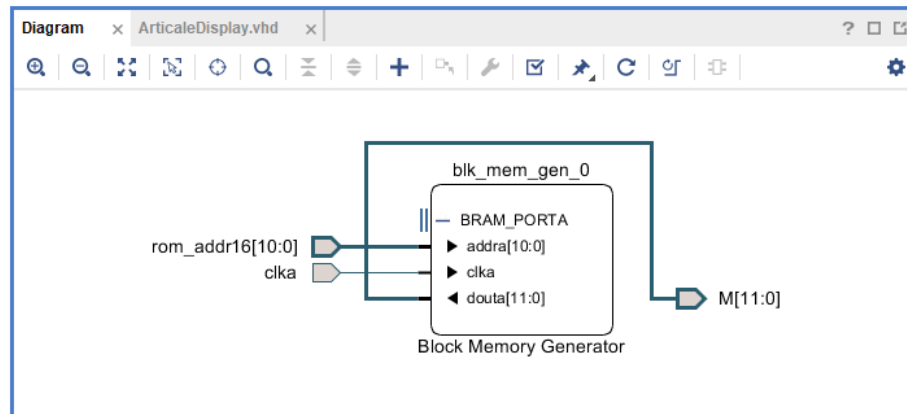


Figura 4. 13 Bloque de memoria para el resguardo de las imágenes.

Para el manejo de imágenes con el puerto VGA, la tarjeta trabaja con 12 bits destinados para el manejo del color (4 rojos, 4 verdes y 4 azules), por lo que generaría un rango de 4096 diferentes colores en pantalla, en comparación de la BASYS 2, que funciona con 8 bits de color (3 rojos, 3 verdes y 2 azules) tendría 256 tonalidades diferentes de color. Para definir el espacio del bloque, el tamaño de bits por cada celda es de 12 bits (la BASYS 3 puede manejar espacios por celda desde 1 bit hasta 4608 bits que pueden variar dependiendo de la cantidad de datos a resguardar, ofreciendo una diversidad de tamaños para almacenar).

La cantidad de datos por cada bloque dependerá directamente del tamaño de la imagen en pixeles (la BASYS 3 te permite guardar una cantidad máxima de 1048576 celdas, pero esta cantidad es directamente proporcional a la cantidad de bits que se guardan por celda), es decir, para una imagen de 15x15 tendrá una cantidad de 225 datos por almacenar, con un espacio de 12 bits por celda creada.

Una vez definidos los espacios de memoria creados en la tarjeta, se debe llenar dichos bloques que generó la tarjeta. Para poder realizar esto, son necesarias herramientas de apoyo de uso libre, que convierten imágenes del tipo de formato .JPG a un tipo de formato creado por la empresa Xilinx® llamado archivos de coeficientes (o en inglés coefficient files). Los archivos

tipo .COE²² definen: el tamaño de los datos en bits, el ancho y el alto de la imagen en pixeles. Dicho flujo puede observarse en la figura 4.14.

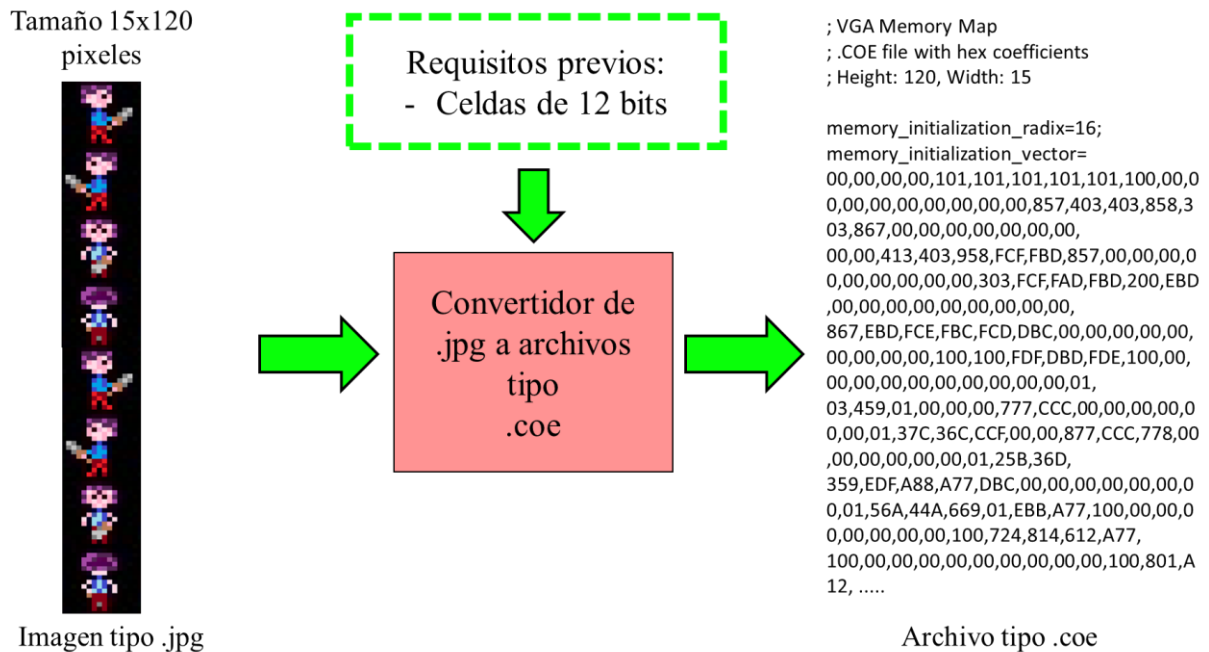


Figura 4. 14 Generación de los archivos tipo .coe con herramientas de uso libre para almacenamiento de imágenes.

Una vez almacenados los datos convertidos en los bloques de memoria, es generado un archivo de tipo VHD, con el cual se puede agregar al proyecto teniendo acceso a los datos que contienen los bloques de memoria, solamente añadiendo una señal de entrada de tipo vector que se encarga de dar las direcciones de memoria, otra señal de salida de tipo vector donde regresará (en 12 bits) los datos correspondientes a la dirección proporcionada. Y por último una señal de reloj que permita sincronizar el bloque de memoria.

Teniendo el dato de la imagen a desplegar, se debe saber en qué posición de la pantalla será desplegado cada uno de los pixeles. Por lo que las señales síncronas recibidas del módulo VGA640x480 serán interpretadas. Primero definiendo la posición deseada en pantalla para su despliegue, después realizar las conversiones de la posición deseada a la posición real en

²² Los archivos de tipo coeficientes (COE) fueron creados para el uso especializado de almacenamiento de datos dentro de los bloques de memoria incluidos en los dispositivos.

pantalla, y por último obtener la dirección en memoria con respecto a la posición en pantalla enviada por las señales síncronas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity ArticaleROM_wrapper is
  port (
    M : out STD_LOGIC_VECTOR ( 11 downto 0 );
    clka : in STD_LOGIC;
    rom_addr16 : in STD_LOGIC_VECTOR ( 10 downto 0 )
  );
end ArticaleROM_wrapper;
```

Figura 4. 15 Entidad generada para el bloque de memoria.

Para calcular la dirección deseada del bloque de memoria se realiza el siguiente proceso:

- Se obtiene la posición en X y Y en tiempo real con respecto a la posición deseada y el BP correspondiente a cada eje.
- Definir la señal de habilitación de despliegue de imagen, donde estará validando el tamaño de la imagen y las posiciones deseadas a desplegar.
- Convertir las posiciones en tiempo real (los datos del primer paso) al valor correspondiente a la dirección en memoria a buscar.

Algoritmo obtenerDirrecciónCorrespondienteEnMemoria

```
HBP <- 144
VBP <- 31
w <- 15
h <- 15
R1 <- 100
C1 <- 100
imageON <- 0
SalidaM <- 0
Mientras master_reset = 0 Hacer
  Leer hc
  leer vc
  Leer videON
  leer master_reset

  ypix <- vc - VBP - R1
  xpix <- hc - HBP - C1
```

```

rom_addr = (ypix * h) + xpix
escribir rom_addr
Si (((hc > C1 + HBP) y (hc <= C1 + HBP + w)) y
      ((vc >= R1 + VBP) y (vc < R1 + VBP + h))) Entonces
      imageON <- 1
SiNo
      imageOn <- 0
Fin Si
escribir imageON

Leer ImageM
Si ((imagen = 1) Y (videON = 1)) entonces
      SalidaM <- imagen
SiNo
      SalidaM <- 0
Fin Si
Escribir SalidaM
Fin Mientras
FinAlgoritmo

```

Pseudocódigo 5 - Algoritmo para obtener la dirección de memoria.

4.2.7 IMPLEMENTACIÓN DEL AUDIO

El despliegue del audio es dependiente del dispositivo electrónico, ya que la tarjeta no cuenta con un convertidor de digital-analógico incluido. Económicamente adquirir un dispositivo electrónico extra es más factible que utilizar una tarjeta que ya lo tenga incorporado.

El dispositivo Pmode AMP3[23] es un amplificador con convertidor analógico-digital que recibe los datos de audio por medio del protocolo I2S para transmisión de audio. En la figura 4.16 se puede observar donde se conecta el amplificador al puerto específico.

Una vez definido cómo se generan los bloques de memoria, se almacenan y se consumen, el audio trabaja casi de la misma forma. Solamente que el requisito del sistema es que se generan bloques de memoria de 16 bits por dato guardado en dicho bloque. Existen convertidores de archivos de audio WAV²³ a archivos de tipo coeficiente de uso libre.

Por medio del software libre *beepbox*[24] se generaron los sonidos típicos de 8 bits en formatos de tipo WAV, generando los archivos de coeficientes que se guardarán en los bloques

²³ WAV (Conocido en inglés Waveform Audio Format) es un formato de audio digital creado por Microsoft® y por IBM®, el cual su principal función es para almacenar sonidos con formato mono o estéreo.

de memoria. El sonido generado es de 1 segundo, con lo que al convertirlo se genera un total de 51328 celdas de 8 bits cada uno como se muestra en la figura 4.17.



Figura 4. 16 Definición del puerto de salida y conexión con del dispositivo.

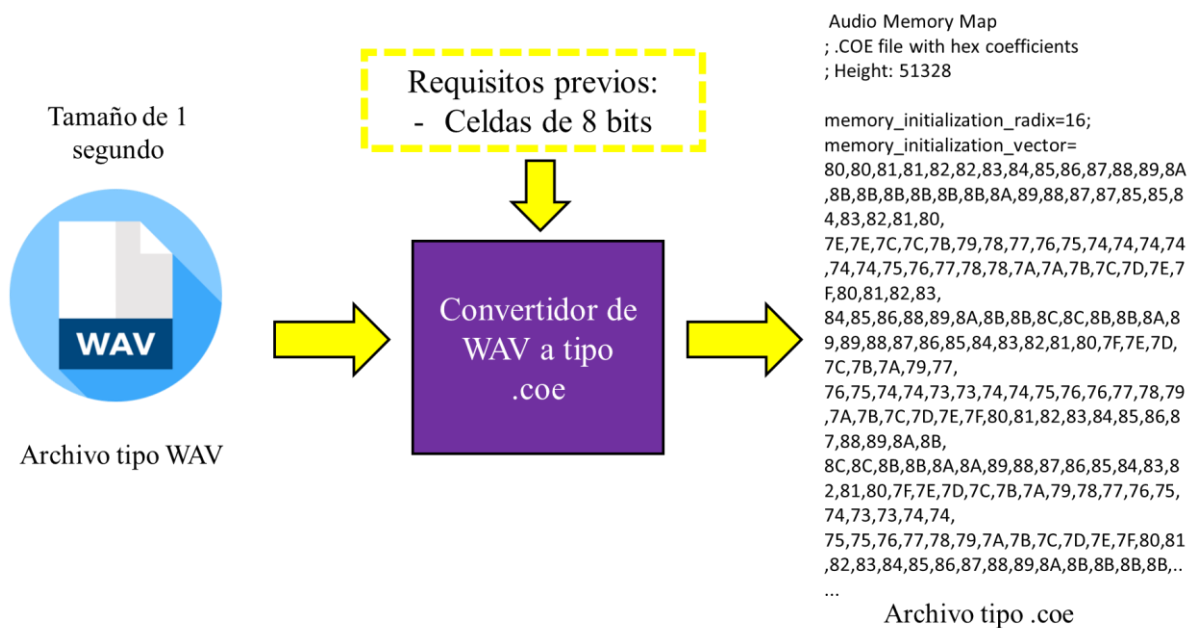
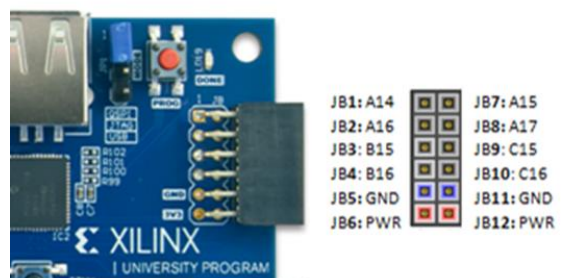


Figura 4. 17 Generación de los archivos tipo .coe.

Obtenidos los datos para desplegar, se debe seguir el proceso definido por el protocolo I2S y el diseño realizado para el protocolo I2S justificado a la izquierda, enviándose por los puertos correspondientes del dispositivo electrónico utilizado para el diseño. La salida del audio puede observarse en la figura 4.18 mostrando los pines para su conexión con el dispositivo PMODE AMP3.



BASYS 3

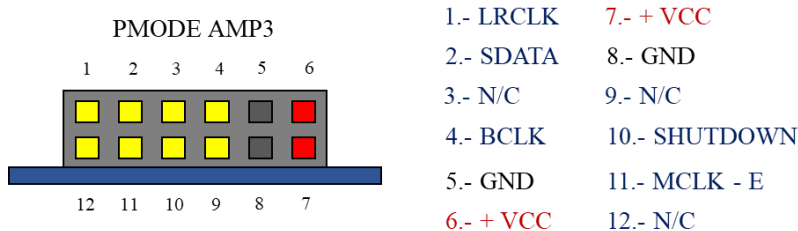


Figura 4. 18 Pines definidos por la tarjeta y por el dispositivo PMODE AMP3.

4.2.8 INTERACCIÓN CON EL PUERTO VGA (MÓDULO DE CONTROL VGA)

En todo este proceso de despliegue de imágenes, se tiene contemplado que varios módulos desplegarán una imagen o los valores RGB en un solo puerto VGA. Por lo que se diseñó un módulo para detectar la imagen que desplegará sus valores RGB del píxel donde se llenará en pantalla.

Analizando la situación previa, se maneja el principio de un multiplexor, solamente que cambia la esencia de la selección del despliegue del valor que se envía. En la imagen 4.19 se puede observar la interacción que existe entre los módulos que despliegan imágenes en pantalla y con el módulo *ControlVGA*. Las funciones que realizará este módulo son: enviar sus señales RGB de 12 bits del valor correspondiente del píxel y su señal de estado de despliegue.

Describiendo los datos requeridos que maneja los sistemas, la señal de estado de despliegue es una señal de tipo informativa que envía cada módulo, indicando que es momento de desplegarse en pantalla. Para esto, cada uno de los módulos está conectado con el sistema de señales síncronas *VGA640x480*, el cual le indica el valor del píxel y del renglón que se está desplegando en tiempo real. Cada módulo de imagen tiene la responsabilidad de revisar si los píxeles y renglones que se tienen en tiempo real corresponden en la posición que debe desplegarse. Al ser exitoso esta condición, se habilitará la señal de estado de despliegue, donde esta llegará al módulo *ControlVGA* y desplegará el valor recibido del píxel correspondiente a la posición de la señal síncrona.

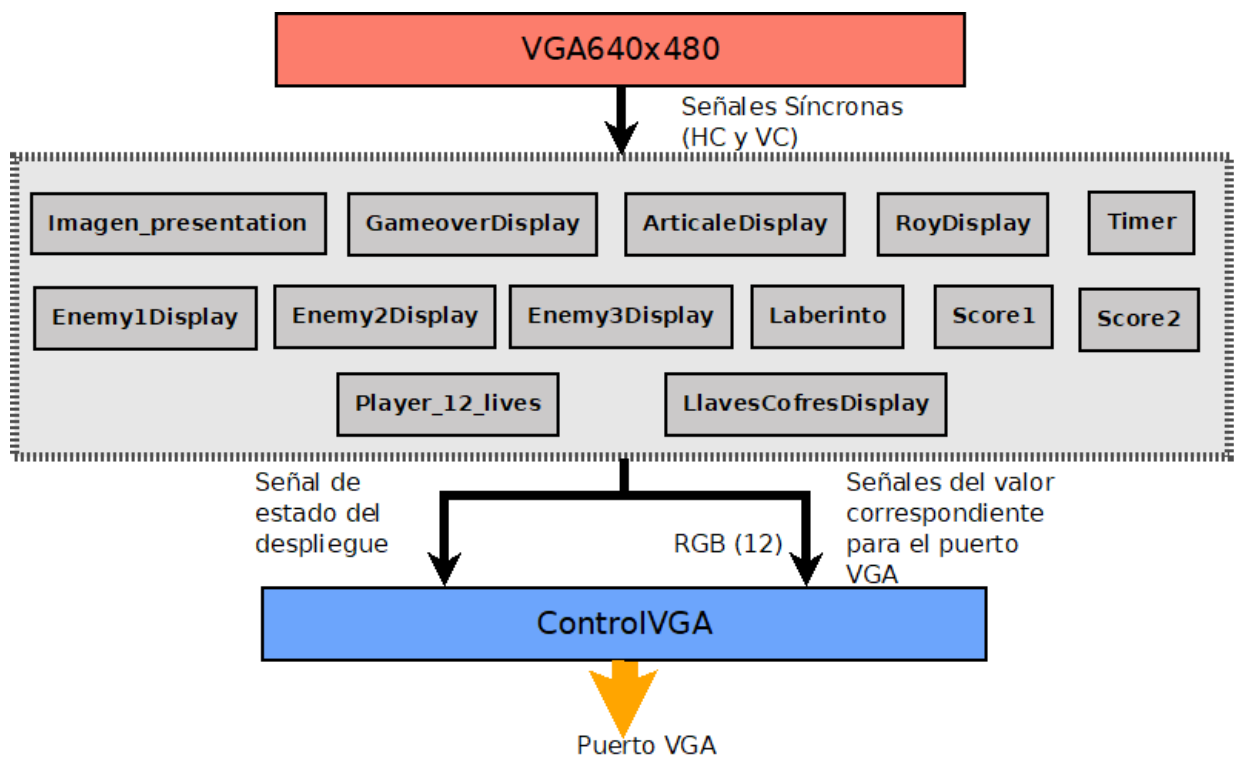


Figura 4. 19 Controlador de señales hacia el puerto VGA.

Como se habló en el capítulo anterior, el controlador del sistema de procesamiento de imágenes debe tener una jerarquía para cada imagen y es definida por este sistema como se puede observar en el Pseudocódigo 6, se define la jerarquía que tiene cada una de las imágenes a desplegar. Este modelo está diseñado para el puerto VGA evitando que exista una superposición de señales recibidas por todos los módulos que contengan imágenes.

Para llevar a cabo el control del puerto VGA, es necesario conocer la conexión entre la tarjeta BASYS 3 y el puerto VGA como se muestra en la figura 4.20. La ventaja de utilizar una jerarquía en el despliegue de imágenes conlleva el darle el acceso a cada imagen según su privilegio definido, evitando la superposición de señales y asegurando que solamente pase una sola señal de imagen por píxel hacia el puerto VGA.

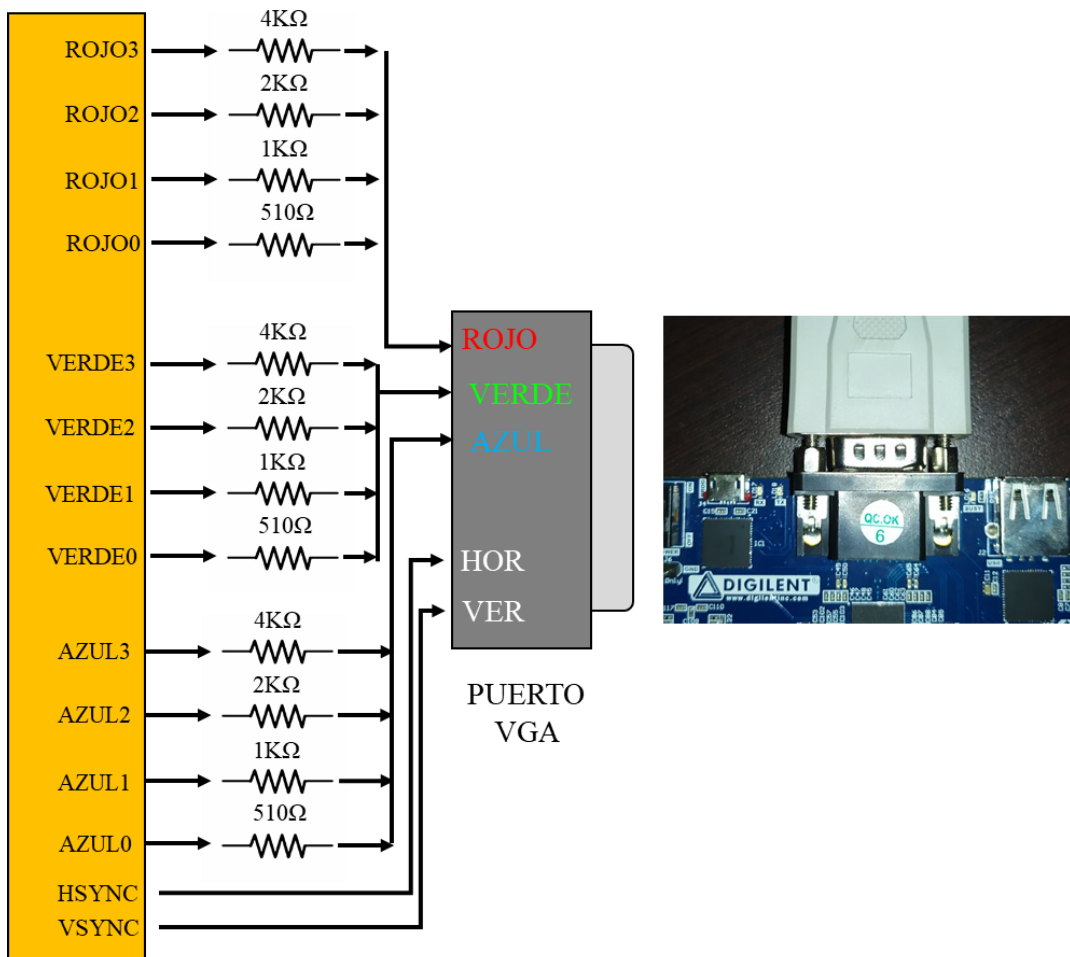
Al tener en control las señales de las imágenes sobre el puerto VGA, la entrada de datos por medio de los mandos, el flujo de la aplicación y la salida de audio por medio de la tarjeta de desarrollo, en el siguiente capítulo se hablará sobre los resultados obtenidos sobre la aplicación del sistema implementado sobre este dispositivo.

```

Algoritmo controladorDeImag
  Mientras master_reset = 0 Hacer
    Leer master_reset
    Leer PresentationON, PresentationM
    Leer GameoverON, GameoverM
    Leer ArticON, ArticM
    leer RoyON, RoyM
    Leer LaberintoON, LaverintoM
    Si PresentationON = 1 Entonces
      SalidaVGA <- PresentationM
    SiNo
      si GameoverON = 1 Entonces
        SalidaVGA <- GameoverM
      SiNo
        si ArticON = 1 Entonces
          SalidaVGA <- ArticM
        SiNo
          si RoyON = 1 Entonces
            SalidaVGA <- RoyM
          SiNo
            si LaberintoON= 1 Entonces
              SalidaVGA <-
                LaberintoON
            SiNo
              SalidaVGA <- 0
            FinSi
          FinSi
        FinSi
      FinSi
    FinSi
  Fin Si
Fin Mientras
FinAlgoritmo

```

Pseudocódigo 6 - Algoritmo para definición de la jerarquía.



FPGA : ARTIX - 7

Figura 4. 20 Conexiones entre el puerto VGA y el FPGA.

CAPÍTULO 5 RESULTADOS

El manejo total de la aplicación conlleva el uso de ciertos recursos que ofrece la tarjeta, la figura 5.1 muestra los recursos utilizados de tipo Lógica Programable (PL) y el uso de los bloques de memoria como los repositorios de las imágenes utilizadas.

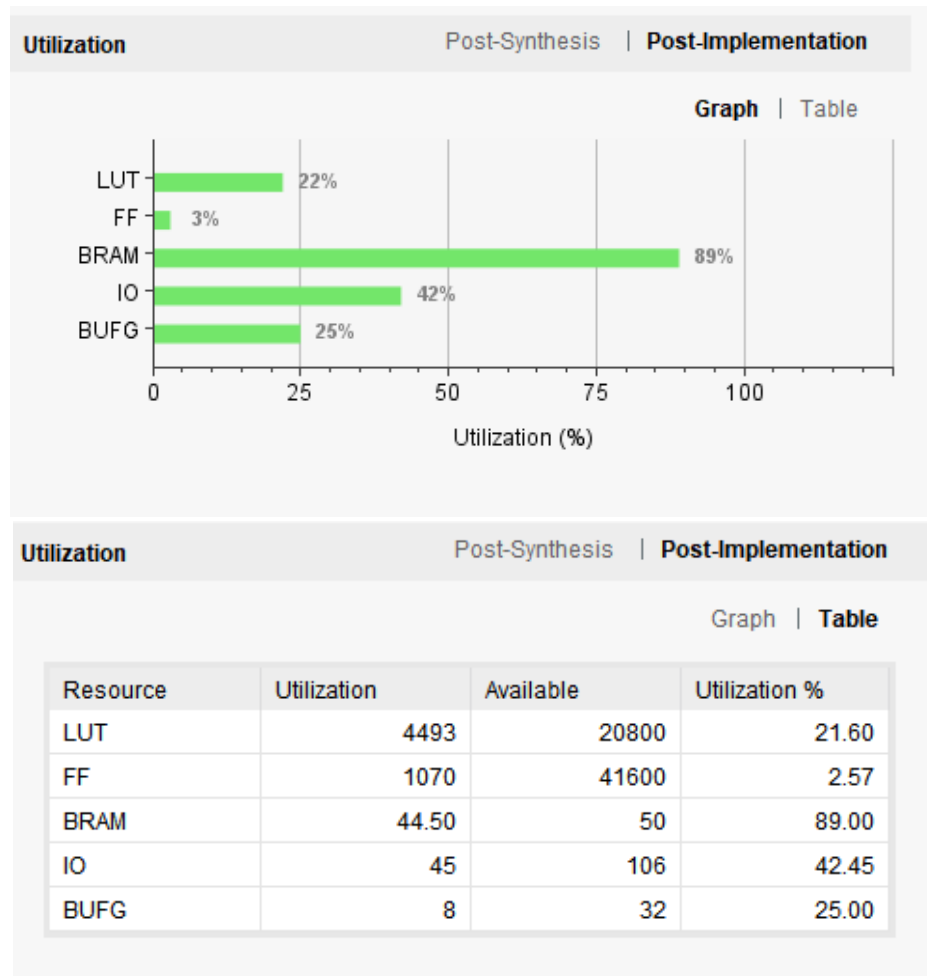


Figura 5. 1 Recursos utilizados en la Tarjeta BASYS 3.

Los resultados muestran el manejo de los recursos como lo son los LUTs²⁴, los puertos de entrada y salida (IO), los buffers globales (BUFG), flip-flops disponibles (FF) y los bloques

²⁴ LUT (Por sus siglas en inglés *LookUp Table*) es conocido como tabla de búsquedas o tablas de verdad, es una unidad de medida utilizadas para implementar la lógica combinacional y ser almacenadas en tablas de verdad.

de memoria (BRAM). Donde destaca con un 89% principalmente en el uso de los bloques de memoria donde se resguardan los datos correspondientes de los audios y las imágenes.

En la tabla 6 se muestra el tamaño de cada imagen guardada, la cantidad de bits por celda, el total de celdas por crear, los bloques de memoria utilizados y el tamaño de las direcciones creadas. Se puede observar en los resultados que de 1800 Kb que dispone la tarjeta BASYS 3, se ocuparon 9 bloques de 18 Kb y 22 bloques de 36 Kb en las imágenes.

En cuanto a la funcionalidad de la aplicación se utilizaron LEDs indicadores para conocer en tiempo real de ejecución los estados enviados por el sistema orquestador los cuales pueden observarse en la figura 5.2 indicando el LED utilizado y el significado de estos. Estos indicadores fueron utilizados solamente para comprobar el estado de la aplicación y comprobar los procesos que corresponden a cada señal. En la figura 5.3 se puede observar un ejemplo de la implementación y observación de estos indicadores.

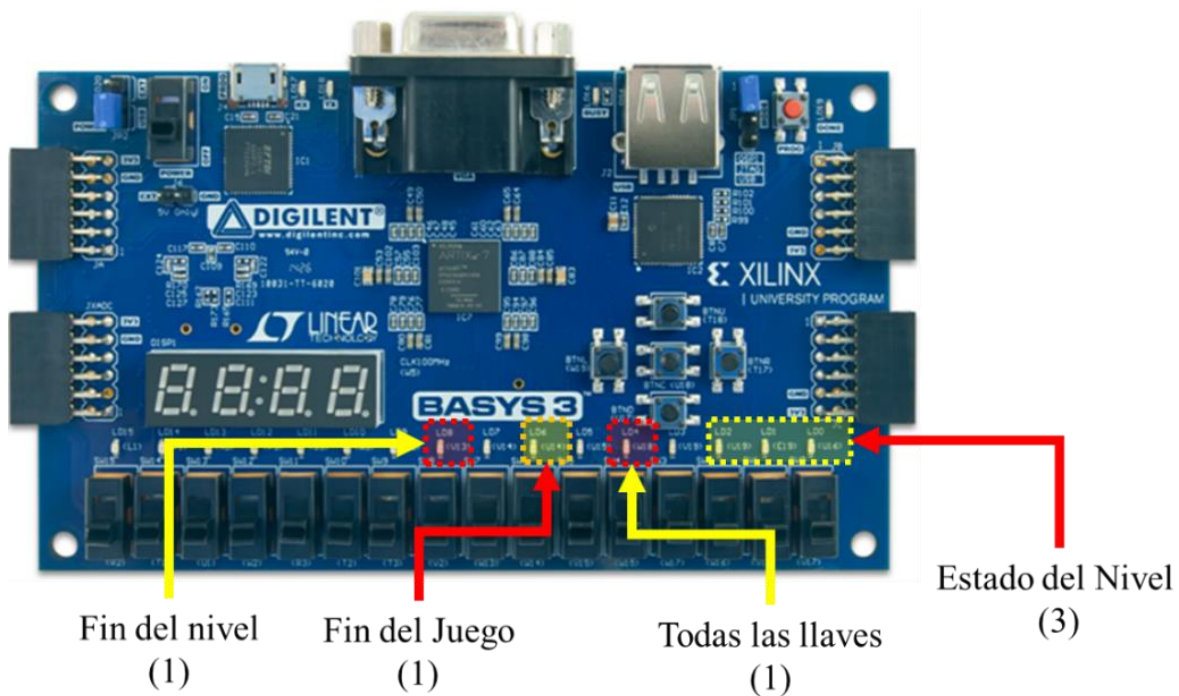


Figura 5. 2 Indicadores de las señales enviadas por el núcleo del sistema.

<i>Imagen</i>	Tamaño de la imagen	Cantidad de bits por celda (bits)	Total de celdas (unidad)	Total de Bits	Bloques de 16Kb	Bloques de 32Kb	Tamaño de las direcciones (bits)
<i>ArticaleROM</i>	15x120	12	1800	21 Kb	0	1	11 (10 down to 0)
<i>RoyROM</i>	15x120	12	1800	21 Kb	0	1	11 (10 down to 0)
<i>EnemyROM</i>	15x90	12	1350	16.2 Kb	0	1	11 (10 down to 0)
<i>GameoverROM</i>	150x30	12	4500	54 Kb	1	2	13 (12 down to 0)
<i>Title1ROM</i>	240x160	12	38400	406.8 Kb	3	12	16 (15 down to 0)
<i>Title2ROM</i>	120x15	12	1800	21.6 Kb	0	1	11 (10 down to 0)
<i>Letras1ROM</i>	64x40	12	2560	30.72 Kb	1	1	12 (11 down to 0)
<i>Letras2ROM</i>	80x40	12	3200	38.4 Kb	1	1	12 (11 down to 0)
<i>LlaveGrisROM</i>	15x15	12	225	3.06 Kb	1	0	8 (7 down to 0)
<i>LlaveOroROM</i>	15x15	12	225	3.06 Kb	1	0	8 (7 down to 0)
<i>NúmerosROM</i>	16x330	12	5280	63.36 Kb	1	2	13 (12 down to 0)
Total	-	12	61140 celdas	679.2 Kb	9	22	-
<i>Total de bits requeridos:</i>			61140 celdas	679.2 Kb	-	-	679.2 Kb
<i>Total de bits utilizados:</i>			73368 celdas	-	144 Kb	704 Kb	848 Kb
<i>Total de bits no utilizados:</i>							168.80 Kb

Tabla 6 Tamaño y gestión de datos para los bloques de memoria.

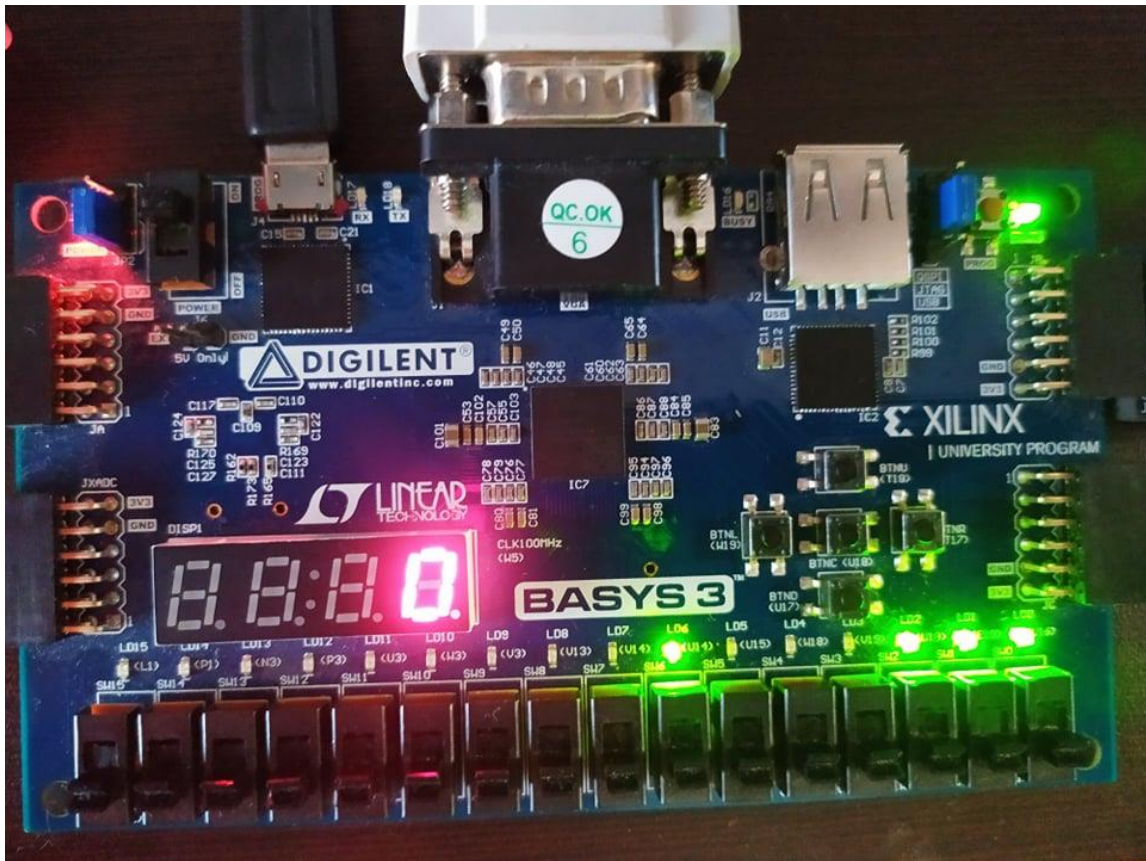


Figura 5. 3 Indicadores del estado “fin de juego”.

El dispositivo en funcionamiento puede observarse en la siguiente figura 5.4, donde se muestran algunos estados implementados sobre la tarjeta BASYS 3 (como lo es la presentación y el nivel 1 recibidos por el sistema orquestador, funcionamiento del temporizador, del puntaje obtenido y el despliegue de las imágenes en pantalla).

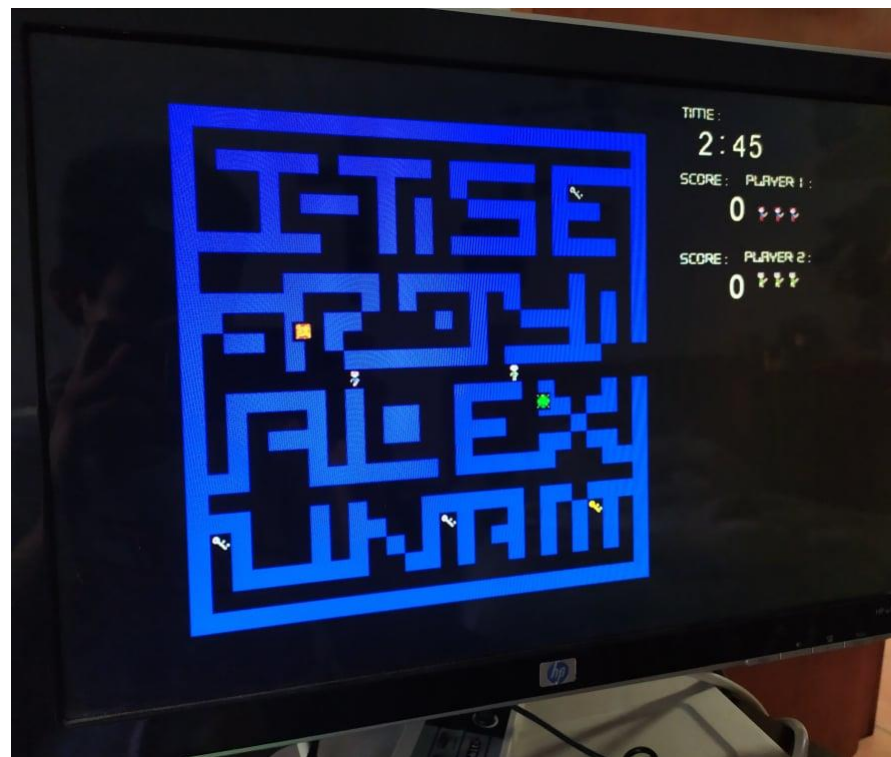
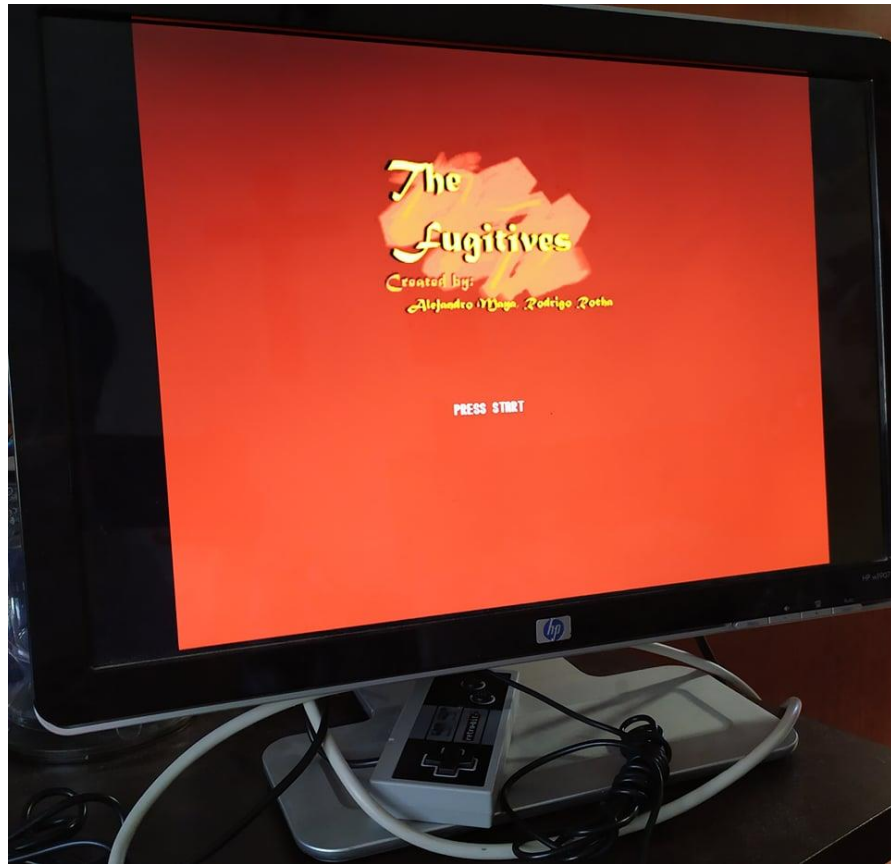


Figura 5. 4 Estado de presentación y Nivel 1.



Figura 5. 5 Estado Nivel 2 y Nivel 3.

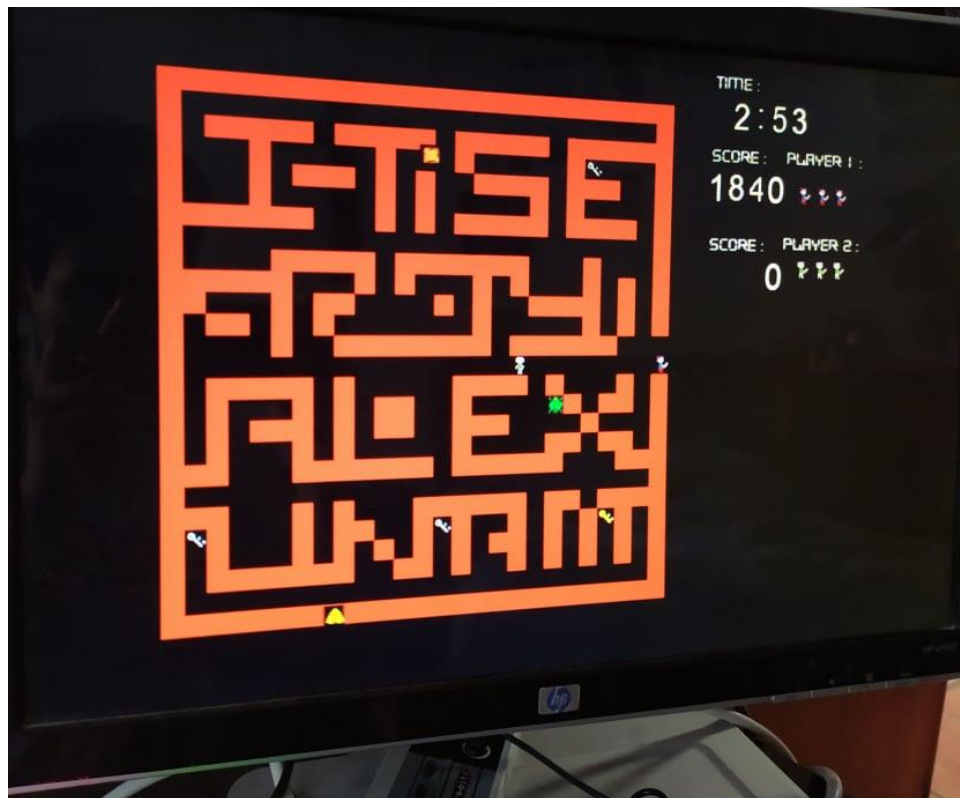
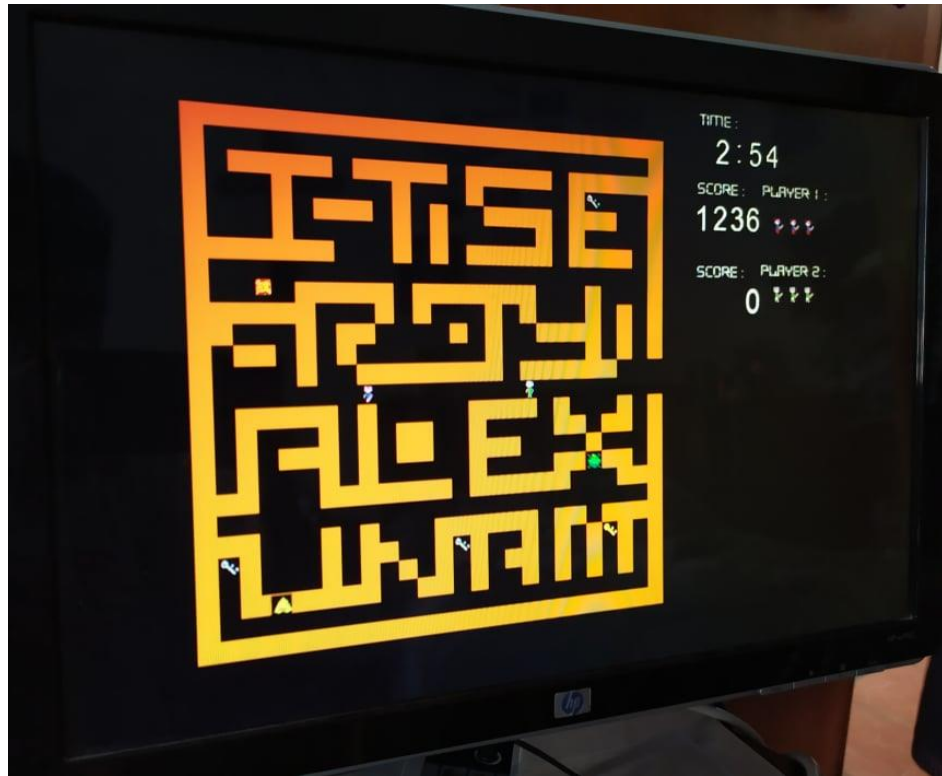


Figura 5. 6 Estado Nivel 4 y Nivel 5.

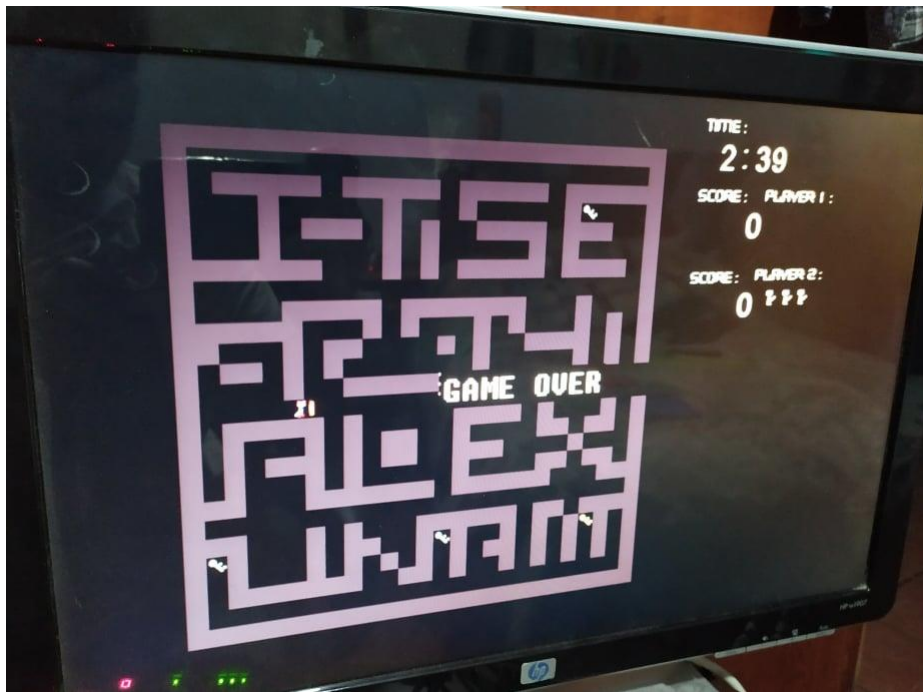


Figura 5. 7 Estado Fin del juego y Juego Ganado.

CONCLUSIONES

A lo largo de este trabajo de investigación, se desarrolló la idea de diseñar e implementar una consola de videojuegos aplicándolo como un sistema de procesamiento de imágenes, utilizando los recursos disponibles que tiene un FPGA. Primeramente, se desarrollaron los módulos correspondientes a los protocolos y estándares utilizados para el funcionamiento de la aplicación propuesta, la cual fue un videojuego implementado por medio del lenguaje de descripción de hardware (VHDL).

Dentro de las ventajas del diseño fue la utilización del estándar VGA para el despliegue de imágenes y el uso del protocolo I2S para la reproducción de audio, donde pueden utilizarse de forma independiente a la aplicación, es decir, que se pueden utilizar en otros procesos que involucren a uno o ambos de los diseños mencionados anteriormente, por ejemplo, el filtrado de audio digital o análisis de imágenes haciendo los ajustes correspondientes.

Otra ventaja es que el diseño es independiente de la tarjeta de desarrollo que se ocupe, para este caso se diseñó en una tarjeta BASYS 3, por el costo-beneficio respecto a otras tarjetas de desarrollo, como se mencionó en el capítulo 4, sin embargo, si se desea migrar el proyecto a otra tarjeta si es posible, siempre y cuando sea considerada la capacidad de memoria del dispositivo a usar y la cantidad de puertos de entrada y salida de que dispone para realizar las modificaciones necesarias.

La principal desventaja del diseño se presenta en la aplicación propuesta, es decir, el videojuego, esto se debe a que los módulos que lo componen tienen demasiadas dependencias entre sí, a diferencia de los protocolos que solo reciben información y la reproducen, un cambio en alguno de los componentes implica modificar dos o más módulos, lo cual podría causar errores si no se conoce correctamente la lógica del juego.

El sistema desarrollado puede optimizarse de diversas formas con el fin de mejorar la experiencia de uso, por ejemplo, cambiar el estándar VGA por el estándar HDMI para mejorar la calidad de la imagen, por supuesto, esto conlleva mejorar el manejo de memoria, por lo que se podría implementar un módulo capaz de leer las imágenes de una memoria externa por medio

del puerto USB, incrementando la cantidad de imágenes que se pueden utilizar y de igual forma su resolución. Por una parte, se puede mejorar la calidad y duración del audio, en la aplicación solo se usan pequeños audios, que utilizando otro medio de almacenamiento podría aumentar tanto la calidad cómo la cantidad de sonidos reproducidos, creando una experiencia más dinámica para el usuario.

Otra forma de optimizar los recursos del sistema es sobre el tratamiento de las imágenes y el espacio de memoria asignada. Es importante que, durante el proceso de diseño e implementación de imágenes, se debe tener en consideración el tamaño de la resolución de las imágenes. Uno de los casos presentados se da en el momento de la creación de los bloques de memoria, los cuales son directamente dependientes de la cantidad de datos que se requieran almacenar. Estos datos son proporcionales al tamaño de la imagen, se puede controlar las medidas de tal forma que puedan beneficiar en el momento la creación y distribución de memoria, impidiendo el despliegue de huecos que no serán utilizados ni leídos por el sistema.

En los resultados obtenidos se observa que se hizo un uso eficiente de la tarjeta de desarrollo, como resultado se utilizó un porcentaje muy bajo de los recursos lógicos disponibles en el FPGA, lo que nos permite concluir que el hecho de aprovechar los algoritmos para desarrollar los protocolos y haciendo el diseño de las máquinas de estados finitos de manera óptima y estructurada repercute directamente en el hardware necesario para el desarrollo de un proyecto, lo cual no sólo va a beneficiar en, por ejemplo, la cantidad de energía necesaria para alimentar el sistema, o la velocidad de ejecución de algún módulo, también nos permite migrar a una tarjeta FPGA más económica sin que afecte en gran medida el funcionamiento de la aplicación.

El trabajo desarrollado constituye una base para el aprendizaje de un lenguaje de descripción aplicado durante el desarrollo de un sistema de procesamiento de imágenes. Asimismo, presenta el uso eficiente de los recursos de un dispositivo destinado al diseño con lógica programable; así como los recursos de los que podemos disponer en una tarjeta de desarrollo. Se da especial importancia a la aplicación de los conceptos fundamentales del diseño de sistemas digitales para el diseño de hardware, protocolos de comunicación y uso de interconexiones estándares.

APÉNDICE

Apéndice 1.- Código del sistema - Divisor de frecuencia.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.STD_LOGIC_unsigned.all;
entity clk_div is
    port (
        mclk : in STD_LOGIC;
        clr : in STD_LOGIC;
        level_state : in std_logic_vector(2 downto 0);
        clk25M : out std_logic;
        clk190 : out std_logic;
        clk10 : out std_logic;
        clk1 : out std_logic;
        EnemyClk : out std_logic
    );
end clk_div;
architecture Behavioral of clk_div is
    signal temp : STD_LOGIC;
    signal temp1 : STD_LOGIC;
    signal temp2 : STD_LOGIC;
    signal temp3 : STD_LOGIC;
    signal counter : integer range 0 to 1 := 0;
    signal counter1 : integer range 0 to 262999 := 0;
    signal counter2 : integer range 0 to 333333 := 0;
    signal counter3 : integer range 0 to 50000000 := 0;
    signal level_1, level_2, level_3, level_4, level_5 : STD_logic;
    signal co_level_1 : integer range 0 to 1300000 := 0;
    signal co_level_2 : integer range 0 to 1050000 := 0;
    signal co_level_3 : integer range 0 to 850000 := 0;
    signal co_level_4 : integer range 0 to 600000 := 0;
    signal co_level_5 : integer range 0 to 550000 := 0;
begin
    process (mclk, clr)
    begin
        if (clr = '1') then
            temp <= '0';
            counter <= 0;
        elsif (rising_edge(mclk)) then
            if (counter = 1) then
                temp <= not(temp);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
    clk25M <= temp;

    process (mclk, clr)
    begin
        if (clr = '1') then
            temp1 <= '0';
```

```

        counter1 <= 0;
    elsif (rising_edge(mclk)) then
        if (counter1 = 262999) then
            temp1 <= not(temp1);
            counter1 <= 0;

        else
            counter1 <= counter1 + 1;
        end if;
    end if;
end process;
clk190 <= temp1;
process (mclk, clr)
begin
    if (clr = '1') then
        temp2 <= '0';
        counter2 <= 0;
    elsif (rising_edge(mclk)) then
        if (counter2 = 333333) then
            temp2 <= not(temp2);
            counter2 <= 0;

        else
            counter2 <= counter2 + 1;
        end if;
    end if;
end process;
clk10 <= temp2;
process (mclk, clr)
begin
    if (clr = '1') then
        temp3 <= '0';
        counter3 <= 0;
    elsif (rising_edge(mclk)) then
        if (counter3 = 49999999) then
            temp3 <= not(temp3);
            counter3 <= 0;

        else
            counter3 <= counter3 + 1;
        end if;
    end if;
end process;
clk1 <= temp3;

process (mclk, clr)
begin
    if (clr = '1') then
        level_1 <= '0';
        co_level_1 <= 0;
    elsif (rising_edge(mclk)) then
        if (co_level_1 = 1299999) then
            level_1 <= not(level_1);
            co_level_1 <= 0;

        else
            co_level_1 <= co_level_1 + 1;
        end if;
    end if;
end process;
process (mclk, clr)

```

```

begin
    if (clr = '1') then
        level_2 <= '0';
        co_level_2 <= 0;
    elsif (rising_edge(mclk)) then
        if (co_level_2 = 1049999) then
            level_2 <= not(level_2);
            co_level_2 <= 0;
        else
            co_level_2 <= co_level_2 + 1;
        end if;
    end if;
end process;
process (mclk, clr)
begin
    if (clr = '1') then
        level_3 <= '0';
        co_level_3 <= 0;
    elsif (rising_edge(mclk)) then
        if (co_level_3 = 849999) then
            level_3 <= not(level_3);
            co_level_3 <= 0;
        else
            co_level_3 <= co_level_3 + 1;
        end if;
    end if;
end process;
process (mclk, clr)
begin
    if (clr = '1') then
        level_4 <= '0';
        co_level_4 <= 0;
    elsif (rising_edge(mclk)) then
        if (co_level_4 = 599999) then
            level_4 <= not(level_4);
            co_level_4 <= 0;
        else
            co_level_4 <= co_level_4 + 1;
        end if;
    end if;
end process;
process (mclk, clr)
begin
    if (clr = '1') then
        level_5 <= '0';
        co_level_5 <= 0;
    elsif (rising_edge(mclk)) then
        if (co_level_5 = 549999) then
            level_5 <= not(level_5);
            co_level_5 <= 0;
        else
            co_level_5 <= co_level_5 + 1;
        end if;
    end if;
end process;
process (level_state)
begin

```

```
    if (level_state = "001") then
        enemyCLK <= level_1;
    elsif (level_state = "010") then
        enemyCLK <= level_2;
    elsif (level_state = "011") then
        enemyCLK <= level_3;
    elsif (level_state = "100") then
        enemyCLK <= level_4;
    elsif (level_state = "101") then
        enemyCLK <= level_5;
    else
        enemyCLK <= '0';
    end if;
end process;
end Behavioral;
```

Apéndice 2.- Código del sistema - VGA640x480.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity vga_640x480 is
    port (
        clk, clr : in std_logic;
        hsync : out std_logic;
        vsync : out std_logic;
        hc : out std_logic_vector(9 downto 0);
        vc : out std_logic_vector(9 downto 0);
        vidon : out std_logic
    );
end vga_640x480;
architecture Behavioral of vga_640x480 is
    constant hbp : std_logic_vector (9 downto 0) := "0010010000";
    constant hfp : std_logic_vector (9 downto 0) := "1100010000";
    constant hpixels : std_logic_vector (9 downto 0) := "1100100000";
    constant vbp : std_logic_vector (9 downto 0) := "0000011111";
    constant vfp : std_logic_vector (9 downto 0) := "0111111111";
    constant vlines : std_logic_vector (9 downto 0) := "1000001001";
    signal hcs, vcs : std_logic_vector (9 downto 0);
    signal vsenable : std_logic;
begin
    process (clk, clr)
    begin
        if clr = '1' then
            hcs <= "0000000000";
        elsif (rising_edge(clk)) then
            if (hcs = hpixels - 1) then
                hcs <= "0000000000";
                vsenable <= '1';
            else
                hcs <= hcs + 1;
                vsenable <= '0';
            end if;
        end if;
    end process;
    hsync <= '0' when (hcs < 145) else '1';

    process (clk, clr, vsenable)
    begin
        if clr = '1' then
            vcs <= "0000000000";
        elsif ((rising_edge(clk)) and (vsenable = '1')) then
            if vcs = vlines - 1 then
                vcs <= "0000000000";
            else
                vcs <= vcs + 1;
            end if;
        end if;
    end process;
    vsync <= '0' when vcs < 5 else '1';
    vidon <= '1' when ((hcs < hfp) and (hcs >= hbp))
        and ((vcs < vfp) and (vcs >= vbp)) else '0';
end architecture;
```

```
        hc <= hcs;  
        vc <= vcs;  
end Behavioral;
```


Apéndice 3.- Código del sistema - Laberinto.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity Laberinto is
    port (
        vidon : in STD_LOGIC;
        hc : in STD_LOGIC_VECTOR (9 downto 0);
        vc : in STD_LOGIC_VECTOR (9 downto 0);
        red : out STD_LOGIC_VECTOR (3 downto 0);
        green : out STD_LOGIC_VECTOR (3 downto 0);
        blue : out STD_LOGIC_VECTOR (3 downto 0);
        inicio : in STD_LOGIC;
        level_state : in std_logic_vector(2 downto 0);
        clk1 : in std_logic;
        master_clk : in STD_LOGIC;
        reset : in STD_LOGIC;
        MazeON : out STD_LOGIC
    );
end Laberinto;
architecture Behavioral of Laberinto is
    constant hbp : std_logic_vector(9 downto 0) := "0010010000";
    constant vbp : std_logic_vector(9 downto 0) := "0000011111";

    constant w : integer := 420;
    constant h : integer := 479;
    signal spriteon : std_logic;
    signal sstart : std_logic;
    signal srojo : std_logic_vector (3 downto 0);
    signal sverde : std_logic_vector (3 downto 0);
    signal sazul : std_logic_vector (3 downto 0);
    signal hc1, vc1 : std_logic_vector (9 downto 0);
    type state_type is (start, check, levelone, leveltwo, levelthree, levelfour,
        levelfive, wingame, gameover);
    signal state : state_type;
    signal counter_wingame : integer range 0 to 5 := 0;
begin
    process (master_clk, reset, inicio)
    begin
        if reset = '1' then
            state <= start;
            sstart <= '0';
            srojo <= "0000";
            sverde <= "0000";
            sazul <= "0000";
        elsif (rising_edge(master_clk)) then
            case state is
                when start =>
                    if inicio = '1' then
                        state <= levelone;
                        sstart <= '1';
                    else
                        state <= start;
                        sstart <= '0';
                    end if;
                when check =>
```

```

        if (level_state = "001") then
            state <= levelone;
        elsif (level_state = "010") then
            state <= leveltwo;
        elsif (level_state = "011") then
            state <= levelthree;
        elsif (level_state = "100") then
            state <= levelfour;
        elsif (level_state = "101") then
            state <= levelfive;
        elsif (level_state = "110") then
            state <= wingame;
        else
            state <= gameover;
        end if;
    when levelone =>
        srojo <= "0001";
        sverde <= "0011";
        sazul <= "1101";
        state <= check;
    when leveltwo =>
        srojo <= "0001";
        sverde <= "1010";
        sazul <= "0010";
        state <= check;
    when levelthree =>
        srojo <= "1001";
        sverde <= "1100";
        sazul <= "0011";
        state <= check;
    when levelfour =>
        srojo <= "1011";
        sverde <= "0110";
        sazul <= "0001";
        state <= check;
    when levelfive =>
        srojo <= "1011";
        sverde <= "0010";
        sazul <= "0000";
        state <= check;
    when wingame =>
        if (counter_wingame = 1) then
            srojo <= "0001";
            sverde <= "0011";
            sazul <= "1101";
        elsif (counter_wingame = 2) then
            srojo <= "0001";
            sverde <= "1010";
            sazul <= "0010";
        elsif (counter_wingame = 3) then
            srojo <= "1001";
            sverde <= "1100";
            sazul <= "0011";
        elsif (counter_wingame = 4) then
            srojo <= "1011";
            sverde <= "0110";
            sazul <= "0001";
        end if;
    end when;
end process;

```

```

        elsif (counter_wingame = 5) then
            srojo <= "1011";
            sverde <= "0010";
            sazul <= "0000";

        end if;
        state <= check;
    when gameover =>
        srojo <= "0010";
        sverde <= "0010";
        sazul <= "0010";
        state <= check;
    end case;
end if;
end process;
process (clk1)
begin
    if (reset = '1') then
        counter_wingame <= 0;
    elsif (rising_edge(clk1)) then
        if (counter_wingame = 5) then
            counter_wingame <= 0;
        else
            counter_wingame <= counter_wingame + 1;
        end if;
    end if;
end process;
hc1 <= hc - hbp;
vc1 <= vc - vbp;
spriteon <= '1' when ((hc > hbp) and (hc <= hbp + w) and
    ((vc >= vbp) and (vc < vbp + h))) and sstart = '1' else '0';
MazeON <= spriteon;
process (spriteon, vidon, hc, vc)
begin
    red <= "0000";
    green <= "0000";
    blue <= "0000";
    if spriteon = '1' and vidon = '1' then
        if ((hc1 >= 30 and hc1 <= 420 and vc1 >= 45 and vc1 < 60) or
            (hc1 >= 30 and hc1 <= 45 and vc1 >= 60 and vc1 < 435) or
            (hc1 > 405 and hc1 <= 420 and vc1 >= 45 and vc1 < 225) or
            (hc1 > 405 and hc1 <= 420 and vc1 >= 255 and vc1 < 435) or
            (hc1 >= 30 and hc1 <= 420 and vc1 >= 420 and vc1 < 435) or
            ((hc1 > 60 and hc1 <= 135 and vc1 >= 75 and vc1 < 90) or
            (hc1 > 90 and hc1 <= 105 and vc1 >= 90 and vc1 < 135) or
            (hc1 > 45 and hc1 <= 135 and vc1 >= 135 and vc1 < 150)) or
            ((hc1 > 120 and hc1 <= 165 and vc1 >= 105 and vc1 < 120)) or
            ((hc1 > 150 and hc1 <= 225 and vc1 >= 75 and vc1 < 90) or
            (hc1 > 180 and hc1 <= 195 and vc1 >= 90 and vc1 < 150)) or
            ((hc1 > 210 and hc1 <= 225 and vc1 >= 105 and vc1 < 150)) or
            ((hc1 > 240 and hc1 <= 315 and vc1 >= 75 and vc1 < 90) or
            (hc1 > 240 and hc1 <= 255 and vc1 >= 90 and vc1 < 105) or
            (hc1 > 255 and hc1 <= 315 and vc1 >= 105 and vc1 < 120) or
            (hc1 > 300 and hc1 <= 315 and vc1 >= 120 and vc1 < 135) or
            (hc1 > 240 and hc1 <= 315 and vc1 >= 135 and vc1 < 150)) or
            ((hc1 > 345 and hc1 <= 405 and vc1 >= 75 and vc1 < 90) or
            (hc1 > 330 and hc1 <= 345 and vc1 >= 75 and vc1 < 150) or
            (hc1 > 345 and hc1 <= 375 and vc1 >= 105 and vc1 < 120) or

```

(hc1 > 345 and hc1 <= 390 and vc1 >= 135 and vc1 < 150) or
(hc1 > 45 and hc1 <= 105 and vc1 >= 180 and vc1 < 195) or
(hc1 > 45 and hc1 <= 60 and vc1 >= 195 and vc1 < 210) or
(hc1 > 60 and hc1 <= 90 and vc1 >= 210 and vc1 < 225) or
(hc1 > 105 and hc1 <= 120 and vc1 >= 165 and vc1 < 240) or
(hc1 > 120 and hc1 <= 180 and vc1 >= 165 and vc1 < 180) or
(hc1 > 165 and hc1 <= 180 and vc1 >= 180 and vc1 < 195) or
(hc1 > 135 and hc1 <= 165 and vc1 >= 195 and vc1 < 210) or
(hc1 > 135 and hc1 <= 150 and vc1 >= 210 and vc1 < 225) or
(hc1 > 195 and hc1 <= 210 and vc1 >= 165 and vc1 < 195) or
(hc1 > 210 and hc1 <= 285 and vc1 >= 165 and vc1 < 180) or
(hc1 > 255 and hc1 <= 270 and vc1 >= 180 and vc1 < 225) or
(hc1 > 150 and hc1 <= 270 and vc1 >= 225 and vc1 < 240) or
(hc1 > 225 and hc1 <= 240 and vc1 >= 195 and vc1 < 210) or
(hc1 > 285 and hc1 <= 300 and vc1 >= 165 and vc1 < 195) or
(hc1 > 300 and hc1 <= 345 and vc1 >= 195 and vc1 < 210) or
(hc1 > 345 and hc1 <= 360 and vc1 >= 165 and vc1 < 225) or
(hc1 > 285 and hc1 <= 375 and vc1 >= 225 and vc1 < 240) or
(hc1 > 375 and hc1 <= 390 and vc1 >= 180 and vc1 < 225) or
(hc1 > 45 and hc1 <= 75 and vc1 >= 315 and vc1 < 330) or
(hc1 > 60 and hc1 <= 75 and vc1 >= 255 and vc1 < 315) or
(hc1 > 75 and hc1 <= 135 and vc1 >= 255 and vc1 < 270) or
(hc1 > 120 and hc1 <= 135 and vc1 >= 270 and vc1 < 315) or
(hc1 > 90 and hc1 <= 120 and vc1 >= 285 and vc1 < 300) or
(hc1 > 120 and hc1 <= 150 and vc1 >= 315 and vc1 < 330) or
(hc1 > 150 and hc1 <= 165 and vc1 >= 255 and vc1 < 315) or
(hc1 > 150 and hc1 <= 225 and vc1 >= 315 and vc1 < 330) or
(hc1 > 240 and hc1 <= 255 and vc1 >= 255 and vc1 < 330) or
(hc1 > 255 and hc1 <= 300 and vc1 >= 255 and vc1 < 270) or
(hc1 > 255 and hc1 <= 285 and vc1 >= 285 and vc1 < 300) or
(hc1 > 255 and hc1 <= 315 and vc1 >= 315 and vc1 < 330) or
(hc1 > 180 and hc1 <= 210 and vc1 >= 270 and vc1 < 300) or
(hc1 > 315 and hc1 <= 330 and vc1 >= 255 and vc1 < 270) or
(hc1 > 315 and hc1 <= 345 and vc1 >= 270 and vc1 < 285) or
(hc1 > 360 and hc1 <= 390 and vc1 >= 270 and vc1 < 285) or
(hc1 > 375 and hc1 <= 390 and vc1 >= 255 and vc1 < 270) or
(hc1 > 345 and hc1 <= 360 and vc1 >= 285 and vc1 < 300) or
(hc1 > 315 and hc1 <= 330 and vc1 >= 300 and vc1 < 330) or
(hc1 > 330 and hc1 <= 345 and vc1 >= 300 and vc1 < 315) or
(hc1 > 360 and hc1 <= 390 and vc1 >= 300 and vc1 < 315) or
(hc1 > 375 and hc1 <= 405 and vc1 >= 315 and vc1 < 330) or
(hc1 > 45 and hc1 <= 75 and vc1 >= 345 and vc1 < 360) or
(hc1 > 60 and hc1 <= 75 and vc1 >= 360 and vc1 < 390) or
(hc1 > 60 and hc1 <= 150 and vc1 >= 390 and vc1 < 405) or
(hc1 > 120 and hc1 <= 135 and vc1 >= 345 and vc1 < 390) or
(hc1 > 150 and hc1 <= 165 and vc1 >= 345 and vc1 < 405) or
(hc1 > 165 and hc1 <= 180 and vc1 >= 360 and vc1 < 375) or
(hc1 > 180 and hc1 <= 195 and vc1 >= 375 and vc1 < 390) or
(hc1 > 195 and hc1 <= 210 and vc1 >= 390 and vc1 < 405) or
(hc1 > 210 and hc1 <= 225 and vc1 >= 345 and vc1 < 405) or
(hc1 > 225 and hc1 <= 300 and vc1 >= 345 and vc1 < 360) or
(hc1 > 240 and hc1 <= 255 and vc1 >= 360 and vc1 < 405) or
(hc1 > 270 and hc1 <= 285 and vc1 >= 375 and vc1 < 390) or
(hc1 > 285 and hc1 <= 300 and vc1 >= 345 and vc1 < 405) or
(hc1 > 315 and hc1 <= 330 and vc1 >= 345 and vc1 < 405) or
(hc1 > 330 and hc1 <= 345 and vc1 >= 345 and vc1 < 360) or
(hc1 > 345 and hc1 <= 360 and vc1 >= 360 and vc1 < 405) or

```
        (hc1 > 360 and hc1 <= 375 and vc1 >= 345 and vc1 < 360) or
        (hc1 > 375 and hc1 <= 390 and vc1 >= 345 and vc1 < 405) or
        (hc1 > 390 and hc1 <= 405 and vc1 >= 345 and vc1 < 360))
    ) then
        red <= srojo;
        green <= sverde;
        blue <= sazul;
    else
        red <= "0000";
        green <= "0000";
        blue <= "0000";
    end if;
end if;
end process;
end Behavioral;
```

Apéndice 4.- Código del sistema - ArticaleDisplay.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity ArticaleDisplay is
    port (
        vidon : in STD_LOGIC;
        hc : in STD_LOGIC_VECTOR (9 downto 0);
        vc : in STD_LOGIC_VECTOR (9 downto 0);
        M : in STD_LOGIC_VECTOR (11 downto 0);
        botones : in STD_LOGIC_VECTOR (3 downto 0);
        rom_addr16 : out STD_LOGIC_VECTOR (10 downto 0);
        red : out STD_LOGIC_VECTOR (3 downto 0);
        green : out STD_LOGIC_VECTOR (3 downto 0);
        blue : out STD_LOGIC_VECTOR (3 downto 0);
        ArticON : out STD_LOGIC;
        ArticCLK : in STD_LOGIC;
        reset : in STD_LOGIC;
        inicio : in STD_LOGIC;
        articale_state : in STD_LOGIC;
        vidas : out STD_LOGIC_VECTOR (1 downto 0);
        player1_dead : out std_logic;
        end_level : in std_logic;
        end_game : in std_logic;
        AC_Xn : out STD_LOGIC_VECTOR (9 downto 0);
        AC_Yn : out STD_LOGIC_VECTOR (9 downto 0)
    );
end ArticaleDisplay;
architecture Behavioral of ArticaleDisplay is
    constant hbp : std_logic_vector(9 downto 0) := "0010010000";
    constant vbp : std_logic_vector(9 downto 0) := "0000011111";

    constant w : integer := 15;
    constant h : integer := 15;

    signal xpix, ypix : std_logic_vector(9 downto 0);
    signal spriteon : std_logic;
    signal C1, R1 : std_logic_vector(9 downto 0);
    signal corrimiento : std_logic_vector(10 downto 0);
    type state_type is (stop, right_u, right_d, left_u, left_d, up_u, up_d, down_u,
        down_d, check, playerdead, gameover, start);
    signal state, last_state, next_state, next_statel : state_type;
    signal direccion, direccion_actual : std_logic_vector(3 downto 0);
    signal C1_N, R1_N, AC_X, AC_Y, LC1, LR1, AC_X1, AC_Y1 : std_logic_vector(9 downto 0);
    signal life : std_logic_vector (1 downto 0);
    signal sstart : std_logic;
    signal passo, passa : std_logic;

begin
    ypix <= vc - vbp - R1;
    xpix <= hc - hbp - C1;
    spriteon <= '1' when ((hc > C1 + hbp) and (hc <= C1 + hbp + w) and
        ((vc >= R1 + vbp) and (vc < R1 + vbp + h))) and sstart = '1' else '0';
    ArticON <= spriteon;
    vidas <= life;
    process (xpix, ypix, corrimiento)
        variable rom_addr1 : std_logic_vector(13 downto 0);
```

```

variable rom_addr2 : std_logic_vector(15 downto 0);
begin
    rom_addr1 := (ypix & "000") +
        ("00" & ypix & "00") +
        ("000" & ypix & "0") +
        ("000" & ypix);
    rom_addr2 := rom_addr1 + ("000000" & xpix);
    rom_addr16 <= rom_addr2(10 downto 0) + corrimiento;

end process;
process (spriteon, vidon, M)
begin
    red <= "0000";
    green <= "0000";
    blue <= "0000";
    if spriteon = '1' and vidon = '1' then
        red <= M(11 downto 8);
        green <= M(7 downto 4);
        blue <= M(3 downto 0);
    end if;
end process;
process (ArticCLK, botones, reset, articale_state)
begin
    if reset = '1' then
        C1 <= "0010010110";
        R1 <= "0011110000";
        LC1 <= "0010010110";
        LR1 <= "0011110000";
        corrimiento <= "00111000010";
        state <= start;
        direccion <= "0100";
        AC_Xn <= C1 + 7;
        AC_Yn <= R1 + 7;
        sstart <= '0';
        life <= "00";
        player1_dead <= '0';
    elsif (end_level = '1') then
        C1 <= "0010010110";
        R1 <= "0011110000";
        LC1 <= "0010010110";
        LR1 <= "0011110000";
        corrimiento <= "00111000010";
        state <= start;
        direccion <= "0100";
        AC_Xn <= C1 + 7;
        AC_Yn <= R1 + 7;
        sstart <= '0';
        life <= life;
        player1_dead <= '0';
    elsif (rising_edge(ArticCLK)) then
        case state is
            when start =>
                if inicio = '1' then
                    state <= stop;
                    sstart <= '1';
                else
                    state <= start;
                end if;
            end case;
        end process;
end process;

```

```

                                sstart <= '0';
                                end if;
when stop =>
    C1 <= LC1;
    R1 <= LR1;
    last_state <= stop;
    corrimiento <= corrimiento;
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001") then
        direccion <= botones;
        state <= check;
    elsif (botones = "0000") then
        state <= stop;
    else
        state <= state;
        direccion <= direccion;
    end if;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
when check =>
    if (direccion = "0100") then
        AC_X <= C1 + 8;
        AC_Y <= R1 + 7;
        if (last_state = right_d) then
            next_state <= right_u;
        else
            next_state <= right_d;
        end if;
    elsif (direccion = "0010") then
        AC_X <= C1 + 6;
        AC_Y <= R1 + 7;
        if (last_state = left_d) then
            next_state <= left_u;
        else
            next_state <= left_d;
        end if;
    elsif (direccion = "0001") then
        AC_X <= C1 + 7;
        AC_Y <= R1 + 6;
        if (last_state = up_d) then
            next_state <= up_u;
        else
            next_state <= up_d;
        end if;
    elsif (direccion = "1000") then
        AC_X <= C1 + 7;
        AC_Y <= R1 + 8;
        if (last_state = down_d) then
            next_state <= down_u;
        else
            next_state <= down_d;
        end if;
    end if;
    if (direccion_actual = "0100") then
        AC_X1 <= C1 + 8;
        AC_Y1 <= R1 + 7;
        if (last_state = right_d) then

```



```

next_statel <= right_u;
else
next_statel <= right_d;
end if;
elsif (direccion_actual = "0010") then
AC_X1 <= C1 + 6;
AC_Y1 <= R1 + 7;
if (last_state = left_d) then
next_statel <= left_u;
else
next_statel <= left_d;
end if;
elsif (direccion_actual = "0001") then
AC_X1 <= C1 + 7;
AC_Y1 <= R1 + 6;
if (last_state = up_d) then
next_statel <= up_u;
else
next_statel <= up_d;
end if;
elsif (direccion_actual = "1000") then
AC_X1 <= C1 + 7;
AC_Y1 <= R1 + 8;
if (last_state = down_d) then
next_statel <= down_u;
else
next_statel <= down_d;
end if;
end if;
if ((AC_X >= 52 and AC_X <= 397 and AC_Y = 67) or
(AC_X = 52 and AC_Y >= 67 and AC_Y <= 127) or
(AC_X >= 52 and AC_X <= 82 and AC_Y >= 97 and AC_Y <= 127) or
(AC_X = 142 and AC_Y >= 67 and AC_Y <= 97) or
(AC_X >= 112 and AC_X <= 172 and AC_Y = 97) or
(AC_X = 112 and AC_Y >= 97 and AC_Y <= 127) or
(AC_X >= 112 and AC_X <= 172 and AC_Y = 127) or
(AC_X = 172 and AC_Y >= 97 and AC_Y <= 127) or
(AC_X = 232 and AC_Y >= 67 and AC_Y <= 97) or
(AC_X >= 202 and AC_X <= 232 and AC_Y = 97) or
(AC_X = 202 and AC_Y >= 97 and AC_Y <= 157) or
(AC_X >= 52 and AC_X <= 292 and AC_Y = 157) or
(AC_X = 232 and AC_Y >= 97 and AC_Y <= 157) or
(AC_X >= 232 and AC_X <= 247 and AC_Y >= 112 and AC_Y <= 127) or
(AC_X >= 232 and AC_X <= 292 and AC_Y = 127) or
(AC_X >= 262 and AC_X <= 322 and AC_Y = 97) or
(AC_X = 322 and AC_Y >= 67 and AC_Y <= 187) or
(AC_X >= 352 and AC_X <= 397 and AC_Y = 97) or
(AC_X >= 382 and AC_X <= 397 and AC_Y >= 97 and AC_Y <= 127) or
(AC_X >= 352 and AC_X <= 397 and AC_Y = 127) or
(AC_X = 397 and AC_Y >= 97 and AC_Y <= 307) or
(AC_X >= 142 and AC_X <= 172 and AC_Y >= 127 and AC_Y <= 157) or
(AC_X >= 292 and AC_X <= 397 and AC_Y = 157) or
(AC_X >= 307 and AC_X <= 337 and AC_Y >= 157 and AC_Y <= 187) or
(AC_X >= 367 and AC_X <= 397 and AC_Y >= 157 and AC_Y <= 172) or
(AC_X = 367 and AC_Y >= 157 and AC_Y <= 217) or
(AC_X >= 52 and AC_X <= 97 and AC_Y >= 157 and AC_Y <= 172) or
(AC_X = 187 and AC_Y >= 157 and AC_Y <= 217) or

```

```

(AC_X >= 67 and AC_X <= 97 and AC_Y = 202) or
(AC_X = 97 and AC_Y >= 202 and AC_Y <= 247) or
(AC_X = 52 and AC_Y >= 217 and AC_Y <= 307) or
(AC_X >= 52 and AC_X <= 97 and AC_Y >= 232 and AC_Y <= 247) or
(AC_X >= 97 and AC_X <= 337 and AC_Y = 247) or
(AC_X >= 127 and AC_X <= 142 and AC_Y >= 232 and AC_Y <= 247) or
(AC_X = 127 and AC_Y >= 187 and AC_Y <= 247) or
(AC_X >= 127 and AC_X <= 157 and AC_Y = 187) or
(AC_X >= 172 and AC_X <= 217 and AC_Y >= 202 and AC_Y <= 217) or
(AC_X >= 157 and AC_X <= 247 and AC_Y = 217) or
(AC_X = 247 and AC_Y >= 187 and AC_Y <= 217) or
(AC_X >= 217 and AC_X <= 247 and AC_Y = 187) or
(AC_X = 217 and AC_Y >= 187 and AC_Y <= 217) or
(AC_X = 277 and AC_Y >= 187 and AC_Y <= 217) or
(AC_X >= 277 and AC_X <= 337 and AC_Y = 217) or
(AC_X >= 277 and AC_X <= 292 and AC_Y >= 202 and AC_Y <= 217) or
(AC_X >= 337 and AC_X <= 412 and AC_Y = 247) or
(AC_X >= 382 and AC_X <= 412 and AC_Y >= 232 and AC_Y <= 247) or
(AC_X >= 337 and AC_X <= 367 and AC_Y >= 247 and AC_Y <= 262) or
(AC_X = 352 and AC_Y >= 247 and AC_Y <= 277) or
(AC_X >= 82 and AC_X <= 112 and AC_Y = 277) or
(AC_X = 82 and AC_Y >= 277 and AC_Y <= 382) or
(AC_X >= 52 and AC_X <= 397 and AC_Y = 337) or
(AC_X >= 82 and AC_X <= 112 and AC_Y >= 307 and AC_Y <= 382) or
(AC_X = 142 and AC_Y >= 247 and AC_Y <= 307) or
(AC_X >= 172 and AC_X <= 232 and AC_Y >= 247 and AC_Y <= 262) or
(AC_X = 172 and AC_Y >= 262 and AC_Y <= 307) or
(AC_X >= 172 and AC_X <= 232 and AC_Y = 307) or
(AC_X >= 217 and AC_X <= 232 and AC_Y >= 247 and AC_Y <= 307) or
(AC_X = 232 and AC_Y >= 247 and AC_Y <= 337) or
(AC_X >= 262 and AC_X <= 307 and AC_Y = 277) or
(AC_X >= 292 and AC_X <= 337 and AC_Y = 292) or
(AC_X >= 262 and AC_X <= 307 and AC_Y = 307) or
(AC_X = 307 and AC_Y >= 247 and AC_Y <= 307) or
(AC_X >= 292 and AC_X <= 307 and AC_Y >= 277 and AC_Y <= 307) or
(AC_X >= 367 and AC_X <= 397 and AC_Y = 292) or
(AC_X = 352 and AC_Y >= 307 and AC_Y <= 352) or
(AC_X >= 337 and AC_X <= 367 and AC_Y >= 322 and AC_Y <= 337) or
(AC_X = 52 and AC_Y >= 367 and AC_Y <= 412) or
(AC_X >= 52 and AC_X <= 397 and AC_Y = 412) or
(AC_X = 142 and AC_Y >= 337 and AC_Y <= 382) or
(AC_X >= 172 and AC_X <= 202 and AC_Y >= 337 and AC_Y <= 352) or
(AC_X >= 187 and AC_X <= 202 and AC_Y >= 337 and AC_Y <= 367) or
(AC_X = 202 and AC_Y >= 337 and AC_Y <= 382) or
(AC_X = 172 and AC_Y >= 382 and AC_Y <= 412) or
(AC_X >= 172 and AC_X <= 187 and AC_Y >= 397 and AC_Y <= 412) or
(AC_X = 232 and AC_Y >= 367 and AC_Y <= 412) or
(AC_X >= 262 and AC_X <= 277 and AC_Y = 367) or
(AC_X = 262 and AC_Y >= 367 and AC_Y <= 397) or
(AC_X >= 262 and AC_X <= 277 and AC_Y >= 397 and AC_Y <= 412) or
(AC_X = 307 and AC_Y >= 337 and AC_Y <= 412) or
(AC_X = 337 and AC_Y >= 367 and AC_Y <= 412) or
(AC_X = 367 and AC_Y >= 367 and AC_Y <= 412) or
(AC_X = 397 and AC_Y >= 367 and AC_Y <= 412)

```

```

) then

```

```

    state <= next_state;

```

```

    passo <= '1';

```

```

elsif ((AC_X1 >= 52 and AC_X1 <= 397 and AC_Y1 = 67) or
(AC_X1 = 52 and AC_Y1 >= 67 and AC_Y1 <= 127) or
(AC_X1 >= 52 and AC_X1 <= 82 and AC_Y1 >= 97 and AC_Y1 <= 127) or
(AC_X1 = 142 and AC_Y1 >= 67 and AC_Y1 <= 97) or
(AC_X1 >= 112 and AC_X1 <= 172 and AC_Y1 = 97) or
(AC_X1 = 112 and AC_Y1 >= 97 and AC_Y1 <= 127) or
(AC_X1 >= 112 and AC_X1 <= 172 and AC_Y1 = 127) or
(AC_X1 = 172 and AC_Y1 >= 97 and AC_Y1 <= 127) or
(AC_X1 = 232 and AC_Y1 >= 67 and AC_Y1 <= 97) or
(AC_X1 >= 202 and AC_X1 <= 232 and AC_Y1 = 97) or
(AC_X1 = 202 and AC_Y1 >= 97 and AC_Y1 <= 157) or
(AC_X1 >= 52 and AC_X1 <= 292 and AC_Y1 = 157) or
(AC_X1 = 232 and AC_Y1 >= 97 and AC_Y1 <= 157) or
(AC_X1 >= 232 and AC_X1 <= 247 and AC_Y1 >= 112 and AC_Y1 <= 127)
or
(AC_X1 >= 232 and AC_X1 <= 292 and AC_Y1 = 127) or
(AC_X1 >= 262 and AC_X1 <= 322 and AC_Y1 = 97) or
(AC_X1 = 322 and AC_Y1 >= 67 and AC_Y1 <= 187) or
(AC_X1 >= 352 and AC_X1 <= 397 and AC_Y1 = 97) or
(AC_X1 >= 352 and AC_X1 <= 397 and AC_Y1 = 127) or
(AC_X1 = 397 and AC_Y1 >= 97 and AC_Y1 <= 307) or
(AC_X1 >= 142 and AC_X1 <= 172 and AC_Y1 >= 127 and AC_Y1 <= 157)
or
(AC_X1 >= 292 and AC_X1 <= 397 and AC_Y1 = 157) or
(AC_X1 >= 307 and AC_X1 <= 337 and AC_Y1 >= 157 and AC_Y1 <= 187)
or
(AC_X1 >= 367 and AC_X1 <= 397 and AC_Y1 >= 157 and AC_Y1 <= 172)
or
(AC_X1 = 367 and AC_Y1 >= 157 and AC_Y1 <= 217) or
(AC_X1 >= 52 and AC_X1 <= 97 and AC_Y1 >= 157 and AC_Y1 <= 172) or
(AC_X1 = 187 and AC_Y1 >= 157 and AC_Y1 <= 217) or
(AC_X1 >= 67 and AC_X1 <= 97 and AC_Y1 = 202) or
(AC_X1 = 97 and AC_Y1 >= 202 and AC_Y1 <= 247) or
(AC_X1 = 52 and AC_Y1 >= 217 and AC_Y1 <= 307) or
(AC_X1 >= 52 and AC_X1 <= 97 and AC_Y1 >= 232 and AC_Y1 <= 247) or
(AC_X1 >= 97 and AC_X1 <= 337 and AC_Y1 = 247) or
(AC_X1 >= 127 and AC_X1 <= 142 and AC_Y1 >= 232 and AC_Y1 <= 247)
or
(AC_X1 = 127 and AC_Y1 >= 187 and AC_Y1 <= 247) or
(AC_X1 >= 127 and AC_X1 <= 157 and AC_Y1 = 187) or
(AC_X1 >= 172 and AC_X1 <= 217 and AC_Y1 >= 202 and AC_Y1 <= 217)
or
(AC_X1 >= 157 and AC_X1 <= 247 and AC_Y1 = 217) or
(AC_X1 = 247 and AC_Y1 >= 187 and AC_Y1 <= 217) or
(AC_X1 >= 217 and AC_X1 <= 247 and AC_Y1 = 187) or
(AC_X1 = 217 and AC_Y1 >= 187 and AC_Y1 <= 217) or
(AC_X1 = 277 and AC_Y1 >= 187 and AC_Y1 <= 217) or
(AC_X1 >= 277 and AC_X1 <= 337 and AC_Y1 = 217) or
(AC_X1 >= 277 and AC_X1 <= 292 and AC_Y1 >= 202 and AC_Y1 <= 217)
or
(AC_X1 >= 337 and AC_X1 <= 412 and AC_Y1 = 247) or
(AC_X1 >= 382 and AC_X1 <= 412 and AC_Y1 >= 232 and AC_Y1 <= 247)
or
(AC_X1 >= 337 and AC_X1 <= 367 and AC_Y1 >= 247 and AC_Y1 <= 262)
or
(AC_X1 = 352 and AC_Y1 >= 247 and AC_Y1 <= 277) or
(AC_X1 >= 82 and AC_X1 <= 112 and AC_Y1 = 277) or
(AC_X1 = 82 and AC_Y1 >= 277 and AC_Y1 <= 382) or
(AC_X1 >= 52 and AC_X1 <= 397 and AC_Y1 = 337) or
(AC_X1 >= 82 and AC_X1 <= 112 and AC_Y1 >= 307 and AC_Y1 <= 382)
or
(AC_X1 = 142 and AC_Y1 >= 247 and AC_Y1 <= 307) or
(AC_X1 >= 172 and AC_X1 <= 232 and AC_Y1 >= 247 and AC_Y1 <= 262)
or
(AC_X1 = 172 and AC_Y1 >= 262 and AC_Y1 <= 307) or
(AC_X1 >= 172 and AC_X1 <= 232 and AC_Y1 = 307) or
(AC_X1 >= 217 and AC_X1 <= 232 and AC_Y1 >= 247 and AC_Y1 <= 307)
or
(AC_X1 = 232 and AC_Y1 >= 247 and AC_Y1 <= 337) or

```

```

        (AC_X1 >= 262 and AC_X1 <= 307 and AC_Y1 = 277) or
        (AC_X1 >= 292 and AC_X1 <= 337 and AC_Y1 = 292) or
        (AC_X1 >= 262 and AC_X1 <= 307 and AC_Y1 = 307) or
        (AC_X1 = 307 and AC_Y1 >= 247 and AC_Y1 <= 307) or
        (AC_X1 >= 292 and AC_X1 <= 307 and AC_Y1 >= 277 and AC_Y1 <= 307)
or
        (AC_X1 >= 367 and AC_X1 <= 397 and AC_Y1 = 292) or
        (AC_X1 = 352 and AC_Y1 >= 307 and AC_Y1 <= 352) or
        (AC_X1 >= 337 and AC_X1 <= 367 and AC_Y1 >= 322 and AC_Y1 <= 337)
or
        (AC_X1 = 52 and AC_Y1 >= 367 and AC_Y1 <= 412) or
        (AC_X1 >= 52 and AC_X1 <= 397 and AC_Y1 = 412) or
        (AC_X1 = 142 and AC_Y1 >= 337 and AC_Y1 <= 382) or
        (AC_X1 >= 172 and AC_X1 <= 202 and AC_Y1 >= 337 and AC_Y1 <= 352)
or
        (AC_X1 >= 187 and AC_X1 <= 202 and AC_Y1 >= 337 and AC_Y1 <= 367)
or
        (AC_X1 = 202 and AC_Y1 >= 337 and AC_Y1 <= 382) or
        (AC_X1 = 172 and AC_Y1 >= 382 and AC_Y1 <= 412) or
        (AC_X1 >= 172 and AC_X1 <= 187 and AC_Y1 >= 397 and AC_Y1 <= 412)
or
        (AC_X1 = 232 and AC_Y1 >= 367 and AC_Y1 <= 412) or
        (AC_X1 >= 262 and AC_X1 <= 277 and AC_Y1 = 367) or
        (AC_X1 = 262 and AC_Y1 >= 367 and AC_Y1 <= 397) or
        (AC_X1 >= 262 and AC_X1 <= 277 and AC_Y1 >= 397 and AC_Y1 <= 412)
or
        (AC_X1 = 307 and AC_Y1 >= 337 and AC_Y1 <= 412) or
        (AC_X1 = 337 and AC_Y1 >= 367 and AC_Y1 <= 412) or
        (AC_X1 = 367 and AC_Y1 >= 367 and AC_Y1 <= 412) or
        (AC_X1 = 397 and AC_Y1 >= 367 and AC_Y1 <= 412)
    ) then
        state <= next_state1;
    else
        state <= stop;
        C1 <= LC1;
        R1 <= LR1;
        passa <= '1';
    end if;
    when right_u =>
        if (passo = '1') then
            C1 <= LC1;
            R1 <= LR1;
        end if;
        passo <= '0';
        C1 <= C1 + 1;
        R1 <= R1;
        AC_Xn <= C1 + 7;
        AC_Yn <= R1 + 7;
        corrimiento <= "000000000000";
        if (botones = "1000" or botones = "0100" or
            botones = "0010" or botones = "0001") then
            direccion <= botones;
        else
            direccion <= "0100";
        end if;
        direccion_actual <= "0100";
        last_state <= right_u;
        LC1 <= C1;
        LR1 <= R1;
        if (articale_state = '1') then
            state <= playerdead;
        else
            state <= check;
        end if;
    end when;
end process;

```

```

        end if;
when right_d =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;

    end if;
    passo <= '0';
    C1 <= C1 + 1;
    R1 <= R1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "01110000100";
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001") then
        direccion <= botones;

    else
        direccion <= "0100";

    end if;
    direccion_actual <= "0100";
    last_state <= right_d;
    LC1 <= C1;
    LR1 <= R1;
    if (artical_state = '1') then
        state <= playerdead;

    else
        state <= check;

    end if;
when left_u =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;

    end if;
    passo <= '0';
    C1 <= C1 - 1;
    R1 <= R1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "00011100001";
    state <= check;
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001")

        direccion <= botones;

    else
        direccion <= "0010";

    end if;
    direccion_actual <= "0010";
    last_state <= left_u;
    LC1 <= C1;
    LR1 <= R1;
    if (artical_state = '1') then
        state <= playerdead;

    else
        state <= check;

    end if;
when left_d =>
    if (passo = '1') then

```

then

```

        C1 <= LC1;
        R1 <= LR1;
    end if;
    passo <= '0';
    C1 <= C1 - 1;
    R1 <= R1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "10001100101";
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001") then
        direccion <= botones;
    else
        direccion <= "0010";
    end if;
    direccion_actual <= "0010";
    last_state <= left_d;
    LC1 <= C1;
    LR1 <= R1;
    if (articale_state = '1') then
        state <= playerdead;
    else
        state <= check;
    end if;
when up_u =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;
    end if;
    passo <= '0';
    C1 <= C1;
    R1 <= R1 - 1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "01010100011";
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001") then
        direccion <= botones;
    else
        direccion <= "0001";
    end if;
    direccion_actual <= "0001";
    passo <= '0';
    last_state <= up_u;
    LC1 <= C1;
    LR1 <= R1;
    if (articale_state = '1') then
        state <= playerdead;
    else
        state <= check;
    end if;
when up_d =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;
    end if;
    passo <= '0';

```

```

C1 <= C1;
R1 <= R1 - 1;
AC_Xn <= C1 + 7;
AC_Yn <= R1 + 7;
corrimiento <= "11000100111";
if (botones = "1000" or botones = "0100" or
    botones = "0010" or botones = "0001") then
    direccion <= botones;
else
    direccion <= "0001";
end if;
direccion_actual <= "0001";
last_state <= up_d;
LC1 <= C1;
LR1 <= R1;
if (artical_state = '1') then
    state <= playerdead;
else
    state <= check;
end if;
when down_u =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;
    end if;
    passo <= '0';
    C1 <= C1;
    R1 <= R1 + 1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "00111000010";
    if (botones = "1000" or botones = "0100" or
        botones = "0010" or botones = "0001") then
        direccion <= botones;
    else
        direccion <= "1000";
    end if;
    direccion_actual <= "1000";
    last_state <= down_u;
    LC1 <= C1;
    LR1 <= R1;
    if (artical_state = '1') then
        state <= playerdead;
    else
        state <= check;
    end if;
when down_d =>
    if (passo = '1') then
        C1 <= LC1;
        R1 <= LR1;
    end if;
    passo <= '0';
    C1 <= C1;
    R1 <= R1 + 1;
    AC_Xn <= C1 + 7;
    AC_Yn <= R1 + 7;
    corrimiento <= "10101000110";

```

```

        if (botones = "1000" or botones = "0100" or
            botones = "0010" or botones = "0001") then
            direccion <= botones;
        else
            direccion <= "1000";
        end if;
        direccion_actual <= "1000";
        last_state <= down_d;
        LC1 <= C1;
        LR1 <= R1;
        if (artical_state = '1') then
            state <= playerdead;
        else
            state <= check;
        end if;
    when playerdead =>
        if (life <= "10") then
            life <= life + 1;
            C1 <= "0010010110";
            R1 <= "0011110000";
            LC1 <= "0010010110";
            LR1 <= "0011110000";
            corrimiento <= "00111000010";
            direccion <= "0100";
            AC_Xn <= C1 + 7;
            AC_Yn <= R1 + 7;
            player1_dead <= '0';
            state <= stop;
        else
            state <= gameover;
        end if;
    when gameover =>
        state <= gameover;
        player1_dead <= '1';
    end case;
end if;
end process;
end Behavioral;

```


Apéndice 5.- Código del sistema - Acceso a la ROM de Articale.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library UNISIM;
use UNISIM.VCOMPONENTS.all;
entity ArticaleROM_wrapper is
    port (
        M : out STD_LOGIC_VECTOR (11 downto 0 );
        clka : in STD_LOGIC;
        rom_addr16 : in STD_LOGIC_VECTOR (10 downto 0 )
    );
end ArticaleROM_wrapper;
architecture STRUCTURE of ArticaleROM_wrapper is
    component ArticaleROM is
        port (
            rom_addr16 : in STD_LOGIC_VECTOR (10 downto 0 );
            clka : in STD_LOGIC;
            M : out STD_LOGIC_VECTOR (11 downto 0 )
        );
    end component ArticaleROM;
begin
    ArticaleROM_i : component ArticaleROM
    port map(
        M(11 downto 0) => M(11 downto 0),
        clka => clka,
        rom_addr16(10 downto 0) => rom_addr16(10 downto 0)
    );
end STRUCTURE;
```

Apéndice 6.- Código del sistema - Control VGA.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ControlVGA is
    port (
        ArticON : in STD_LOGIC;
        MazeON : in STD_LOGIC;
        LlaveON : in STD_LOGIC;
        RoyON : in STD_LOGIC;
        ScoreON1, ScoreON2, lives12ON : in STD_LOGIC;
        TimerON : in STD_LOGIC;
        redA : in STD_LOGIC_VECTOR(3 downto 0);
        greenA : in STD_LOGIC_VECTOR(3 downto 0);
        blueA : in STD_LOGIC_VECTOR(3 downto 0);
        redM : in STD_LOGIC_VECTOR(3 downto 0);
        greenM : in STD_LOGIC_VECTOR(3 downto 0);
        blueM : in STD_LOGIC_VECTOR(3 downto 0);
        redL : in STD_LOGIC_VECTOR(3 downto 0);
        greenL : in STD_LOGIC_VECTOR(3 downto 0);
        blueL : in STD_LOGIC_VECTOR(3 downto 0);
        redR : in STD_LOGIC_VECTOR(3 downto 0);
        greenR : in STD_LOGIC_VECTOR(3 downto 0);
        blueR : in STD_LOGIC_VECTOR(3 downto 0);
        redSc1, redSc2 : in STD_LOGIC_VECTOR(3 downto 0);
        greenSc1, greenSc2 : in STD_LOGIC_VECTOR(3 downto 0);
        blueSc1, blueSc2 : in STD_LOGIC_VECTOR(3 downto 0);
        redT : in STD_LOGIC_VECTOR(3 downto 0);
        greenT : in STD_LOGIC_VECTOR(3 downto 0);
        blueT : in STD_LOGIC_VECTOR(3 downto 0);
        redT12 : in STD_LOGIC_VECTOR(3 downto 0);
        greenT12 : in STD_LOGIC_VECTOR(3 downto 0);
        blueT12 : in STD_LOGIC_VECTOR(3 downto 0);
        imageON : in std_logic;
        gameoverON : in std_logic;
        redI : in std_logic_vector(3 downto 0);
        blueI : in std_logic_vector(3 downto 0);
        greenI : in std_logic_vector(3 downto 0);
        redGO : in std_logic_vector(3 downto 0);
        blueGO : in std_logic_vector(3 downto 0);
        greenGO : in std_logic_vector(3 downto 0);
        Enemy1ON : in std_logic;
        redE1 : in std_logic_vector(3 downto 0);
        blueE1 : in std_logic_vector(3 downto 0);
        greenE1 : in std_logic_vector(3 downto 0);
        Enemy2ON : in std_logic;
        redE2 : in std_logic_vector(3 downto 0);
        blueE2 : in std_logic_vector(3 downto 0);
        greenE2 : in std_logic_vector(3 downto 0);
        Enemy3ON : in std_logic;
        redE3 : in std_logic_vector(3 downto 0);
        blueE3 : in std_logic_vector(3 downto 0);
        greenE3 : in std_logic_vector(3 downto 0);
        redS : out STD_LOGIC_VECTOR(3 downto 0);
        greenS : out STD_LOGIC_VECTOR(3 downto 0);
        blueS : out STD_LOGIC_VECTOR(3 downto 0)
    );
```

```

end ControlVGA;
architecture Behavioral of ControlVGA is
begin
    MuxVGA : process (ArticON, MazeON, LLaveON, RoyON, TimerON, lives12ON,
        ScoreON1, ScoreON2, redA, greenA, blueA, redM,
        greenM, blueM, redL, greenL, blueL,
        redR, greenR, blueR, redSc1, greenSc1, blueSc1,
        redSc2, greenSc2, blueSc2, redT, greenT, blueT,
        imageON, redI, greenI, blueI, gameoverON, redGO, greenGO, blueGO,
        redT12, greenT12, blueT12, Enemy1ON, redE1, greenE1, blueE1,
        Enemy2ON, redE2, greenE2, blueE2, Enemy3ON, redE3, greenE3, blueE3 )
    begin
        if (imageON = '1' ) then
            redS <= redI;
            greenS <= greenI;
            blueS <= blueI;
        elsif (gameoverON = '1') then
            redS <= redGO;
            greenS <= greenGO;
            blueS <= blueGO;
        elsif (ArticON = '1') then
            redS <= redA;
            greenS <= greenA;
            blueS <= blueA;
        elsif (RoyON = '1') then
            redS <= redR;
            greenS <= greenR;
            blueS <= blueR;
        elsif (LLaveON = '1') then
            redS <= redL;
            greenS <= greenL;
            blueS <= blueL;
        elsif (Enemy1ON = '1') then
            redS <= redE1;
            greenS <= greenE1;
            blueS <= blueE1;
        elsif (Enemy3ON = '1') then
            redS <= redE3;
            greenS <= greenE3;
            blueS <= blueE3;
        elsif (Enemy2ON = '1') then
            redS <= redE2;
            greenS <= greenE2;
            blueS <= blueE2;
        elsif (MazeON = '1') then
            redS <= redM;
            greenS <= greenM;
            blueS <= blueM;
        elsif (ScoreON1 = '1') then
            redS <= redSc1;
            greenS <= greenSc1;
            blueS <= blueSc1;
        elsif (ScoreON2 = '1') then
            redS <= redSc2;
            greenS <= greenSc2;
            blueS <= blueSc2;
        elsif (TimerON = '1') then
    
```

```
        redS <= redT;
        greenS <= greenT;
        blueS <= blueT;
    elsif (lives12ON = '1') then
        redS <= redT12;
        greenS <= greenT12;
        blueS <= blueT12;
    else
        redS <= "0000";
        greenS <= "0000";
        blueS <= "0000";
    end if;
end process MuxVGA;
end Behavioral;
```

Apéndice 7.- Código del sistema - Núcleo del sistema.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity gamecore is
    port (
        reset : in std_logic;
        master_clk : in std_logic;
        start_game : in std_logic;
        end_time : in std_logic;
        all_keys : in std_logic;
        player1_dead : in std_logic;
        player2_dead : in std_logic;
        level_state : out std_logic_vector(2 downto 0);
        end_level : out std_logic;
        win_game : out std_logic;
        end_game : out std_logic
    );
end gamecore;
architecture Behavioral of gamecore is
    type state_type is (intro, start, levelone, leveltwo, levelthree, levelfour,
        levelfive, wingame, gameover);
    signal state : state_type;
begin
    process (reset, master_clk, start_game )
    begin
        if reset = '1' then
            state <= intro;
            end_level <= '0';
            end_game <= '0';
            win_game <= '0';
            level_state <= "000";
        elsif (rising_edge(master_clk)) then
            case state is
                when intro =>
                    if start_game = '1' then
                        state <= start;
                    else
                        state <= intro;
                        end_level <= '0';
                        end_game <= '0';
                        win_game <= '0';
                        level_state <= "000";
                    end if;
                when start =>
                    end_level <= '0';
                    end_game <= '0';
                    win_game <= '0';
                    state <= levelone;
                when levelone =>
                    if (all_keys = '1' and end_time = '0') then
                        end_level <= '1';
                        end_game <= '0';
                        win_game <= '0';
                        state <= leveltwo;
                    elsif (end_time = '1' or (player1_dead = '1' or
                        player2_dead = '1')) then
```

```

        end_level <= '0';
        end_game <= '1';
        win_game <= '0';
        state <= gameover;
    else
        end_level <= '0';
        end_game <= '0';
        win_game <= '0';
        state <= levelone;
        level_state <= "001";
    end if;
when leveltwo =>
    if (all_keys = '1' and end_time = '0') then
        end_level <= '1';
        end_game <= '0';
        win_game <= '0';
        state <= levelthree;
    elsif (end_time = '1' or (player1_dead = '1' or
        player2_dead = '1')) then
        end_level <= '0';
        win_game <= '0';
        end_game <= '1';
        state <= gameover;
    else
        end_level <= '0';
        end_game <= '0';
        win_game <= '0';
        state <= leveltwo;
        level_state <= "010";
    end if;
when levelthree =>
    if (all_keys = '1' and end_time = '0') then
        end_level <= '1';
        end_game <= '0';
        win_game <= '0';
        state <= levelfour;
    elsif (end_time = '1' or (player1_dead = '1' or
        player2_dead = '1')) then
        end_level <= '0';
        end_game <= '1';
        win_game <= '0';
        state <= gameover;
    else
        end_level <= '0';
        end_game <= '0';
        win_game <= '0';
        state <= levelthree;
        level_state <= "011";
    end if;
when levelfour =>
    if (all_keys = '1' and end_time = '0') then
        end_level <= '1';
        end_game <= '0';
        win_game <= '0';
        state <= levelfive;
    elsif (end_time = '1' or (player1_dead = '1' or
        player2_dead = '1')) then

```

```

        end_level <= '0';
        end_game <= '1';
        win_game <= '0';
        state <= gameover;
    else
        end_level <= '0';
        end_game <= '0';
        win_game <= '0';
        state <= levelfour;
        level_state <= "100";
    end if;
when levelfive =>
    if (all_keys = '1' and end_time = '0') then
        end_level <= '1';
        end_game <= '0';
        state <= wingame;
        win_game <= '1';
    elsif (end_time = '1' or (player1_dead = '1' or
        player2_dead = '1')) then
        end_level <= '0';
        end_game <= '1';
        win_game <= '0';
        state <= gameover;
    else
        end_level <= '0';
        end_game <= '0';
        win_game <= '0';
        state <= levelfive;
        level_state <= "101";
    end if;
when wingame =>
    end_level <= '0';
    end_game <= '0';
    win_game <= '1';
    state <= wingame;
    level_state <= "110";
when gameover =>
    end_level <= '0';
    win_game <= '0';
    end_game <= '1';
    state <= gameover;
    level_state <= "111";
end case;
end if;
end process;
end Behavioral;

```

Apéndice 8.- Código del sistema - Controlador del Audio.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.STD_LOGIC_unsigned.all;
entity Audio is
    port (
        Master_clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        LRCLK : out STD_LOGIC;
        BCLK : out STD_LOGIC;
        MCLK : out STD_LOGIC;
        shutdown : out STD_LOGIC;
        sampledata : in STD_LOGIC_VECTOR (7 downto 0 );
        samplememory : out STD_LOGIC_VECTOR (15 downto 0 );
        gotkey : in STD_LOGIC;
        endkey : out STD_LOGIC;
        ksampledata : in STD_LOGIC_VECTOR (7 downto 0 );
        ksamplmemory : out STD_LOGIC_VECTOR (14 downto 0 );
        SampleOut : out STD_LOGIC
    );
end Audio;
architecture Behavioral of Audio is
    signal temp : STD_LOGIC;
    signal temp1 : STD_LOGIC;
    signal temp2 : STD_LOGIC;
    signal counter : integer range 0 to 2 := 0;
    signal counter1 : integer range 0 to 11 := 0;
    signal counter2 : integer range 0 to 7 := 0;
    type state_type is (start, shift, delay, kstart, kshift, kdelay);
    signal state : state_type;
    signal samplebuff : STD_LOGIC_VECTOR (15 downto 0);
    signal bit_count : STD_LOGIC_VECTOR (3 downto 0);
    signal lrclk_last : STD_LOGIC;
    signal bclk_last : STD_LOGIC;
    signal sampleleft : STD_LOGIC_VECTOR(7 downto 0);
    signal sampleright : STD_LOGIC_VECTOR(7 downto 0);
    signal samplememoryint : STD_LOGIC_VECTOR (15 downto 0);
    signal kbit_count : STD_LOGIC_VECTOR (3 downto 0);
    signal ksamplmemoryint : STD_LOGIC_VECTOR (14 downto 0);
begin
    process (Master_clk, clr)
    begin
        if (clr = '1') then
            temp <= '0';
            counter <= 0;
        elsif (rising_edge(Master_clk)) then
            if (counter = 2) then
                temp <= not(temp);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
    MCLK <= temp;
    process (temp, clr)
```



```

begin
    if (clr = '1') then
        temp1 <= '0';
        counter1 <= 0;
    elsif (rising_edge(temp)) then
        if (counter1 = 11) then
            temp1 <= not(temp1);
            counter1 <= 0;
        else
            counter1 <= counter1 + 1;
        end if;
    end if;
end process;
BCLK <= temp1;
process (temp1, clr)
begin
    if (clr = '1') then
        temp2 <= '0';
        counter2 <= 0;
    elsif (falling_edge(temp1)) then
        if (counter2 = 7) then
            temp2 <= not(temp2);
            counter2 <= 0;
        else
            counter2 <= counter2 + 1;
        end if;
    end if;
end process;
LRCLK <= temp2;

leftjustified : process (Master_clk, clr, gotkey)
begin
    if clr = '1' then
        state <= start;
        bit_count <= "0000";
        lrclk_last <= '0';
        bclk_last <= '0';
        samplememoryint <= "0000000000000000";
        shutdown <= '0';
        endkey <= '0';
        kbit_count <= "0000";
        ksamplmemoryint <= "0000000000000000";
    elsif (rising_edge(Master_clk)) then
        case state is
            when start =>
                if ((lrclk_last = '0' and temp2 = '1') and
                    (bclk_last = '1' and temp1 = '0') ) then
                    samplebuff <= sampleleft &
                                sampleright;
                    shutdown <= '1';
                    state <= shift;
                else
                    state <= start;
                end if;
            endkey <= '0';
            lrclk_last <= temp2;
            bclk_last <= temp1;
        end case;
    end if;
end process;

```

```

when shift =>
    SampleOut <= samplebuff(15);
    samplebuff(15 downto 1) <=
        samplebuff(14 downto 0);
    bit_count <= bit_count + 1;
    state <= delay;
    if (bit_count = "1111") then
        bit_count <= "0000";
        samplememoryint <=
            samplememoryint + 1;
        if (samplememoryint =
            "1100100001111111") then
            samplebuff <= sampleleft &
                sampleright;
            samplememoryint <=
                "0000000000000000";
        else
            samplebuff <= sampleleft &
                sampleright;
        end if;
        if (gotkey = '1') then
            state <= kstart;
        end if;
    end if;
    lrclk_last <= temp2;
    bclk_last <= temp1;
when delay =>
    if ((bclk_last = '1' and temp1 = '0')) then
        state <= shift;
    else
        state <= delay;
    end if;
    lrclk_last <= temp2;
    bclk_last <= temp1;
when kstart =>
    if ((lrclk_last = '0' and temp2 = '1') and
        (bclk_last = '1' and temp1 = '0')) then
        samplebuff <= sampleleft &
            sampleright;
        state <= kshift;
    else
        state <= kstart;
    end if;
    lrclk_last <= temp2;
    bclk_last <= temp1;
when kshift =>
    SampleOut <= samplebuff(15);
    samplebuff(15 downto 1) <=
        samplebuff(14 downto 0);
    kbit_count <= kbit_count + 1;
    state <= kdelay;
    if (kbit_count = "1111") then
        kbit_count <= "0000";
        ksamplmemoryint <=
            ksamplmemoryint + 1;
        if (ksamplmemoryint =
            "100101011110101") then

```

```

samplebuff <= sampleleft &
    sampleright;
ksamplememoryint <=
    "0000000000000000";
endkey <= '1';
state <= start;

else
    samplebuff <= sampleleft &
        sampleright;

end if;

end if;
lrclk_last <= temp2;
bclk_last <= temp1;
when kdelay =>
    if ((bclk_last = '1' and temp1 = '0')) then
        state <= kshift;

    else
        state <= kdelay;

    end if;
    lrclk_last <= temp2;
    bclk_last <= temp1;

end case;

end if;
end process leftjustified;
process (clr)
begin
    if (clr = '1') then
        samplememory <= "0000000000000000";
        sampleleft <= "00000000";
        sampleright <= "00000000";
        ksamplememory <= "0000000000000000";
    elsif (gotkey = '1') then
        ksamplememory <= ksamplememoryint;
        sampleleft <= ksampladata;
        sampleright <= ksampladata;

    else
        samplememory <= samplememoryint;
        sampleleft <= sampladata;
        sampleright <= sampladata;

    end if;
end process;
end Behavioral;

```

Apéndice 9.- Código del sistema - Controlador de los mandos.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.STD_LOGIC_unsigned.all;
entity Controller is
    port (
        Master_clk : in STD_LOGIC;
        clr : in STD_LOGIC;
        cData : in STD_LOGIC;
        cLatch : out STD_LOGIC;
        cClock : out STD_LOGIC;
        direccion : out STD_LOGIC_VECTOR (3 downto 0);
        selection : out STD_LOGIC;
        start : out STD_LOGIC;
        Zbotton : out STD_LOGIC;
        Ybotton : out STD_LOGIC;
        cData1 : in STD_LOGIC;
        cLatch1 : out STD_LOGIC;
        cClock1 : out STD_LOGIC;
        direccion1 : out STD_LOGIC_VECTOR (3 downto 0);
        selection1 : out STD_LOGIC;
        start1 : out STD_LOGIC;
        Zbotton1 : out STD_LOGIC;
        Ybotton1 : out STD_LOGIC
    );
end Controller;
architecture Behavioral of Controller is
    type state_type is (latch, readY, readandshift);
    signal state : state_type;
    signal timecounter : STD_LOGIC_VECTOR (3 downto 0 );
    signal bitcounter : STD_LOGIC_VECTOR (2 downto 0);
    signal tempDataOut : STD_LOGIC_VECTOR (7 downto 0);
    signal tempData : STD_LOGIC_VECTOR (7 downto 0);
    type state_typed1 is (latch1, readY1, readandshift1);
    signal statel : state_typed1;
    signal timecounter1 : STD_LOGIC_VECTOR (3 downto 0 );
    signal bitcounter1 : STD_LOGIC_VECTOR (2 downto 0);
    signal tempDataOut1 : STD_LOGIC_VECTOR (7 downto 0);
    signal tempData1 : STD_LOGIC_VECTOR (7 downto 0);
    signal temp : STD_LOGIC;
    signal counter : integer range 0 to 49 := 0;
begin
    process (Master_clk, clr)
    begin
        if (clr = '1') then
            temp <= '0';
            counter <= 0;
        elsif (rising_edge(Master_clk)) then
            if (counter = 49) then
                temp <= not(temp);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;
end process;
```

```

ReadControl : process (temp, clr, cData)
begin
    if clr = '1' then
        state <= latch;
        timecounter <= "0000";
        bitcounter <= "000";
        tempDataOut <= "11111111";
        tempData <= "11111111";
        cClock <= '0';
        cLatch <= '0';
    elsif (rising_edge(temp)) then
        case state is
            when latch =>
                cLatch <= '1';
                if (timecounter <= 11) then
                    timecounter <= timecounter + 1;
                    state <= latch;
                else
                    cLatch <= '0';
                    timecounter <= "0000";
                    state <= ready;
                end if;
            when ready =>
                tempDataOut (0) <= cData;
                if (timecounter <= 3) then
                    timecounter <= timecounter + 1;
                    state <= ready;
                else
                    timecounter <= "0000";
                    tempDataOut (7 downto 1) <=
                        tempDataOut (6 downto 0);
                    state <= readandshift;
                end if;
            when readandshift =>
                if (timecounter <= 10) then
                    timecounter <= timecounter + 1;
                    if (timecounter <= 5) then
                        cClock <= '1';
                    else
                        cClock <= '0';
                    end if;
                    if (timecounter = 5) then
                        tempDataOut (0) <= cData;
                        bitcounter <= bitcounter +
1;
                    end if;
                else
                    timecounter <= "0000";
                    if (bitcounter <= 6) then
                        tempDataOut (7 downto 1) <=
                            tempDataOut (6 downto 0);
                        state <= readandshift;
                    else
                        bitcounter <= "000";
                        tempdata (7 downto 0) <=
                            tempDataOut (7 downto 0);
                        state <= Latch;
                    end if;
                end if;
            end if;
        end case;
    end if;
end process;

```

```

end if;
end if;
end case;
end if;
end process ReadControl;
ReadControl1 : process (temp, clr, cData1)
begin
    if clr = '1' then
        state1 <= latch1;
        timecounter1 <= "0000";
        bitcounter1 <= "000";
        tempDataOut1 <= "11111111";
        tempData1 <= "11111111";
        cClock1 <= '0';
        cLatch1 <= '0';
    elsif (rising_edge(temp)) then
        case state1 is
            when latch1 =>
                cLatch1 <= '1';
                if (timecounter1 <= 11) then
                    timecounter1 <= timecounter1 + 1;
                    state1 <= latch1;
                else
                    cLatch1 <= '0';
                    timecounter1 <= "0000";
                    state1 <= readY1;
                end if;
            when readY1 =>
                tempDataOut1 (0) <= cData1;
                if (timecounter1 <= 3) then
                    timecounter1 <= timecounter1 + 1;
                    state1 <= readY1;
                else
                    timecounter1 <= "0000";
                    tempDataOut1 (7 downto 1) <=
                    tempDataOut1 (6 downto 0);
                    state1 <= readandshift1;
                end if;
            when readandshift1 =>
                if (timecounter1 <= 10) then
                    timecounter1 <= timecounter1 + 1;
                    if (timecounter1 <= 5) then
                        cClock1 <= '1';
                    else
                        cClock1 <= '0';
                    end if;
                    if (timecounter1 = 5) then
                        tempDataOut1 (0) <= cData1;
                        bitcounter1 <= bitcounter1 +
1;
                    end if;
                else
                    timecounter1 <= "0000";
                    if (bitcounter1 <= 6) then
                        tempDataOut1 (7 downto 1) <=
                        tempDataOut1 (6 downto 0);
                        state1 <= readandshift1;
                    end if;
                end if;
            end if;
        end case;
    end if;
end process ReadControl1;

```

```

else
    bitcounter1 <= "000";
    tempdata1 (7 downto 0) <=
    tempDataOut1 (7 downto 0);
    state1 <= Latch1;
end if;
end if;
end case;
end if;
end process ReadControl1;
start <= not(tempData(4));
direccion(0) <= not(tempData(3));
direccion(1) <= not(tempData(1));
direccion(2) <= not(tempData(0));
direccion(3) <= not(tempData(2));
start1 <= not(tempData1(4));
direccion1(0) <= not(tempData1(3));
direccion1(1) <= not(tempData1(1));
direccion1(2) <= not(tempData1(0));
direccion1(3) <= not(tempData1(2));
end Behavioral;

```

BIBLIOGRAFÍA

- [1] «Use of FPGAs in a Digital System Design Course with Computer Gaming Applications: American Society for Engineering Education». [En línea]. Disponible en: <https://www.asee.org/public/conferences/106/papers/21195/view>. [Accedido: 05-nov-2019].
- [2] F. S. Espinosa y F. Trujillo-Romero, «Procesamiento de imágenes en FPGA con visualización en una pantalla VGA», n.º 108, p. 23.
- [3] «HC230F1020AJ de Intel | ASIC», *Arrow.com*. [En línea]. Disponible en: <http://www.arrow.com/es-mx/products/hc230f1020aj/intel>. [Accedido: 15-feb-2020].
- [4] «DM164137 Microchip Technology | Placas de desarrollo, kits, programadores | DigiKey». [En línea]. Disponible en: <https://www.digikey.com.mx/product-detail/es/microchip-technology/DM164137/DM164137-ND/5252514>. [Accedido: 15-feb-2020].
- [5] «Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users», *Digilent*. [En línea]. Disponible en: <https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/>. [Accedido: 15-feb-2020].
- [6] A. van der Ploeg, «Why use an FPGA instead of a CPU or GPU?», *Medium*, 14-ago-2018. [En línea]. Disponible en: <https://blog.esciencecenter.nl/why-use-an-fpga-instead-of-a-cpu-or-gpu-b234cd4f309c>. [Accedido: 07-nov-2019].
- [7] «Intel Stratix 10 DX Device Overview». [En línea]. Disponible en: <https://www.intel.com/content/www/us/en/programmable/documentation/sfx1556814887907.html>. [Accedido: 07-nov-2019].
- [8] «Intel Completes Acquisition of Altera», *Intel Newsroom*. [En línea]. Disponible en: <https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/>. [Accedido: 07-nov-2019].
- [9] «Project Catapult», *Microsoft Research*. [En línea]. Disponible en: <https://www.microsoft.com/en-us/research/project/project-catapult/>. [Accedido: 17-dic-2019].
- [10] A. Cuevas Velázquez, B. J. Godínez Salinas, y A. D. Silva Simón, «Diseño de Prácticas de Laboratorio para el área de Sistemas Digitales», *Sistemas Digitales*, UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO, MÉXICO, CIUDAD UNIVERSITARIA, 2012.
- [11] Ukranio Coronilla Contreras, «Procesamiento digital de video en tiempo real y “video wall” con la PC».
- [12] E. A. Perdomo Hourné, L. M. Garcés-Socarrás, y A. J. Cabrera Sarmiento, «Diseño de bloques para el procesado de imágenes en lenguaje de descripción de hardware», *Ing. Electrónica Automática Comun.*, vol. 34, n.º 2, pp. 9-18, ago. 2013.
- [13] «Historia de los videojuegos». [En línea]. Disponible en: <https://www.fib.upc.edu/retro-informatica/historia/videojocs.html>. [Accedido: 18-mar-2019].
- [14] Strafefox, *How video games were made - part 1: Graphics*. .

- [15] E. Crabill, «FPGA-based Video Games», p. 3, 2005.
- [16] A. Perez-Poch, «Videojuegos Retro: creatividad e ingenio para crear una consola desde cero», vol. 7, p. 11.
- [17] «VGA Signal 640 x 480 @ 60 Hz Industry standard timing». [En línea]. Disponible en: <http://www.tinyvga.com/vga-timing/640x480@60Hz>. [Accedido: 20-may-2019].
- [18] «Basys 3 Reference Manual [Reference.Digilentinc]». [En línea]. Disponible en: <https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual>. [Accedido: 16-mar-2019].
- [19] Philips Semiconductors, «I2SBUS.pdf». [En línea]. Disponible en: <https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf>. [Accedido: 18-jun-2019].
- [20] «Interfacing an I2S Device to an MSP430 Device», p. 7, 2010.
- [21] «The NES Controller Handler». [En línea]. Disponible en: <https://tresi.github.io/nes/>. [Accedido: 21-nov-2019].
- [22] «Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users», *Digilent*. [En línea]. Disponible en: <https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/>. [Accedido: 05-jun-2019].
- [23] «Pmod AMP3 Reference Manual [Reference.Digilentinc]». [En línea]. Disponible en: <https://reference.digilentinc.com/reference/pmod/pmodamp3/reference-manual>. [Accedido: 10-dic-2019].
- [24] «BeepBox». [En línea]. Disponible en: https://www.beepbox.co/#8n31s0k0l00e03t2mm0a7g0fj07i0r1o3210T1v1L4u82q1d1f6y1z2C0c1AcFfB7ViQ0245P7788E0000T0v1L4u58q1d1f8y6z7C0w5c0h1T5v1L4u33q1d5f7y1z6C1c0h0HYw0040199b9999T4v1L4uf0q1z6666ji8k8k3jSBKSJJAArriiiii07JCABrzrrrrrr00YrkqHrsrrrrjr005zrAqzrjzrrqr1jRjrjGGrrzsrsA099ijrABJJIAzrrtirqrjqixzsrAjrqiqaqqysttAJqjikirizrHtBJJAzArzrIsRCITKSS099ijrAJS____Qg99habbCAYrDzh00b4h40000000h4g000000014h000000004h400000000p16000000. [Accedido: 30-dic-2019].

