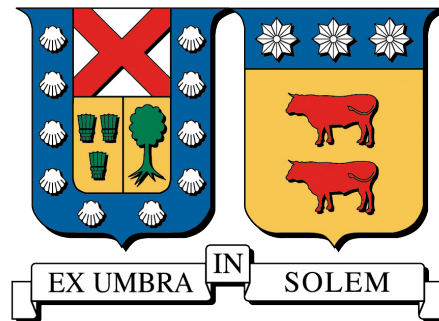


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA  
DEPARTAMENTO DE ELECTRÓNICA  
VALPARAÍSO - CHILE



**DISEÑO DE EXPERIENCIAS PARA ENSEÑANZA DE DISEÑO DE  
SISTEMAS DIGITALES CON UN ENFOQUE EXPERIMENTAL  
UTILIZANDO FPGA**

**TOMÁS ANTONIO VELÁSQUEZ LEE**

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL ELECTRÓNICO**

PROFESOR GUÍA : GONZALO CARVAJAL  
PROFESOR CORREFERENTE : ANGEL CEDEÑO  
PROFESOR CORREFERENTE : MAURICIO SOLÍS

NOVIEMBRE 2023

# Índice general

|                                                                                      |           |
|--------------------------------------------------------------------------------------|-----------|
| <b>1. Introducción</b>                                                               | <b>5</b>  |
| 1.1. Motivación y contexto . . . . .                                                 | 5         |
| 1.2. Planteamiento del problema . . . . .                                            | 9         |
| 1.3. Alcances y contribuciones . . . . .                                             | 10        |
| 1.4. Organización del informe . . . . .                                              | 11        |
| <b>2. Antecedentes</b>                                                               | <b>12</b> |
| 2.1. Estudio de cursos similares . . . . .                                           | 12        |
| 2.1.1. Cursos revisados . . . . .                                                    | 12        |
| 2.1.2. Estado de ELO212 en relación a cursos alternativos revisados. . . . .         | 14        |
| 2.2. Scripts de revisión asistida . . . . .                                          | 15        |
| 2.3. Herramientas de diseño alternativas a Vivado . . . . .                          | 15        |
| 2.3.1. IcarusVerilog [1] . . . . .                                                   | 16        |
| 2.3.2. Visual Studio Code [2] . . . . .                                              | 17        |
| 2.3.3. TerosHDL [3] . . . . .                                                        | 17        |
| 2.3.4. Simulador VGA . . . . .                                                       | 18        |
| <b>3. Diseño de nuevo material</b>                                                   | <b>19</b> |
| 3.1. <i>Scripts</i> de revisión asistida . . . . .                                   | 19        |
| 3.2. Nuevas herramientas para diseño y análisis funcional. . . . .                   | 20        |
| 3.2.1. Ventajas y desventajas del flujo de trabajo . . . . .                         | 20        |
| 3.3. Actividades con interfaz VGA para procesamiento de imágenes . . . . .           | 22        |
| 3.3.1. Actividad base . . . . .                                                      | 23        |
| 3.3.2. Definición de nuevas actividades . . . . .                                    | 25        |
| 3.3.3. Introducción a interfaz VGA . . . . .                                         | 27        |
| 3.3.4. Filtros de procesamiento de imágenes. . . . .                                 | 29        |
| 3.3.5. Aumento de resolución de video e introducción de problemas de <i>timing</i> . | 44        |
| <b>4. Material desarrollado</b>                                                      | <b>50</b> |
| 4.1. Listado de material . . . . .                                                   | 50        |
| <b>5. Conclusión y trabajo futuro</b>                                                | <b>52</b> |
| 5.1. Conclusiones . . . . .                                                          | 52        |
| 5.2. Trabajo Futuro . . . . .                                                        | 54        |

# Índice de tablas

|                                                                                                            |    |
|------------------------------------------------------------------------------------------------------------|----|
| 3.1. Caption . . . . .                                                                                     | 36 |
| 3.2. Utilización de LUT y numero de niveles lógicos de versiones 1 y 2 de módulo<br>grayscale . . . . .    | 46 |
| 3.3. Utilización de LUT y numero de niveles lógicos de versiones 1, 2 y 3 de modulo<br>grayscale . . . . . | 47 |

# Índice de figuras

|                                                                                                         |    |
|---------------------------------------------------------------------------------------------------------|----|
| 1.1. Flujo de diseño de sistemas digitales en FPGA . . . . .                                            | 7  |
| 3.1. Vista del flujo de trabajo en Visual Studio Code . . . . .                                         | 21 |
| 3.2. Formas de onda vistas por GTKWave . . . . .                                                        | 22 |
| 3.3. Esquemático de las conexiones del puerto VGA . . . . .                                             | 24 |
| 3.4. Diagrama de alto nivel de dispositivos distribuidos conectados por interfaz UART. . . . .          | 24 |
| 3.5. Vista de las actividades en el 8bitworkshop IDE . . . . .                                          | 27 |
| 3.6. Actividades del simulador replicadas en la FPGA y mostradas en un monitor . . . . .                | 28 |
| 3.7. BRAM siendo accedida dos ciclos antes de mostrar los píxeles en pantalla . . . . .                 | 30 |
| 3.8. Sistema con la inclusión de la BRAM . . . . .                                                      | 30 |
| 3.9. Configuración de la BRAM . . . . .                                                                 | 31 |
| 3.10. Imagen original mostrada en pantalla . . . . .                                                    | 32 |
| 3.11. Funcionamiento interno del color scrambler . . . . .                                              | 33 |
| 3.12. Canales de color de la imagen separados utilizando el filtro color scrambler . . . . .            | 33 |
| 3.13. Sistema con la inclusión del color scrambler . . . . .                                            | 33 |
| 3.14. Sistema con la inclusión del grayscale . . . . .                                                  | 34 |
| 3.15. Comparación de la imagen normal y el filtro grayscale . . . . .                                   | 34 |
| 3.16. Enumeración de píxeles en la pantalla . . . . .                                                   | 36 |
| 3.17. Diagrama del sistema con la inclusión del dithering . . . . .                                     | 37 |
| 3.18. Comparación de la imagen normal y el filtro dithering . . . . .                                   | 37 |
| 3.19. Comparación de la imagen con grayscale y grayscale con dithering . . . . .                        | 37 |
| 3.20. Representación del buffer . . . . .                                                               | 38 |
| 3.21. Entradas y salidas del buffer . . . . .                                                           | 39 |
| 3.22. Representación del problema con el buffer . . . . .                                               | 40 |
| 3.23. Configuración del CORDIC . . . . .                                                                | 43 |
| 3.24. Comparación de imágenes antes y después de activar el filtro Sobel . . . . .                      | 43 |
| 3.25. Diagrama del sistema con la inclusión del Sobel . . . . .                                         | 44 |
| 3.26. Reporte de <i>timing</i> en 640x480 . . . . .                                                     | 44 |
| 3.27. Reporte de <i>timing</i> en 800x600 que no cumple requerimientos de <i>timing</i> . . . . .       | 44 |
| 3.28. Visualización del peor <i>path</i> en el esquemático a 800x600 . . . . .                          | 45 |
| 3.29. Reporte de <i>timing</i> en 800x600 luego de aproximar Sobel . . . . .                            | 45 |
| 3.30. Visualización del peor <i>path</i> en el esquemático a 800x600 luego de aproximar Sobel . . . . . | 46 |
| 3.31. Reporte de <i>timing</i> exitoso en 800x600 . . . . .                                             | 46 |
| 3.32. Reporte de <i>timing</i> en 1024x768 que no cumple requerimientos de <i>timing</i> . . . . .      | 47 |
| 3.33. Visualización del peor <i>path</i> en el esquemático a 1024x768 . . . . .                         | 47 |
| 3.34. Reporte de <i>timing</i> final en 1024x768 . . . . .                                              | 47 |

|                                                                                                            |    |
|------------------------------------------------------------------------------------------------------------|----|
| 3.35. Diagrama de la máquina de estados generado por TerosHDL que guarda la<br>imagen en la BRAM . . . . . | 49 |
|------------------------------------------------------------------------------------------------------------|----|

# 1 | Introducción

El presente informe reporta el trabajo realizado en torno al desarrollo de experiencias prácticas orientadas a la enseñanza de los fundamentos del diseño de sistemas digitales utilizando herramientas de diseño y tecnologías modernas. En específico, se consideró el diseño de actividades para ser insertadas en el marco de la asignatura ELO212 - Laboratorio de Sistemas Digitales, perteneciente al Departamento de Electrónica de la Universidad Técnica Federico Santa María (UTFSM). En su versión actual, esta asignatura apunta a introducir los fundamentos del diseño de sistemas digitales utilizando *Hardware Description Languages* (HDL) y su posterior implementación y evaluación en *Field Programmable Gate Arrays* (FPGA).

El trabajo acá reportado corresponde a una continuación directa a la memoria de titulación desarrollada por Patricio Henríquez, titulada *Diseño de experiencias prácticas para diseño y verificación de sistemas digitales utilizando lenguajes de descripción de hardware y herramientas de simulación* [4]. El trabajo previo mencionado aborda el diseño de experiencias con un énfasis en simulación, un estudio de distintas herramientas de diseño que puedan ser utilizadas en el curso, y la creación de *scripts* de verificación para revisión automatizada/asistida de actividades incorporando técnicas de verificación modernas y procedimientos utilizados en la industria. El trabajo desarrollado en esta memoria utiliza y valida los procedimientos para diseño de experiencias establecidos en [4], enfocándose en los procesos posteriores a la verificación funcional, incluyendo implementación, análisis de desempeño y requerimientos temporales, programación de la FPGA, y la conexión con periféricos.

Este capítulo entrega la motivación y contexto para el trabajo realizado, y establece los problemas y desafíos que se abordarán. Posteriormente, se especifican los objetivos, alcances y principales contribuciones de este trabajo, para finalizar con la descripción de la organización del resto del informe.

## 1.1. Motivación y contexto

Esta memoria de título se origina a raíz de la constante necesidad de actualizar los enfoques educativos y métodos de evaluación para la asignatura “ELO212: Laboratorio de Sistemas Digitales”, impartida en el Departamento de Electrónica de la UTFSM. A lo largo de los años, esta asignatura se ha destacado por su fuerte enfoque práctico, mediante el cual los estudiantes trabajan en forma grupal, abordando tareas relacionadas con el diseño, la implementación y la validación de sistemas digitales utilizando un flujo de diseño y herramientas orientados a FPGA. La metodología de enseñanza se basa en sesiones de laboratorio semanales en las cuales se realizan actividades dirigidas, donde los grupos de estudiantes aplican conocimientos específicos

adquiridos previamente en sesiones teóricas y por medio de auto-estudio para resolver desafíos relacionados con el diseño y la implementación de sistemas de creciente complejidad. Todo esto se lleva a cabo bajo la supervisión cercana de profesores y ayudantes.

Si bien el enfoque práctico se considera esencial para una asignatura de esta naturaleza, el uso de tecnologías modernas que están sujetas a cambios disruptivos periódicos de corto plazo, como rápidos cambios en los modelos de FPGA o en los *softwares* disponibles, impone desafíos particulares relacionados con la necesidad constante de revisar y actualizar las experiencias y actividades a desarrollar durante las sesiones.

Para dar un contexto a la motivación y objetivo de este trabajo, es necesario entender las etapas del flujo del diseño de un sistema digital. La Figura 1.1 muestra un diagrama de flujo del proceso que va desde la formulación de la intención de diseño hasta la implementación física del sistema en *hardware*. Cada una de estas etapas se describe brevemente a continuación:

- **Intención de diseño / Especificación de requerimientos:** Corresponde a la especificación de un sistema por medio de un comportamiento deseado de entrada/salida y la definición de requerimientos técnicos específicos y métricas cuantificables y verificables. Por ejemplo, una actividad típica a desarrollar en el curso es la implementación de un sistema antirebote para pulsadores electromecánicos, en los cuales se pide que el sistema filtre los pulsos de presión del pulsador que duren menos de cierta cantidad de ciclos (configurable), y que por cada pulso de entrada que dure más de una cantidad de ciclos se genere un único pulso de salida de exactamente un ciclo de reloj.
- **Diseño del sistema y descripción en HDL:** El diseño a nivel de sistema corresponde a la creación de un diagrama de bloques de alto nivel, incluyendo los bloques principales, o módulos, con sus entradas, salidas, e interconexiones. Posteriormente, el diseño arquitectural corresponde a la descripción de la lógica interna de cada bloque funcional y sus interfaces utilizando las abstracciones provistas por un HDL. Para el caso de interés de este trabajo, se trabaja a nivel de descripción RTL.
- **Simulación de comportamiento:** Se verifica la funcionalidad de la descripción RTL del sistema para verificar que la lógica descrita ejecuta el comportamiento de entrada/salida deseado mediante el uso de *testbenches*. Para esto, se definen un conjunto de estímulos o valores de entrada para los cuales se conoce su salida deseada, y se ingresan como estímulos a tiempos de simulación predefinidos para verificar que las salidas toman los valores esperados. Otra forma de realizar lo anterior es mediante el uso de SystemVerilog *Assertions*, que permiten definir reglas para verificar que las señales de un sistema sigan el comportamiento especificado en la intención de diseño, con base en las reglas anteriores el sistema puede ser expuesto a distintos estímulos de entrada, y en el caso de que una de las reglas no se cumpla, la prueba entregará mensajes de error. En este nivel del flujo de diseño, solo se valida el comportamiento lógico, y no se consideran detalles de la implementación física, como los retardos o comportamiento de señales eléctricas.
- **Síntesis:** Esta etapa toma como entrada la descripción en HDL, la librería de primitivas disponibles de la FPGA objetivo, las restricciones de síntesis definidas (por ejemplo, restringir el uso de una primitiva o esquema de codificación de máquinas de estado), y otras especificaciones para generar un *gate-level netlist*, el cual corresponde una lista de las primitivas a utilizar y las conexiones entre estos.
- **Implementación:** Esta etapa tiene como entradas el *gate-level netlist* generado anterior-

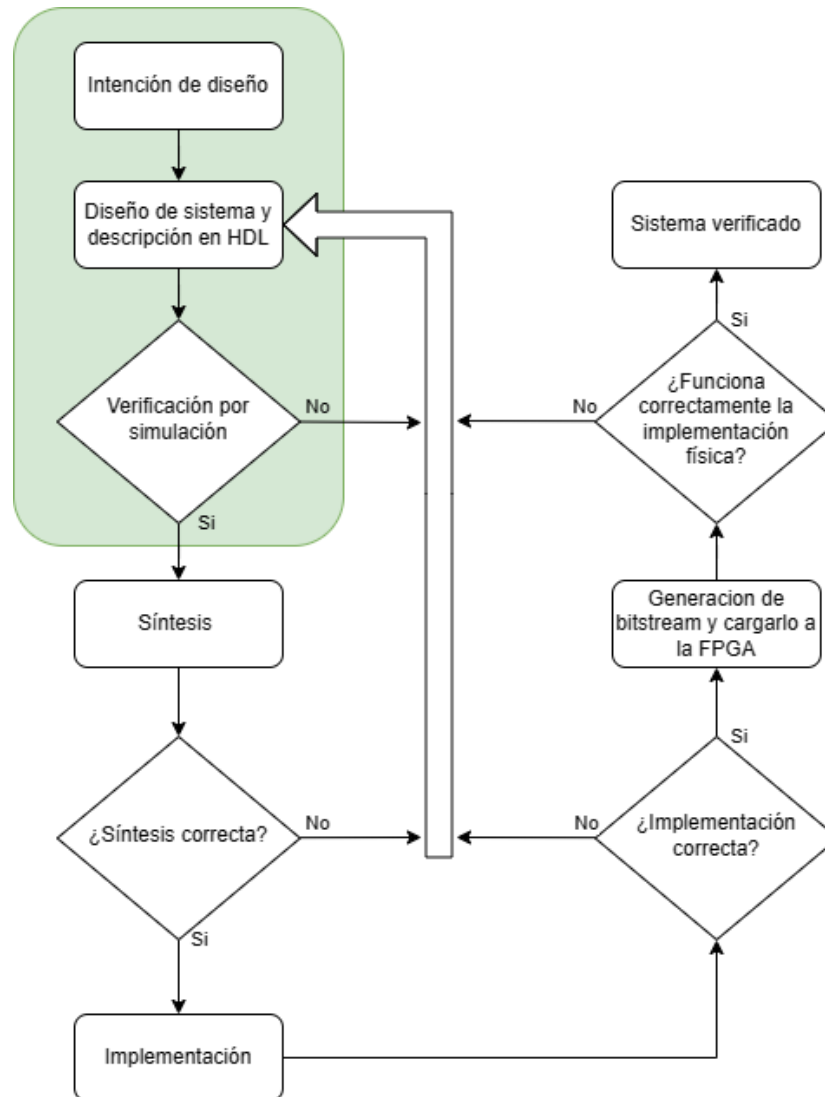


Figura 1.1: Flujo de diseño de sistemas digitales en FPGA

mente y un archivo *constraint* que asigna que puertos del módulo top corresponden a que puertos en el chip físico, las propiedades del reloj y otros requerimientos y restricciones de área, *timing*, donde la latencia de la lógica entre registros debe permanecer dentro de ciertos rangos, consumo de energía y otros. Con estas entradas, se intenta recrear el *gate-level netlist* utilizando los recursos físicos de la FPGA objetivo y cumpliendo las restricciones anteriormente definidas, en el caso de ser exitoso, se generara un listado de todos los recursos físicos a utilizar en la FPGA, y la configuración que estos deben tener, además del enrutamiento de las señales entre estos recursos.

- **Generación de bistream y programación de chip FPGA:** Utilizando las salidas de la etapa anterior, se genera un bitstream, el cual es un archivo que contiene la configuración completa de la FPGA, el cual se utiliza para programarla, permitiendo verificar el correcto funcionamiento del sistema ya cargado en la FPGA objetivo, terminando el proceso en el caso de ser exitoso.

Es importante enfatizar que cada una de las etapas indicadas en el flujo de diseño ejecuta múltiples subetapas, cada una representando un proceso computacionalmente complejo. Si bien las etapas del flujo de diseño tienen una naturaleza de ejecución secuencial (los archivos de salida de una etapa corresponden a la entrada de la siguiente etapa), cada etapa se enfoca en procesos distintos, cada uno con sus potenciales fuentes de error, por lo cual es posible que los archivos generados en una etapa de abstracción alta arrastren errores que solo sean detectados en etapas posteriores asociados a capas de abstracción más bajas. La complejidad de cada etapa es tal, que incluso para sistemas simples cada una puede tomar varios minutos, siendo común que se pueda llegar al orden de horas en sistemas medianamente complejos. Es por esto que resulta crítico aplicar procedimientos que permitan detectar la mayor cantidad de errores en etapas tempranas. Así, es importante que los estudiantes participen en distintas experiencias que les permitan desarrollar habilidades en todas estas etapas de diseño.

Como parte de un plan de renovación de experiencias, y también en respuesta a la necesidad de operar en formato online durante el período de pandemia iniciado el 2019, con estudiantes sin acceso al equipamiento de *hardware*, el año 2020 se implementó un plan de renovación de experiencias con un fuerte foco en profundizar los aspectos asociados a las etapas iniciales del flujo de diseño, correspondientes a la sección marcada en verde en la Figura 1.1, incluyendo formalidades en la especificación de requerimientos, descripción en RTL, y especialmente en verificación funcional de los diseños mediante *testbenches*. Este trabajo se materializó en el trabajo de memoria referenciado en [4], que dejó como entregables guías de trabajo, cápsulas de vídeo que ilustran conceptos fundamentales del proceso de descripción en HDL y simulación funcional, y *scripts* para revisión asistida que verifican en forma semi-automatizada que los diseños entregados por los estudiantes cumplan con los requerimientos establecidos en la intención de diseño. Las herramientas desarrolladas se han integrado desde entonces al flujo de trabajo regular de la asignatura y han generado evidentes beneficios para los estudiantes y para el *staff*. En la versión actual del curso, los estudiantes trabajan durante la primera mitad del semestre exclusivamente en base a simulaciones, sin tener acceso al *hardware* físico. Este enfoque permite a los estudiantes reforzar los conceptos asociados a las primeras etapas del flujo de diseño, siendo evaluados de forma exhaustiva, rigurosa, y clara al momento de detectar errores, recibiendo *feedback* específico sobre puntos de fallo y como corregirlos.

Luego del período de pandemia, y una vez se retomaron las actividades presenciales regulares en que los estudiantes tienen acceso al equipamiento físico, y habiendo sistematizado los procesos de enseñanza y evaluación asociados a las primeras etapas del diseño, quedó en evidencia la necesidad de sistematizar las actividades y procedimientos de revisión para las siguientes etapas del diseño asociadas a la implementación física. Por ejemplo, se observó que, si bien las simulaciones permiten analizar en detalle el comportamiento de sistemas relativamente simples como los estudiados en la primera parte del curso (por ejemplo, arreglos de contadores, circuitos aritméticos, y máquinas de estado), a medida que se integran sistemas más complejos que utilizan protocolos de comunicación e interfaces de vídeo, la cantidad de señales y las posibles combinaciones entre estas hacen que las simulaciones tomen demasiado tiempo en ejecutarse y que la inspección visual de formas de onda se vuelva inefectiva para identificar detalles operacionales, a menos que se apliquen técnicas avanzadas de verificación que escapen del foco del curso.

La experiencia obtenida durante los últimos años en el desarrollo del curso permitió apreciar la relevancia de enfatizar explícitamente la necesidad de entender cada una de las etapas del diseño digital y entender las complejidades asociadas a cada una. Considerando que la versión

del curso realizada al momento de iniciar este trabajo de memoria cubría adecuadamente los aspectos de diseño funcional y simulación, se identificó la necesidad de actualizar la parte experimental del trabajo con *hardware* del curso, aplicando el mismo nivel de rigurosidad a las etapas de síntesis, implementación y validación física, generando experiencias que refuercen estas etapas, haciendo que los alumnos se enfrenten a problemas específicos de estas, como por ejemplo trabajar con recursos limitados y enfrentarse a problemas de sincronización y *timing*, que son aspectos críticos en cualquier sistema sincrónico.

El trabajo realizado en esta memoria apunta a cubrir la necesidad planteada por medio de la preparación de material docente para experiencias prácticas de diseño de sistemas digitales, con un foco en las etapas de implementación, análisis de funcionalidad y desempeño a nivel de componentes físicas, programación del *hardware* y conexión de periféricos.

## 1.2. Planteamiento del problema

Para el desarrollo de esta memoria de título se planteó como objetivo principal el diseño de actividades, la generación de guías de trabajo, material de apoyo, y pautas de evaluación asociados a actividades de laboratorio enfocadas en los etapas de implementación física de sistemas digitales, considerando como tecnología base las FPGA. Específicamente, se considera la realización de actividades para el uso de *Intellectual Property* (IP) *cores*, y la implementación de una interfaz de video VGA, comenzando con la descripción e implementación del *driver* asociado, y realizar actividades de complejidad incremental hasta llegar a tareas de procesamiento de vídeo que ilustren los desafíos asociados a verificación de requerimientos temporales y *trade-off* asociados.

El trabajo desarrollado consideró las siguientes etapas:

- **Revisión y renovación de procedimientos y material base de actividades orientadas a simulación:** Esta etapa tiene como objetivo identificar potenciales puntos de actualización del material generado en [4], incorporando experiencias y *feedback* recolectado durante la ejecución del curso entre los años 2021 y 2023. Además, se realiza una nueva revisión sobre cursos similares disponibles en plataformas de internet, y una revisión actualizada de herramientas de *software* para el diseño alternativas a Vivado.

- **Exploración de nuevo flujo de trabajo con herramientas de diseño alternativas:** el software por defecto del curso siempre ha sido Vivado de Xilinx, el cual permite cubrir todas las etapas de diseño especificadas en la Figura

pero debido a algunos defectos, este software no es el ideal para todos los alumnos, por lo que se desarrollará un nuevo flujo de trabajo, utilizando nuevas herramientas, para los alumnos y staff.

- **Generación de material orientado a interfaz de vídeo y procesamiento de imágenes con implementación en *hardware*:** Si bien versiones anteriores del curso han realizado actividades prácticas con la interfaz VGA, estas se presentaban a los estudiantes a modo de proyecto para trabajo independiente, enfocándose principalmente en los resultados visuales sin contar con una guía de trabajo, material de apoyo, ni especificación de requerimientos sobre el *hardware* descrito. El objetivo de esta etapa corresponde a la formalización de actividades orientadas a procesamiento de vídeo por

medio de guías de trabajo con especificación de objetivos de aprendizaje y resultados esperados medibles y verificables. Se considera también la incorporación de una guía enfocada en análisis de *timing* de un circuito secuencial, lo cual no se había evaluado en versiones anteriores del curso.

### 1.3. Alcances y contribuciones

Para el trabajo desarrollado y el material generado se consideraron los siguientes alcances:

- El lenguaje RTL considerado tanto para descripción de *hardware* es SystemVerilog (especificado en el *standard* IEEE 1800 [5]), el cual está establecido como el lenguaje oficial del curso ELO212.
- Todas las actividades preparadas consideran la implementación del flujo de trabajo orientado a FPGA, utilizando la herramienta de diseño Xilinx's Vivado 2022.2, y considerando como *hardware* objetivo la tarjeta de desarrollo Nexys A7 de Digilent con un chip Artix7 de Xilinx, la cual, al momento de realizar este trabajo, es el *hardware* que se encuentra disponible para los estudiantes durante la realización del curso. Dado que se usa principalmente código estándar de SystemVerilog, se espera que los diseños sean válidos para nuevas versiones de Vivado y puedan ser portados directamente o con modificaciones menores a otras tarjetas con FPGA de Xilinx, aunque los resultados en términos de desempeño y uso de recursos pueden cambiar.
- En la creación del material de interfaz de video se considera exclusivamente el protocolo VGA, debido a que es el puerto de video disponible en la tarjeta Nexys A7. En relación a lo anterior, también existen algunas sutilezas de la tasa de refresco y resoluciones a utilizar con los monitores presentes en el laboratorio, específicamente la tasa de 60Hz no es reconocida, por lo que restringe a una tasa de 75Hz la cual funciona correctamente. En cuanto a la resolución, se trabajó con especificaciones con relación de aspecto 4/3. Aunque el puerto VGA se podría considerar parcialmente obsoleto, sigue siendo utilizando como una alternativa de bajo costo para ilustrar los conceptos fundamentales de los protocolos de vídeo. Se espera que la documentación generada para el protocolo VGA sirva de base para generar nuevo material orientado a HDMI.

La contribución principal de este trabajo de memoria se presenta en la forma de entregables que incluyen guías de actividades, *scripts* de revisión asistida mediante *assertions*, cápsulas de vídeo, y otros. El material generado permite sistematizar la entrega de contenidos asociados a las etapas de implementación física de un sistema digital utilizando FPGA, e incorpora nuevas actividades asociadas al uso de IP y análisis de *timing* que hasta ahora no habían sido incorporadas al curso. Todo el material generado fue discutido y revisado en conjunto con el profesor guía, profesores que regularmente dictan la asignatura, y *staff* de ayudantes con distinto nivel de experiencia. Parte de las actividades preparadas se incorporaron directamente al flujo de trabajo de la asignatura durante el primer semestre del 2023. El listado de entregables específicos se encuentra disponible en el Capítulo 4, y todo el material asociado esta respaldado en un repositorio, quedando accesible para ser utilizado durante versiones futuras del curso.

Es importante destacar que el autor de este trabajo cursó ELO212 como estudiante el año 2019, y desde el 2021 hasta la fecha de entrega de este trabajo ha participado como ayudante de la asignatura, tanto en sus formatos online y presencial. Los resultados de este trabajo incorporan

la experiencia personal considerando la perspectiva de estudiante y ayudante experimentado, lo cual se espera sea de utilidad para mejorar la calidad del aprendizaje de los estudiantes.

## 1.4. Organización del informe

La mayor parte de los desarrollos derivados de este trabajo de título consisten en diseños, códigos de ejemplo, material audiovisual, y tutoriales, los cuales serán utilizados como insumos para la preparación de material educacional. El material fue revisado y validado por parte del *staff* experimentado del curso ELO212, y se encuentra disponible en un repositorio. Por indicaciones del profesor guía, al momento de completar este trabajo, el repositorio de material se encuentra con acceso restringido (se puede solicitar acceso consultando al autor o profesor guía). Debido a esto, el presente informe solo se enfoca en aspectos de contexto general sobre el trabajo desarrollado, omitiendo detalles específicos y evaluaciones experimentales. Si bien estos insumos se encuentra disponibles y revisados por el profesor guía, se espera que sean liberados posteriormente, una vez pase una etapa adicional de depuración y se complete la preparación de guías de trabajo a aplicar en el curso.

El resto del informe se encuentra organizado en los siguientes capítulos:

- **Capítulo 1 - Introducción:** Se presentan los antecedentes y principales motivaciones de este trabajo, para luego plantear los problemas a resolver e introducir la estructura del documento.
- **Capítulo 2 - Antecedentes:** Se actualizan los resultados entregados en el estudio de cursos similares al nuestro y herramientas alternativas a Vivado que se realizaron en [4].
- **Capítulo 3 - Diseño de nuevo material:** Se genera un nuevo flujo de trabajo basado en herramientas alternativas a Vivado, luego se enfoca en el diseño de una experiencia de interfaz de vídeo en VGA y procesamiento de imágenes.
- **Capítulo 4 - Material desarrollado:** Se lista el material generado con una breve explicación de su contenido. Como se mencionó, por la naturaleza del material, este se encuentra por ahora con acceso restringido y se omite entregar detalles técnicos.
- **Capítulo 5 - Conclusión y trabajo futuro:** Se resumen los resultados obtenidos y el conocimiento adquirido, seguido de una propuesta de trabajos para el futuro de la asignatura.

## 2 | Antecedentes

La primera parte de este trabajo consistió en la revisión de material de referencia y herramientas disponibles para realizar las tareas asociadas al curso, con el fin de identificar potenciales actualizaciones al plan de trabajo y las actividades realizadas durante el semestre. Partiendo de la base establecida en el trabajo [4], se revisaron nuevas versiones de cursos similares a ELO212 de distintas instituciones con material abiertamente disponible en la web, nuevas herramientas de desarrollo que presentan alternativas a Vivado en algunas etapas del desarrollo, y revisión de los *scripts* de revisión asistida que se aplican para la revisión de evaluaciones basadas en simulación funcional.

Considerando todo el material revisado, en varios casos las conclusiones sobre la utilidad de algunas herramientas fueron similares a lo ya reportado en el informe [4], reforzando la validez y vigencia de gran parte de los procedimientos ya establecidos. Así, en este capítulo solo se reportan los aspectos complementarios al reporte previo, incluyendo descripción de cursos similares y algunas propiedades útiles disponibles en versiones recientes de herramientas de desarrollo.

### 2.1. Estudio de cursos similares

Con el propósito de poner el curso ELO212 en contexto a otros que se pudieran encontrar online, se realizó un estudio de todos los cursos gratuitos disponibles en las plataformas Coursera y EDX que fueron encontrados bajo el termino de búsqueda “FPGA” en octubre del 2022. En esta sección primero se describen las principales características y el foco principal de cada curso revisado, para luego ponerlo en contexto con los contenidos de ELO212.

#### 2.1.1. Cursos revisados

##### **Electrónica Digital Bit a Bit: Fundamentos, Verilog y FPGA [6]**

Este curso viene de la Pontificia Universidad Católica de Chile y sirve como una introducción a sistemas digitales, FPGA y Verilog (*standard* IEEE1364 [7], el cual será mencionado de ahora en adelante como Verilog). Este empieza con los fundamentos de los sistemas digitales, para luego en el siguiente capítulo explicar los fundamentos de Verilog. En la segunda mitad del curso se deja de ver Verilog y se comienza a hablar de compuertas lógicas y otros componentes a un nivel físico y como estos se relacionan al funcionamiento de la FPGA, luego en conjunto

a otras cosas, se enseñan distintas técnicas de simplificación de sistemas digitales, como los mapas de Karnaugh.

### **Electrónica Digital Bit a Bit: Diseñando en Verilog para FPGA [8]**

Este curso es un sucesor al anterior, mas en esta ocasión el enfoque es específicamente el diseño de sistemas en Verilog para FPGA, cubriendo una gran gama de circuitos distintos a lo largo de los capítulos, dando ejemplos de como describirlos en Verilog. Se enseñan bloques de circuitos combinacionales, flip flops, registros, circuitos síncronos como máquinas de estado, contadores y circuitos secuenciales asíncronos. Además, se dedica un capítulo al funcionamiento básico de un procesador, sin adentrarse en como describirlo en Verilog.

### **Introducción al diseño de *hardware* con Verilog [9]**

Este curso de GalileoX disponible en EDX resultó ser bastante completo, entregando ejemplos en Verilog de múltiples circuitos, como sumadores, multiplexores, flip flops, contadores y máquinas de estado. Agregado a lo anterior, se adentra en como realizar simulaciones para comprobar que los códigos se comporten de la manera esperada.

### **MIT 6.111 Introductory Digital Systems Laboratory [10]**

Este curso impartido en el *Massachusetts Institute of Technology* (MIT), es el equivalente de ELO212 dictado en la UTFSM, siendo un curso tipo laboratorio con actividades evaluadas además de clases sobre la implementación de variados tipos de sistemas digitales. La versión revisada es la que se dictó en el 2019, al ser la más reciente que se encontró disponible en su pagina web.

Este curso utiliza Verilog como el lenguaje por elección para describir sus sistemas y se destaca por cubrir todo un rango de contenidos equivalente al que se revisa en ELO212, incluso superándolo hacia el final de la asignatura, incluyendo conceptos como el uso de memorias externas en una FPGA, utilización de múltiples protocolos de comunicación (UART, SPI, I2C y I2S) y procesamiento de imágenes.

La primera diferencia a destacar es que el uso de vídeo se ve antes de las máquinas de estado. Esto resulta interesante al permitir que los alumnos puedan internalizar los conceptos básicos de vídeo antes de lo que lo hacen actualmente en el curso de la UTFSM, al mismo tiempo de permitir subir los niveles de abstracción para interactuar con periféricos llamativos en lugar de visualización de señales, lo cual puede ayudar a incrementar la motivación de los estudiantes.

Una adición de este curso respecto a ELO212 es la de un esquema de trabajo *Major/Minor* con las máquinas de estado (FSM), donde una FSM principal delega trabajo a otras máquinas de estado y espera hasta que terminen de realizar alguna operación determinada, para luego recibir el resultado y continuar.

**FPGA computing systems: A Bird's Eye View [11]**

En este curso del *Politecnico di Milano* se hace una explicación a nivel teórico sobre la reconfiguración parcial dinámica, lo cual es la capacidad de una FPGA de reconfigurar una porción de su lógica durante el tiempo de ejecución. Al comienzo del curso se narra la historia de los sistemas reconfigurables en general, para luego en los siguientes módulos adentrarse en los detalles más específicos aplicables a las FPGA, además de esto se ve cuales son los bloques fundamentales que componen la lógica de una FPGA y como se estructura un *bitstream*, lo cual es información importante para el tópico de este curso en particular.

**FPGA computing systems: Partial Dynamic Reconfiguration [12]**

En esta continuación al anterior curso del *Politenico di Milano*, se hace un estudio más a fondo sobre la reconfiguración parcial dinámica, dando en un comienzo más información sobre el funcionamiento a nivel físico de un sistema que utiliza esta funcionalidad, para luego enseñar varios flujos de trabajo con los cuales se pueden diseñar sistemas utilizando esta herramienta. En el capítulo final del curso se describen arquitecturas de FPGA en la nube para ser utilizadas como aceleradores de otros procesos.

**2.1.2. Estado de ELO212 en relación a cursos alternativos revisados.**

A partir de la revisión de los cursos mencionados en Coursera y EDx, se pudo observar que todos corresponden a cursos introductorios al diseño de circuitos digitales mediante HDL y que se encuentran en general, alineados a los contenidos que se revisan actualmente en el curso ELO212, partiendo con la descripción de primitivas básicas para ir avanzando hacia sistemas y abstracciones más complejas en forma incremental. Con respecto a la componente práctica, se observó que los cursos revisados utilizaban como lenguaje de descripción de *hardware* Verilog en su *standard* IEEE1364 [7], lo que los deja desactualizados con respecto a ELO212, en el cual actualmente se utiliza SystemVerilog en su *standard* IEEE 1800 [5]. Además, todos los cursos se enfocaban únicamente en la descripción en HDL y en la simulación, con mayor énfasis en lo primero. Por otra parte, en ninguno de los casos se cubrían tutoriales para el uso de las herramientas asociadas, dejando esta tarea a los estudiantes. De la misma forma, ninguno de los cursos revisa en detalle los aspectos asociados a las capas más bajas de abstracción, asociadas a la implementación y configuración de los dispositivos. Por lo tanto, se concluye que, en términos de los contenidos entregados, el curso ELO212 es lo suficientemente completo para ser comparable a lo ofrecido, incluso superándolos para un curso introductorio.

El curso más completo de los revisados corresponde al ofrecido por el MIT, el cual al igual que ELO212 consiste de un laboratorio con actividades prácticas que cubre la completitud del contenido de ELO212 además de algunos tópicos más avanzados. Este curso, a cambio del nuestro, es dictado a estudiantes más experimentados que ya han cursado alguna asignatura de sistemas digitales, algoritmos, y procesamiento de imágenes, por lo que es de esperar que sean capaces de realizar experiencias más avanzadas enfocándose exclusivamente en aspectos de implementación en *hardware*, sin estar aprendiendo de conceptos fundamentales al mismo tiempo. Si resultaría interesante evaluar algunas modificaciones al flujo de actividades con base en lo observado en el curso del MIT [10], que trabaja con interfaces de vídeo en forma temprana, incluso antes de revisar máquinas de estado. Este enfoque tendría sentido, pues

para manejar una interfaz de vídeo y desplegar formas básicas se requiere principalmente el manejo de contadores sincronizados, lo que se ve en las primeras sesiones. Este enfoque permitiría aumentar la motivación de los estudiantes al tener resultados visibles y llamativos de los conceptos que están revisando, más allá de las interacciones con botones y LEDs, en una etapa temprana del curso. También, en la parte en que se trabaja con FSMs, resultaría interesante incorporar una actividad con un estructura del tipo Major/Minor, con el objetivo de que los alumnos aprendan a manejar comunicaciones más avanzadas entre módulos.

Los cursos revisados del *Politecnico di Milano* [11,12] cubren aspectos teóricos y prácticos asociados a reconfiguración parcial de FPGAs, lo que se considera un tópico avanzado que escapa de los objetivos de un curso introductorio. Sin embargo, resulta interesante analizar en más detalle su contenido para incorporarlo como ejemplo de tópico avanzado en este curso o en su versión avanzada.

Con base en lo observado, se concluye que la versión actual del curso ELO212 se encuentra bien estructurada y entrega lo contenidos suficientes para un curso introductorio, equiparándose a otros cursos populares de otras instituciones nacionales e internacionales. Se destaca también el factor distintivo del fuerte enfoque práctico, cubriendo no solo las partes de descripción de *hardware* y simulación, sino también aspectos asociados a la implementación física y el uso de herramientas de diseño típicas en la industria que no son cubiertos en los otros cursos revisados. En este sentido, se espera que la formalización de actividades orientadas a estas capas más bajas de abstracción que, como ya ha sido mencionado, es uno de los objetivos de este trabajo de memoria y fortalecerán aún más el curso.

## 2.2. Scripts de revisión asistida

Una de las responsabilidades que se asumieron al realizar este proyecto de título, es la de actualizar las pruebas automáticas que se utilizan en el curso para asistir la revisión del trabajo de los estudiantes.

Estas pruebas de revisión asistida funcionan utilizando una mezcla de *scripts* Tcl y códigos escritos en SystemVerilog *Assertions*, los cuales son utilizados para verificar distintos comportamientos de las señales de salida de un módulo durante una simulación, respecto a estímulos en las entradas y el paso del tiempo. Distintas reglas, llamadas *assertions*, pueden ser formadas para poder determinar que comportamientos son, o no son esperados en un sistema para poder verificar si fue correctamente implementado o no.

Estas pruebas fueron creadas en el trabajo [4], y se requiere verificación de si un usuario no experto puede utilizarlas sin ayuda de usuarios más experimentados haciendo uso de la documentación disponible. También se necesita saber si modificarlas para adaptarlas a cambios menores en las guías es factible o no, estos aspectos serán revisados mas adelante durante el capítulo del diseño de nuevo material.

## 2.3. Herramientas de diseño alternativas a Vivado

En el trabajo reportado en el informe [4], se presenta una descripción de distintas herramientas de acceso gratuito para el diseño de sistemas digitales utilizando HDL. En dicho trabajo,

y considerando las versiones de las herramientas disponibles en ese entonces, se llegó a la conclusión de Xilinx's Vivado era la única herramienta que cumplía con todos los requerimientos establecidos para su utilización en el curso. Por otro lado, algunas desventajas de Vivado incluyen sus altos requerimientos de recursos computacionales (alto consumo de RAM y disco duro) y ser una herramienta cerrada con documentación limitada. Si bien se consideraron varias alternativas de herramientas que presentaban ventajas como ser de código abierto, amplia comunidad de usuario y documentación, y reducción considerable en los requerimientos computacionales, las versiones revisadas en ese entonces presentaban limitaciones que impedían su utilización directa en el curso ELO212. Un factor crítico en este contexto fue que ninguna de las herramientas evaluadas soportaba SystemVerilog, que es el lenguaje establecido en el curso.

Como parte del trabajo de investigación realizado en las asignatura ELO307 previo a esta memoria de título, se observó que algunas de las herramientas más populares para descripción y simulación de circuitos digitales descritos en HDL habían incorporado soporte para SystemVerilog en sus versiones recientes. Por este motivo, se decidió realizar una revisión actualizada de estas herramientas para evaluar si podían ser incorporadas como parte del flujo de trabajo del curso, ofreciendo una alternativa a los estudiantes en el caso de que requieran realizar pruebas sin requerir un computador personal de altas prestaciones.

En esta sección, se resumen las características de las herramientas que, en sus recientes versiones, se identificaron como útiles para ser integradas al flujo de trabajo del curso. Posteriormente, en la Sección 3.2, se describe un flujo de trabajo concreto que integra estas herramientas, ofreciendo no solo un complemento a Vivado para etapas tempranas del diseño, sino que también algunas características de visualización que no se encuentran disponibles en Vivado.

### 2.3.1. IcarusVerilog [1]

IcarusVerilog es una herramienta multiplataforma de simulación enfocada para Verilog IEEE-1364 2005 [7]. Debido a lo extenso y complejo del estándar se reconoce que aún existen trabajos inconclusos en relación al compilador y sintetizador propiamente tal, la gran ventaja es que se mantiene en continuo desarrollo, habiendo introducido hace poco soporte para SystemVerilog IEEE-1800 2012 [13], el que mejora con cada actualización de la herramienta y ha demostrado estar lo suficientemente desarrollado para las estructuras básicas utilizadas en el curso.

La herramienta permite generar un archivo .vvp utilizando el comando iverilog sobre el *testbench* del modulo deseado, este archivo puede ser llamado en conjunto al comando vvp, que es un motor de simulación que ejecutará el *testbench*. Si se configura el *testbench* de manera correcta, luego de correr la simulación se generara un archivo .vcd, el cual contiene la información de todas las señales en el tiempo, estas se pueden ver en su forma de ondas con la aplicación GTKWave [14], que se instala en conjunto a IcarusVerilog. Todo este proceso es simple de realizar y las pruebas que se ejecutaron logran simular los códigos asociados a las experiencias del curso sin problemas.

#### Ventajas

- El uso de la herramienta se encuentra bien documentado, mediante ejemplos concretos de casos de uso y tutoriales para su fácil comprensión.

- Al no presentar una interfaz gráfica (GUI), el requerimiento de procesamiento es menor en comparación a otras herramientas como Vivado.
- La herramienta se encuentra en un desarrollo continuo por lo que en un futuro podría soportar más características de SystemVerilog IEEE-1800 2012 [13], quizás expandiéndose a la última versión del estándar publicado el 2017, las características de SystemVerilog IEEE-1800 2012 [13] que no se soportan actualmente son desconocidas y el desarrollador Stephen Williams solo ha indicado que falta trabajo por hacer sin entrar en más detalle.

### Desventajas

- El soporte para SystemVerilog IEEE-1800 2012 [13] no se encuentra completo de acuerdo a lo establecido por el desarrollador.
- Se limita a simulaciones, por lo que para eventual implementación en FPGA se debe contar con una segunda herramienta que realice dichas operaciones.

### 2.3.2. Visual Studio Code [2]

Visual Studio Code es un editor de código basado en código abierto disponible en todas las plataformas el cual es desarrollado por Microsoft. Este editor de código en particular tiene una amplia selección de extensiones generadas por terceros que añaden nuevas funcionalidades en el caso de desearlas, entre estas extensiones se encuentra una con soporte de SystemVerilog, permitiendo la generación automática de fragmentos de código, destacar la sintaxis, y varias funcionalidades más. Lo anterior sumado a que permite fácil acceso a la terminal para ejecutar los comandos de IcarusVerilog [1], y que incluye soporte integrado de Git, es la razón por la que seleccionó esta herramienta para la edición del código.

### 2.3.3. TerosHDL [3]

TerosHDL es una extensión de Visual Studio Code que busca proporcionar un *integrated development environment* (IDE) sobre el cual trabajar con HDL, que con la configuración correcta permite trabajar los sistemas como proyectos, dando información de las dependencias entre los módulos y manteniendo un historial de resultados de las simulaciones anteriores, también tiene la habilidad de generar documentación de los módulos de manera automática si se escriben bajo un cierto formato y de esta misma manera también puede inferir diagramas de máquinas de estados en base al código, esto resultaría útil en las experiencias más avanzadas del curso.

La lista de funcionalidades es extensa, pero dentro de los puntos a destacar que serán útiles para los alumnos sin mayores complicaciones, se destacan la generación de *templates* de instancias de módulos, la generación de *templates* de simulaciones de modulo y la generación de diagramas de estados basándose en la descripción de una máquina de estados.

### 2.3.4. Simulador VGA

La herramienta web 8bitworkshop [15] es un entorno de desarrollo integrado pensado para ejecutar código de computadores y consolas de videojuegos antiguos la cual emula el *hardware* desde el navegador haciendo uso de JavaScript. También se jacta de soportar el HDL Verilog(*standard* IEEE1364 [7]), pero con base en experimentos realizados en el documento [4], también soporta sintaxis de SystemVerilog(*standard* IEEE1800 [5]).

En conjunto a la capacidad de simular computadoras, consolas de videojuego y código HDL SystemVerilog, esta herramienta dispone de un simulador de monitor VGA que permite ver en tiempo real los cambios que uno realiza en el código, lo cual planea utilizarse para introducir a los alumnos al protocolo VGA en la experiencia . Debe destacarse que esta herramienta se ejecuta localmente en el ordenador por lo que el rendimiento dependerá de las especificaciones del computador que la este ejecutando, y debido a lo anterior, aunque el monitor VGA simulado soporta resoluciones de 640x480, se encontró que una resolución más pequeña de 256x240 proporciona una experiencia mas fluida en computadores de gama baja.

## 3 | Diseño de nuevo material

En este capítulo se describen las bases del nuevo material desarrollado teniendo en cuenta los antecedentes descritos anteriormente y los objetivos planteados en la Sección 1.2, comenzando con una descripción breve del trabajo realizado con los *scripts* de revisión asistida, seguido de la creación de un nuevo flujo de trabajo utilizando herramientas alternativas a Vivado, y luego planteando la creación de nuevas actividades, documentándose los resultados obtenidos en el proceso de diseño.

Respecto a los códigos asociados a las guías, solo se describen aspectos generales, quedando el material asociado con detalles específicos de implementación con acceso restringido en el repositorio.

### 3.1. *Scripts* de revisión asistida

Desde la experiencia del autor como ayudante del curso a partir de la introducción de las pruebas de revisión asistida, el cómo utilizarlas ha sido conocimiento que esencialmente ha sido transmitido de boca en boca, lo cual ha funcionado bien hasta el momento, pero que ha requerido de reuniones donde los ayudantes más experimentados enseñan a utilizarlas a los demás miembros del *staff*, además de comunicaciones posteriores a la reunión para solucionar problemas específicos que puedan tener los ayudantes mas novatos. En un esfuerzo para simplificar el proceso de aprender a utilizar esta herramienta y que no se necesite depender de otros para esto, se trabajó en la creación de un tutorial en la forma de un documento que ilustra la forma correcta de utilizar las pruebas, indicando buenas costumbres que minimizan las posibilidades de encontrarse con errores que han sido encontrados y solucionados en el pasado. Durante la realización del curso ELO212 en el primer semestre del 2023, este tutorial fue entregado a un total de ocho ayudantes de variada experiencia, quienes pudieron ejecutar las pruebas sin problemas en su primer intento luego de seguir las indicaciones del documento.

Otro punto que buscaba verificarse durante la realización de la asignatura, era la facilidad de modificar estas pruebas para acomodar cambios en las guías. En cuanto a la experiencia adquirida modificando las pruebas durante el semestre, cambiar los archivos de *assertions* para ajustarlos a un cambio en el funcionamiento de un módulo o el nombre de las señales involucradas, es una tarea relativamente sencilla luego de haber adquirido un poco de experiencia revisando los archivos. Una tarea que puede ser más compleja es cualquier cambio relacionado a la estructura de los archivos que entregan los estudiantes, ya que adaptar las pruebas para estos cambios requiere corregir los *scripts* Tcl que manejan las pruebas, necesitando cambios coordinados en varios archivos al mismo tiempo, los cuales pueden volverse complejos de

manejar. Afortunadamente estos *scripts* se encuentran actualmente en desarrollo por Patricio Henríquez y estas complicaciones podrían verse solucionadas en un futuro cercano.

## 3.2. Nuevas herramientas para diseño y análisis funcional.

A partir de las herramientas descritas en la Sección 2.3 se decidió crear un nuevo flujo de trabajo alternativo a Vivado, el cual utiliza alrededor de 1 GB de almacenamiento, requiere menos recursos computacionales para funcionar, y todas las herramientas utilizadas son multiplataforma, aunque la compatibilidad entre estas no ha sido estudiada en otras plataformas que no sean Windows. Las herramientas utilizadas son IcarusVerilog [1] para compilar y simular, Visual Studio Code [2] para editar los códigos, GTKWave [14] para mostrar las formas de onda, y TerosHDL [3] para la generación de *templates* de instancias de módulos y simulaciones, además de la visualización de los diagramas de estado de una FSM.

En la Figura 3.1 se puede ver Visual Studio Code, donde se está editando un código para un *testbench*, el cual se puede apreciar coloreado de una forma que simplifica la lectura. Además de esto, se puede ver el terminal en la sección inferior de la pantalla, en el cual se ejecutan los comandos `iverilog` y `vvp` de IcarusVerilog.

En la Figura 3.2 se pueden ver las formas de ondas provenientes de la simulación ejecutada anteriormente, las cuales se están visualizando en GTKWave.

### 3.2.1. Ventajas y desventajas del flujo de trabajo

Ventajas:

- Utilización de herramientas multiplataforma
- Requerimientos de *hardware* menores a los de Vivado, al no tener GUI ni el *overhead* de todas las funcionalidades con las que cuenta Vivado
- Baja utilización de almacenamiento, alrededor de 1 GB
- Generación de templates de instancias de módulos y simulaciones
- Generación de diagramas de máquinas de estado

Desventajas:

- Mensajes de error menos descriptivos a los de Vivado
- Se limita a simulaciones, por lo que todavía se requiere Vivado para implementar
- No genera un árbol de la jerarquía entre los módulos, lo que dificulta el trabajo en diseños con un número alto de estos

Para facilitar el uso de este nuevo flujo de trabajo, se trabajó en la creación de un tutorial para los alumnos en el formato de un vídeo, en el que se explica cómo instalar y utilizar todas las herramientas de la forma más ordenada y simple posible. También se generó un documento con los comandos que deben utilizar aparte de algunos cuántos consejos extras en el uso de las herramientas.

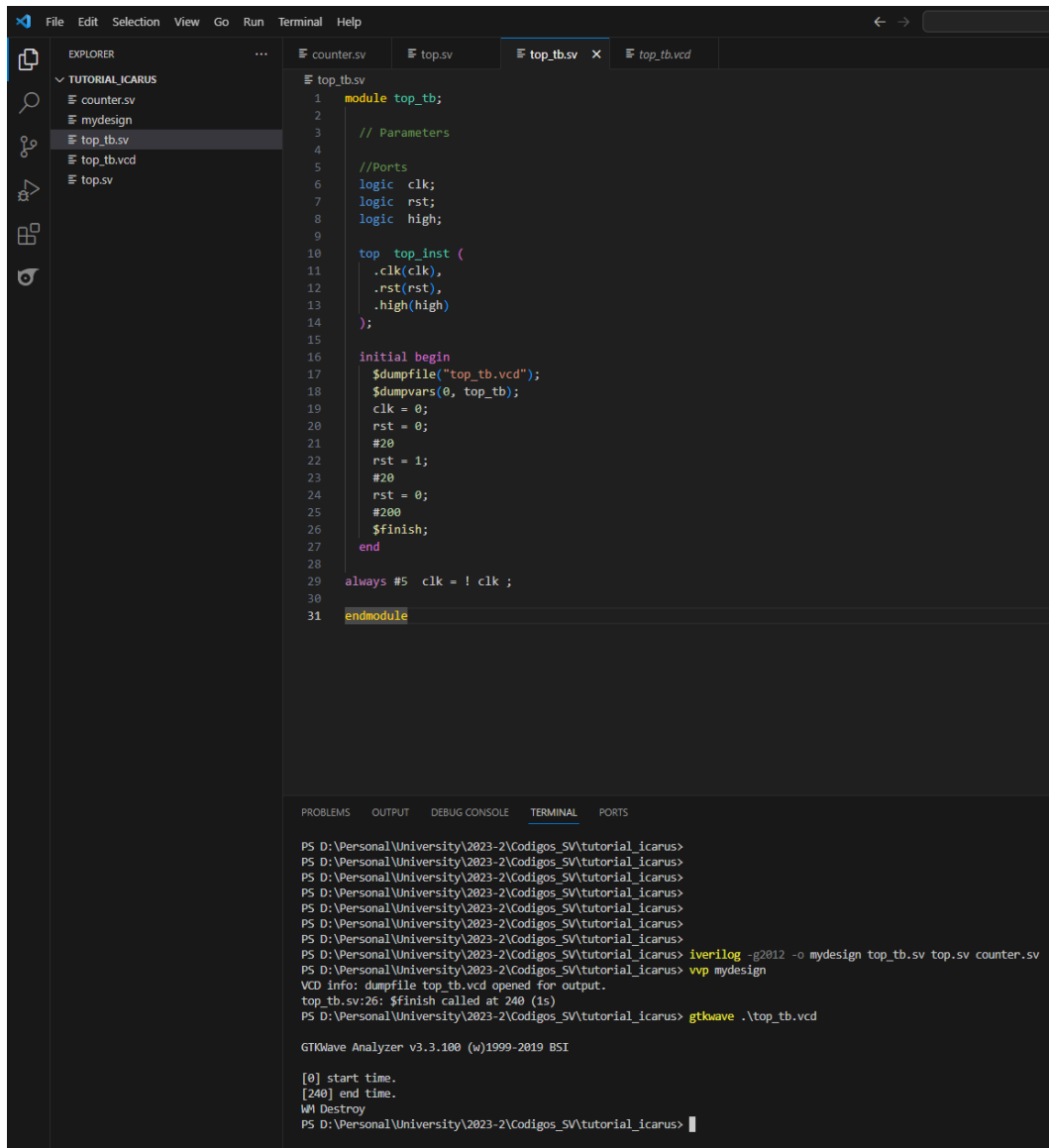


Figura 3.1: Vista del flujo de trabajo en Visual Studio Code

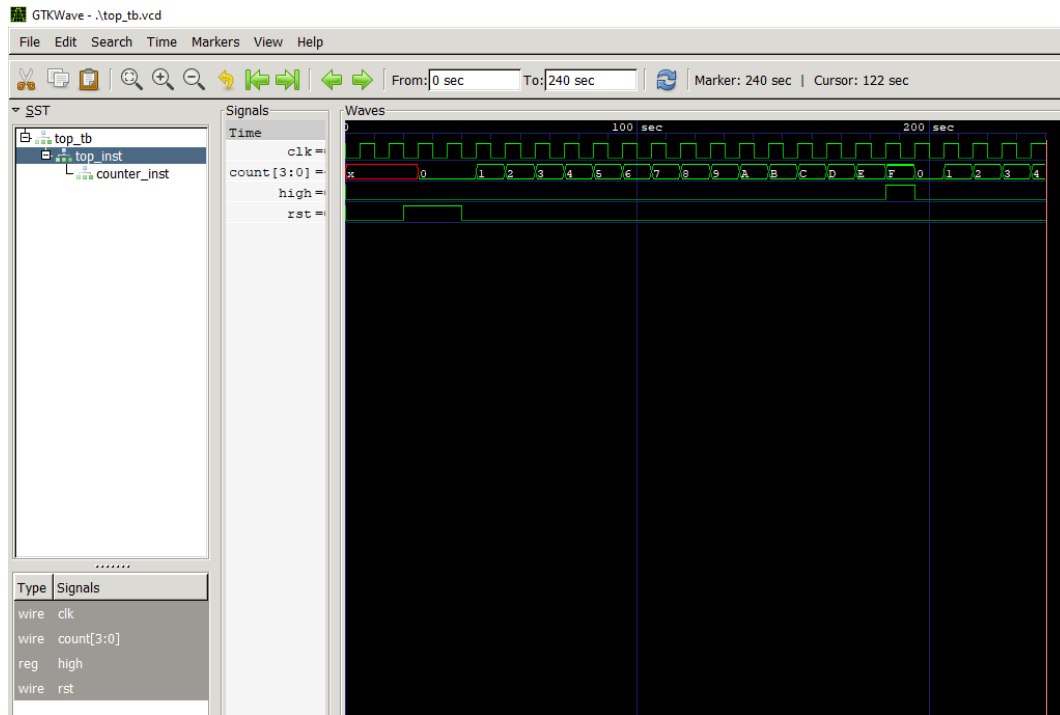


Figura 3.2: Formas de onda vistas por GTKWave

Con la introducción de este nuevo flujo de trabajo debe quedar claro que Vivado sigue siendo la herramienta principal del curso y la única con la posibilidad de realizar todos los pasos del diseño en FPGA en el contexto específico de la asignatura. Se espera que en algún futuro próximo aparezcan otras herramientas que puedan integrarse como alternativas al flujo de trabajo de los alumnos y el *staff*, cuyas funcionalidades sean mas cercanas a las de Vivado.

### 3.3. Actividades con interfaz VGA para procesamiento de imágenes

La experiencia de diseño de una interfaz VGA para desplegar imágenes en un computador externo ha sido utilizada comúnmente como una actividad de cierre del curso, permitiendo integrar todo lo aprendido a lo largo del curso en un dispositivo de uso común y con resultados visualmente atractivos. En la última ocasión que se trabajó con generación de vídeo en la asignatura, se planteó una actividad que consistía en un sistema que permitía desplegar en pantalla una imagen que estaba almacenada en la memoria interna de la FPGA. La imagen almacenada puede desplegarse en pantalla sin modificaciones, o bien puede pasar por distintos filtros de imágenes que se activan o desactivan utilizando los switches de la FPGA. Los filtros incluyen intercambio de canales de colores, escala de grises, dithering, y Sobel. Además, se debe permitir al usuario cambiar la imagen a desplegar en tiempo de ejecución, utilizando el puerto UART para enviar una nueva imagen a la memoria de la FPGA desde un computador externo. Esta actividad se planteó como un modo proyecto, entregando especificaciones generales y archivo base reducidos, y dando libertad a los estudiantes para que investigaran, experimentaran, y tomaran libremente decisiones de diseño sobre como implementar el sistema.

Si bien se observó que la actividad resultó bastante motivante para los estudiantes debido a lo llamativo de los resultados visuales que incentivaban a experimentar con distintas imágenes y algoritmos, también se notaron dificultades por parte de los estudiantes al lidiar con múltiples problemas de distinta índole. Por ejemplo, la mayoría de los estudiantes del curso aún no dominan los conceptos básicos asociados al procesamiento de imágenes, por lo cual debían estudiar por su cuenta detalles de los algoritmos al mismo tiempo de lidiar con los desafíos de la implementación en *hardware* de un sistema con un nivel de complejidad que no habían visto previamente. Esto generaba en un inicio bastante frustración debido a que en el caso que el sistema no funcionara, era difícil identificar si era un problema del algoritmo, de la descripción del *hardware*, o de la comunicación con el computador externo o con el monitor de visualización. Esto se veía amplificado en muchos casos debido a que los estudiantes aún no asimilaban la disciplina de modularización y el *dividir y conquistar*. En muchos casos, los estudiantes se enfocaban en buscar ayuda en internet, reutilizando algunos módulos disponibles y haciéndolos funcionar mediante prueba y error, sin entender el como funcionaban. Producto del número de posibles soluciones y errores que surgían de estas, la asistencia por parte del *staff* se veía dificultada, y finalmente se resumía en intentar corregir rápidamente los errores de los alumnos, sin quedar mucho tiempo para entregar feedback. En aspectos de evaluación, también se hacía difícil realizar una revisión sistemática del diseño, por lo que la evaluación de la actividad se enfocaba principalmente en los aspectos visuales del resultado desplegado en pantalla más que en los aspectos asociados al diseño y descripción del sistema.

Desde la experiencia y observaciones previas, y considerando que VGA presenta una actividad interesante y llamativa, se planteó el objetivo de aplicar la metodología descrita en [4] para diseñar nuevas actividades orientadas a procesamiento de imágenes de una forma estructurada y gradual, acotándose el espacio de resultados esperados para facilitar la revisión y verificación de que las actividades aporten efectivamente a los resultados de aprendizaje del curso. En específico, mediante estas actividades se apunta a reforzar conceptos fundamentales de diseño digital como son el diseño modular y uso de IPs, además de introducir conceptos que no se han tratado previamente como el análisis de *timing* estático.

### 3.3.1. Actividad base

Para el diseño de nuevas actividades, se consideró como punto de partida la experiencia desarrollada en la versión 2019 del curso, cuyas partes principales se resumen a continuación.

La Figura 3.3 muestra el esquemático del conector VGA disponible en la tarjeta Nexys4DDR y Nexys A7. La intensidad de cada canal de color (RED, GREEN, BLUE) se representa por medio de 4 bits. Adicionalmente, se cuenta con las señales de sincronismo horizontal (HS) y vertical (VS). Todas las actividades presentadas en adelante consideran esta representación.

El diagrama de la Figura 3.4 muestra una representación de alto nivel de un sistema de procesamiento de imágenes que aplica filtros en cascada a una imagen recibida desde un sistema externo. El sistema almacena la imagen en una memoria interna con *configuración de doble puerta*, con puertos independientes para la lectura y escritura de datos (cada una con un reloj independiente). El puerto de escritura está conectado a una interfaz UART que recibe y almacena los píxeles de una imagen proveniente desde un sistema externo. El puerto de lectura es controlado mediante una interfaz VGA, la cual lee en forma continua el contenido del buffer para ser procesado por los filtros y mostrados en el display. La interfaz VGA muestra

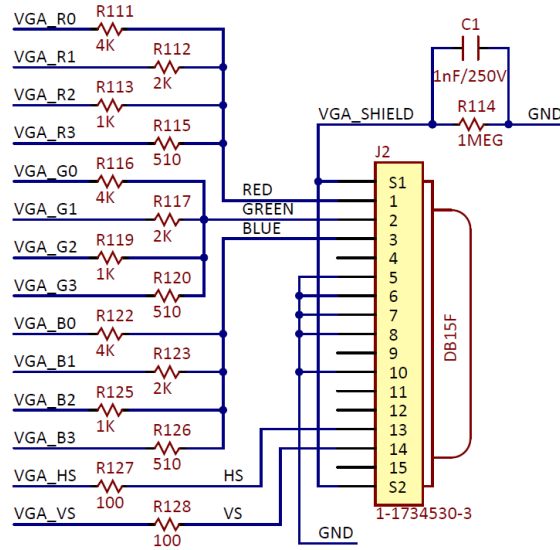


Figura 3.3: Esquemático de las conexiones del puerto VGA

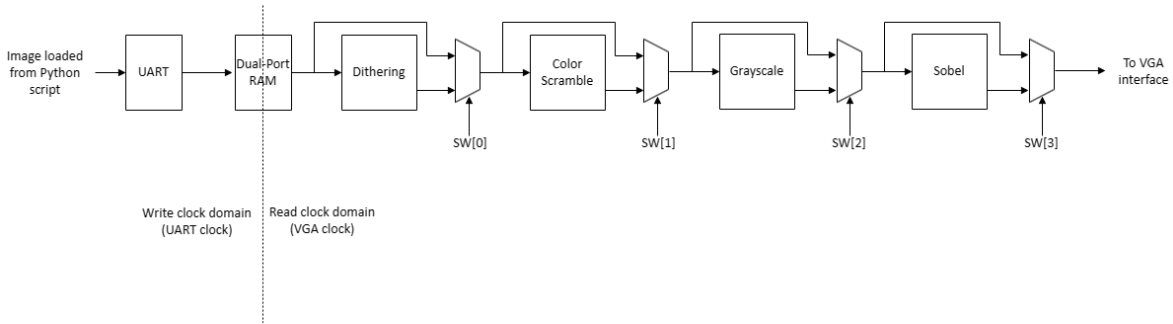


Figura 3.4: Diagrama de alto nivel de dispositivos distribuidos conectados por interfaz UART.

en todo momento el contenido del buffer y los interpreta como píxeles, sin saber si el buffer tiene almacenada una imagen válida o no.

La funcionalidad de cada uno de los filtros mostrados se describe en forma general a continuación:

- *Color scramble*: Intercambia los valores de los canales de color de los píxeles. Por ejemplo, el canal R que va a la interfaz VGA puede transmitir el canal R original, el canal G, el canal B, o estar apagado, de acuerdo a la señal de selección del multiplexor conectado a la interfaz VGA. Este comportamiento es similar para cada uno de los canales, donde cada uno puede transmitir cualquier canal del píxel de entrada o estar apagado de acuerdo a la configuración de los switches.
- *Grayscale*: Convierte una imagen a color en RGB a una imagen en escala de grises. Para esto se debe hacer una manipulación aritmética con los valores de los canales de color de cada píxel.
- *Dithering*: Filtro que permite mejorar la presentación de una imagen en un sistema con paleta de colores reducida. La técnica consiste en aplicar una especie de ruido a una imagen para dar la impresión de contar con una mayor número de tonalidades de las

que efectivamente están disponibles en el sistema. Por ejemplo, en un sistema que solo soporta blanco y negro (un bit de resolución de color), es posible aparentar distintos niveles de grises alternando píxeles blancos y negros con distinta separación entre ellos. Para una imagen a color el concepto es similar. Por ejemplo, al mostrar una imagen de 24 bits de color en un display que solo cuenta con 12 bits, se pueden utilizar los bits mas significativos para definir el color mas cercano en la paleta reducida – esto se conoce como *thresholding* – y usar los bits menos significativos para “redondear” el color (intuitivamente sería como aproximar un número con decimales a su parte entera).

- *Sobel*: Filtro para detección de bordes en una imagen. Este filtro calcula el valor de cada píxel por medio de una suma ponderada de sus píxeles vecinos en el eje horizontal y vertical. Esto quiere decir que para calcular el valor de un píxel, se necesita saber el valor de los píxeles vecinos de la línea anterior y también de la línea siguiente.

Cada uno de los filtros anteriores se aplican en cascada, y pueden ser activados o desactivados mediante los switches.

### 3.3.2. Definición de nuevas actividades

De acuerdo con el feedback recolectado durante la última realización de la actividad descrita previamente, y basándose en lo observado al intentar repetir el trabajo, se observaron los siguientes puntos que resultaron relevantes para definir la formulación de las nuevas actividades a especificar:

- En muchos casos se observó que a pesar de las indicaciones entregadas, muchos estudiantes no aplicaban los principios de *divide and conquer*, tendiendo a conectar todos los módulos para hacer pruebas con el sistema completo, en lugar de diseñar y validar cada módulo por separado antes de realizar la integración. Como era de esperarse, en la mayoría de los casos las primeras pruebas con el sistema completo no funcionaba como se esperaba, observándose variados comportamientos como que la imagen no se desplegaba, se desplegaba deformada, o solo se veían patrones aleatorios. En estos casos, se hacía muy difícil identificar cual era la fuente del problema, incluso con el apoyo del *staff*. Además, siendo el uso de recursos bastante superior a las actividades que los estudiantes han trabajado previamente, los tiempos de síntesis e implementación pueden estar en el orden de decenas de minutos en un computador de gama media <sup>1</sup>, por lo que una exploración tipo prueba y error resultaba bastante ineficiente. Normalmente, y luego de haber invertido bastante tiempo sin obtener frutos, los estudiantes se daban cuenta que el mejor plan de acción era rediseñar el sistema utilizando un enfoque modular con integración incremental.
- Los estudiantes no tienen conocimientos previos sobre como implementar los filtros y tienden a tratar de reutilizar códigos ya hechos en vez de darse el trabajo de entender como funcionan. Dada la variedad de implementaciones, cada uno tiene sus propios problemas y se pierde mucho tiempo dando feedback. Pierden tiempo entendiendo el código en vez de trabajar en el diseño, desviando el foco de los principales resultados de aprendizaje del curso. Como solución a lo anterior, se propone preparar guías con requerimientos e indicaciones precisas sobre la forma de implementar los filtros, buscando

---

<sup>1</sup>En un computador de gama alta con un procesador i5 11600k, tarjeta gráfica rtx 3070 y 16gb de ram, el proceso completo hasta la generación del bitstream tomaba alrededor de 5 minutos

acotar el espacio de soluciones a las que pudiesen llegar los estudiantes, acotando a su vez el espacio de errores que pueden manifestarse para hacer más eficiente la revisión de la guía por parte del *staff*.

- Al reproducir las actividades anteriores para este trabajo, se observó que para los requerimientos especificados de resolución y profundidad de color, y dada la complejidad de algunos filtros y las variadas formas de describirlos, hay una alta probabilidad de obtener problemas de *timing* durante la implementación debido a la alta latencia del sistema. Por ejemplo, la ponderación de los canales de colores para transformar la imagen a color a escala de grises o el uso de un IP para calcular la raíz cuadrada en el algoritmo Sobel puede resultar en un retardo excesivo en los paths, los cuales son los posibles recorridos de las señales a través del sistema, en este caso entre dos registros, afectando los requerimientos de *timing* al operar a la frecuencia requerida para la pixel clock, el cual es el reloj que demarca el cambio de un píxel a otro en la pantalla. Resulta interesante notar que en las pruebas realizadas, la implementación de un sistema con slack negativo, donde la latencia de uno o varios paths supera un límite fijado por la frecuencia del reloj y propiedades del chip físico generando problemas de requerimientos de *timing*, no muestra efectos perceptibles en la calidad del vídeo desplegado. Esto indica que, dado que la evaluación de la experiencia se ha basado en el resultado visual, en muchos casos se puede haber considerado correcto un sistema que tenía fallas fundamentales.

Considerando los puntos anteriores, se decidió reestructurar el trabajo particionando la actividad original en múltiples actividades de complejidad incremental, cada una enfocada en reforzar aspectos específicos de los resultados de aprendizaje esperados del curso y que permitan una evaluación sistemática por parte del *staff*.

Las actividades consideran las siguientes etapas:

- *Introducción a interfaz VGA*: Se introduce al concepto de VGA utilizando el simulador online 8bitworkshop [15] para luego pasar a una implementación en la FPGA de las mismas actividades introductorias realizadas en el simulador.
- *Filtros de procesamiento de imágenes*: Se introduce a los alumnos a los filtros de procesamiento de imágenes, pidiéndoles que desarrollen el filtro grayscale, color swapper, dithering y Sobel. A diferencia de versiones anteriores del curso en que se pedía a los estudiantes que averiguaran sobre los algoritmos, se propone entregar información más concreta sobre la implementación de cada uno, de manera que los estudiantes se enfoquen en los aspectos de descripción e implementación en lugar de la matemáticas asociada al algoritmo. Además, al acotar el espacio de soluciones se reduce la variabilidad de problemas que puedan surgir, lo que permitirá hacer más eficiente el trabajo del *staff*.
- *Resolución de problemas de timing*: Se pide cambiar la resolución de la imagen provocando que surjan problemas de requerimientos de *timing*, para luego guiar a los alumnos en cómo solucionarlos al hacer sus filtros mas eficientes, luego de esto para finalizar la experiencia se les pide que añadan la interfaz UART habilitando la posibilidad de cargar imágenes en tiempo de ejecución.

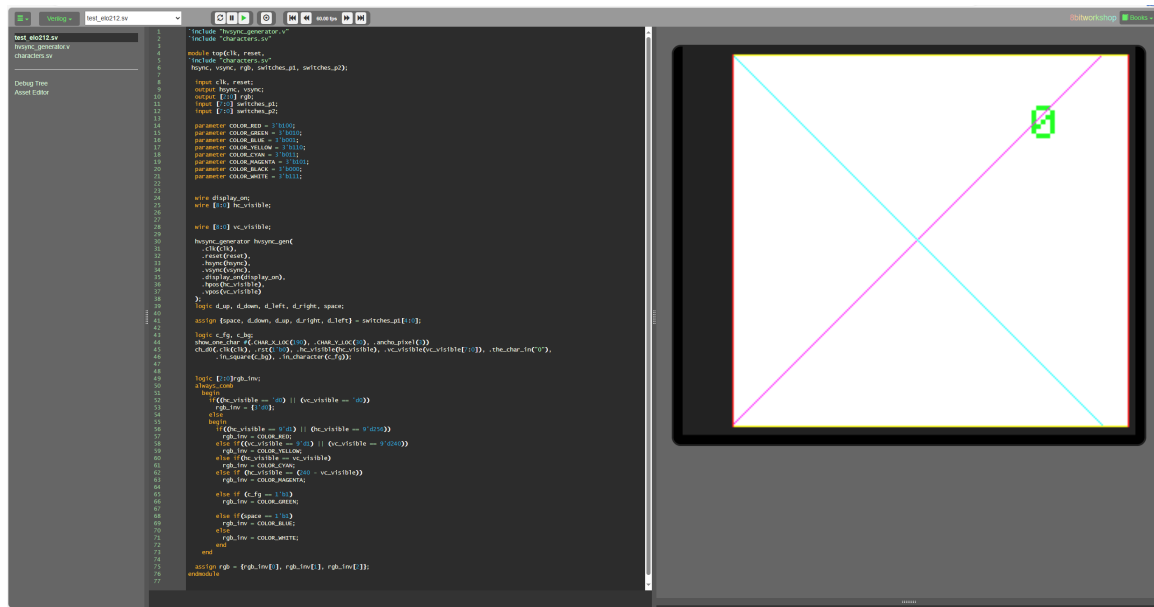


Figura 3.5: Vista de las actividades en el 8bitworkshop IDE

### 3.3.3. Introducción a interfaz VGA

En esta guía se busca introducir a los alumnos de manera orientada al protocolo VGA y al uso de IP, enseñándoles a utilizar el driver y la IP para generar el *pixel clock*, el cual dicta la frecuencia a la que se dibujan los píxeles en pantalla y se deriva a partir de la resolución en píxeles y la tasa de refresco de los cuadros o *frames* en la pantalla.

Como actividad previa los alumnos deberán ver una cápsula de vídeo, generada en el periodo de enseñanza remota por el profesor Mauricio Solís, en donde se explica que es el protocolo VGA, cómo funciona y cómo utilizar el driver que se les entrega. La primera actividad que deberán realizar es la de ingresar al simulador online 8bitworkshop [15] que les permitirá editar código en SystemVerilog y ver los efectos en una pantalla simulada. Luego, la cápsula muestra cómo utilizar el simulador web para dibujar figuras básicas y utilizar un módulo el cual les permitirá dibujar texto en la pantalla. El simulador está restringido a usar 3 bits de profundidad para la paleta de colores RGB (1 bit por canal de color), y el pixel clock no se especifica explícitamente en el código ya que se define automáticamente al cambiar las opciones de visualización por medio de los menús. Para efectos del ejemplo, se reduce la resolución de pantalla a 256x240 píxeles. Si bien es posible utilizar una resolución mayor, el uso de recursos utilizados para la simulación es considerable, lo cual puede afectar el desempeño en computadores personales de gama media y baja.

La Figura 3.5 muestra un ejemplo de una de las actividades mostradas, correspondiente a mostrar dos líneas diagonales de distintos colores, y dibujar un número en pantalla utilizando los archivos base. Todos los archivos utilizados en esta actividad y en las subsiguientes se encuentran en el repositorio en conjunto a una archivo readme que documenta el uso de cada uno.

Una vez los estudiantes hayan logrado un dominio básico sobre el protocolo y el uso del driver a través del simulador, se propone aplicar lo aprendido para implementar las actividades realizadas

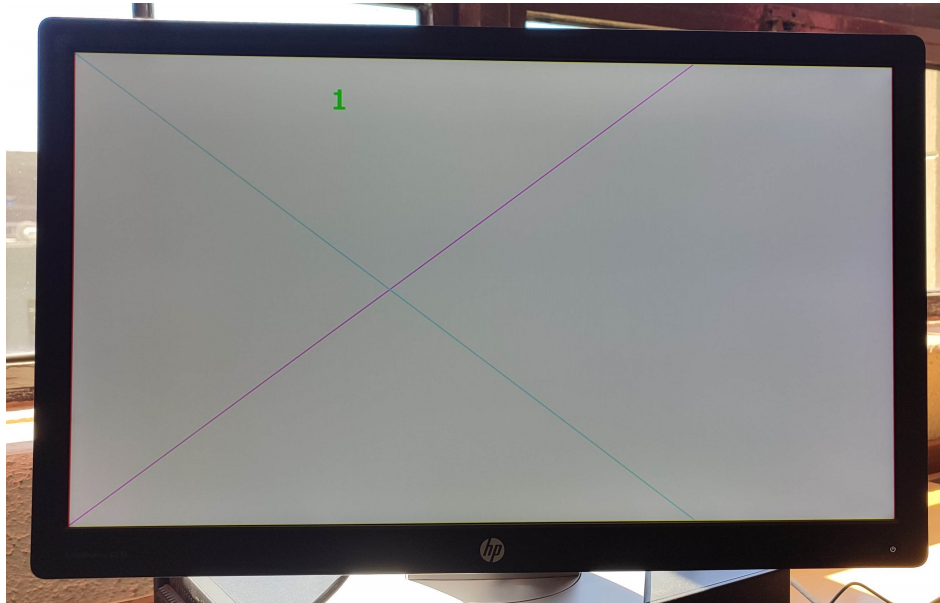


Figura 3.6: Actividades del simulador replicadas en la FPGA y mostradas en un monitor

anteriormente en el simulador en la tarjeta de desarrollo, para así visualizar los resultados en un monitor real (debe tener interfaz VGA). Para esto, se preparó una nueva cápsula de vídeo que describe algunos aspectos técnicos asociados a la descripción e implementación del driver en *hardware*, los cuales difieren en algunos aspectos con respecto a lo utilizado en el simulador. Entre los cambios necesarios se incluyen un ajuste en la resolución utilizada para la pantalla VGA, que para el caso de la pantalla física corresponderá a 640x480@75Hz. Esta nueva resolución y número de cuadros por segundo puede ser introducida a una hoja de cálculos provista en el repositorio, la cual indicara los nuevos parámetros a utilizar en el driver y también el pixel clock requerido, que en este caso resulta ser de 30.75MHz<sup>2</sup>. Para implementar esta frecuencia de reloj en *hardware*, se hace necesario utilizar un IP especializado para manejo de reloj o *Clocking Wizard*, lo cual se omitía en el caso del simulador debido a que se realizaba una emulación por *software*. Además, es necesario invertir la polaridad de las señales de sincronización horizontal y vertical del driver. Todos estos cambios requieren ajustes menores al driver, los que se encuentran debidamente documentados en la carpeta correspondiente y en la cápsula de esta actividad en el repositorio de material.

La Figura 3.6 muestra como ejemplo el resultado de visualización en un monitor físico para la misma actividad mostrada en la Figura 3.5 implementada en el simulador. Se puede notar que la relación de aspecto es distinta, donde la del simulador es cercana a un cuadrado y la de la pantalla es la mas usual 4:3. La diferencia en la razón de aspecto también se puede notar en la distinta posición de las líneas diagonales y el número desplegado relativas a la pantalla.

Una vez replicadas las actividades de simulación y entendidos los principios de funcionamiento de la interfaz, se pueden proponer actividades adicionales para que los estudiantes experimenten con la generación de vídeo, como por ejemplo dibujar un cuadrado de color en movimiento, desplegar texto de distinto largo, etc.

Al finalizar esta guía, se espera que los alumnos dominen los principios del protocolo VGA y

---

<sup>2</sup>Notar que estos valores son referenciales, y pueden variar entre monitores de distintos fabricantes.

tengan un entendimiento técnico del driver para manejar un display y el manejo de códigos para desplegar figuras básicas en pantalla. Con esta base, se puede pasar a actividades más complejas, como la visualización de imágenes en pantalla y la aplicación de distintos filtros a estas.

### 3.3.4. Filtros de procesamiento de imágenes.

El objetivo principal de esta guía es extender la integración de componentes, agregando la posibilidad al sistema de vídeo de desplegar imágenes de relación de aspecto 4:3 y profundidad de color de 12 bits (4 bits por canal) guardadas en un bloque de memoria, además de la implementación de varios filtros de imagen que permitan modificar la imagen original.

Se considera nuevamente una resolución de 640x480@75Hz. Para implementar el bloque de memoria para almacenar los píxeles de la imagen, es necesario utilizar nuevamente el catálogo de IP para generar un bloque basado en primitivas Block Ram (BRAM). Por simplicidad, para esta actividad se asume que la imagen estará precargada en memoria al momento de inicializar el sistema, siendo solo necesario implementar la lógica para leer síncronamente los datos de los píxeles que serán desplegados en la pantalla. La imagen puede ser precargada mediante un archivo .coe, que permite establecer valores iniciales en las celdas de memoria que quedan incrustados en el bitfile, por lo que la imagen se despliega automáticamente al cargar el bitfile. Esta simplificación permite que los estudiantes se enfoquen en aspectos de procesamiento y visualización durante esta etapa, sin preocuparse aún de la lógica de carga de datos en memoria a tiempo de ejecución. En el repositorio de la actividad se incluye un *script* de python para generar un archivo .coe a partir de una imagen de entrada, permitiendo que los alumnos prueben con las imágenes que deseen.

## BRAM

En esta experiencia se considera implementar una memoria capaz de almacenar una imagen con profundidad de color de 24 bits (8 bits por color). Si bien la tarjeta de desarrollo limita la profundidad de color a 12 bits (ver Figura 3.3), los bits adicionales se utilizarán para aplicaciones de filtros de imágenes previo a que sean desplegadas en la pantalla.

En el manual de referencia de la Nexys A7 100T [16] se puede ver que el chip FPGA cuenta con una capacidad total de 4860 Kbits de memoria BRAM. Considerando que se requieren guardar píxeles de 24 bits, es posible calcular el número total de píxeles que se pueden guardar como:

$$N_{\text{píxeles}} = \left\lfloor \frac{\text{total bits of BRAM}}{\text{bits por pixel}} \right\rfloor = \left\lfloor \frac{4860000}{24} \right\rfloor = 202500 \text{ píxeles} \quad (3.1)$$

Al distribuir el total de píxeles en una imagen de relación de aspecto ancho(W):alto(H) de 4:3, se puede calcular la resolución para la dimensión horizontal  $W$  como

$$W \times H = W \times \frac{3}{4}W = 202500 \Rightarrow w = 519 \quad (3.2)$$

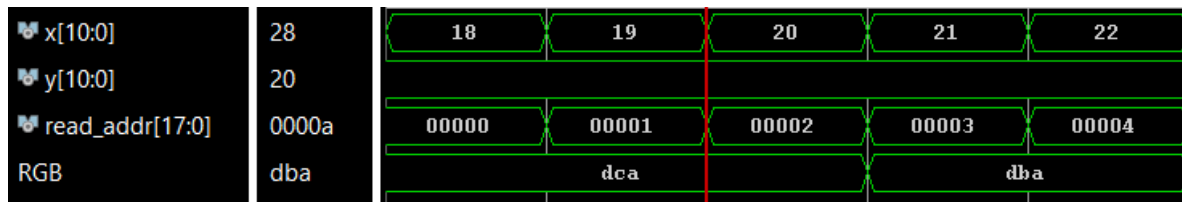


Figura 3.7: BRAM siendo accedida dos ciclos antes de mostrar los píxeles en pantalla

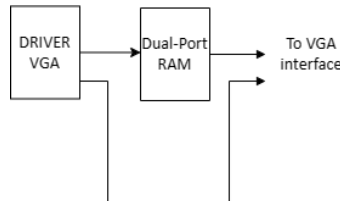


Figura 3.8: Sistema con la inclusión de la BRAM

De lo anterior se obtiene el número máximo de píxeles para una línea horizontal. Sin embargo, esto genera un número de píxeles para la dimensión vertical que no es un número entero. Por lo tanto, se busca el mayor número de píxeles horizontales que se relacione a un número entero de píxeles verticales. Con este criterio, se llega que la dimensión de la imagen a guardar en la memoria es de  $516 \times 387$  píxeles.

Con la información derivada, es posible utilizar el IP catalog para generar el bloque de RAM adecuado para almacenar la imagen. La Figura 3.9 muestra los pasos necesarios para generar este bloque. Respecto al archivo COE, este debe cargarse en la pestaña *Other Options*. En el caso de que los alumnos quisieran generar sus propios archivos COE, deberán utilizar el *script* de python que se generó para este propósito cuyas instrucciones de uso se encuentran en el mismo código.

Un detalle de implementación que es importante conocer es que esta configuración de BRAM tiene una latencia de lectura de dos ciclos de reloj, por lo que un píxel debe ser accedido dos ciclos antes de cuando se vaya a mostrar en pantalla. Por ejemplo, asumamos que la palabra almacenada en la dirección 0x1 de la memoria corresponde al píxel que debe ser mostrado en la posición (21,20) de la pantalla. Considerando la latencia de lectura de dos ciclos, el bus de direcciones de la memoria debe tomar el valor 0x1 cuando el contador de píxeles este en la posición (19,20), de manera que el valor del píxel este disponible cuando el contador de píxeles este efectivamente en la posición (21,20). Esta situación puede observarse en el diagrama temporal de la Figura 3.7, donde el píxel almacenado en la dirección 0x1 tiene valor 0xdba.

En la Figura 3.8 se puede ver un diagrama del sistema hasta el momento con la inclusión de la BRAM, y en la Figura 3.10 se puede ver una imagen cargada en la FPGA y siendo mostrada en una pantalla. Cabe destacar que el reporte de utilización indica que cerca del 100 % de los bloques BRAM efectivamente se encuentran utilizados para este diseño.

Una vez verificada la funcionalidad para mostrar una imagen a colores, se procede a implementar filtros de imágenes de complejidad incremental.

Component Name blk\_mem

| Basic                                                                                                     | Port A Options | Port B Options | Other Options | Summary |
|-----------------------------------------------------------------------------------------------------------|----------------|----------------|---------------|---------|
| Interface Type Native <input type="checkbox"/> Generate address interface with 32 bits                    |                |                |               |         |
| Memory Type True Dual Port RAM <input type="checkbox"/> Common Clock                                      |                |                |               |         |
| <b>ECC Options</b>                                                                                        |                |                |               |         |
| ECC Type No ECC                                                                                           |                |                |               |         |
| <input type="checkbox"/> Error Injection Pins Single Bit Error Injection                                  |                |                |               |         |
| <b>Write Enable</b>                                                                                       |                |                |               |         |
| <input type="checkbox"/> Byte Write Enable                                                                |                |                |               |         |
| Byte Size (bits) 9                                                                                        |                |                |               |         |
| <b>Algorithm Options</b>                                                                                  |                |                |               |         |
| Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information. |                |                |               |         |
| Algorithm Minimum Area                                                                                    |                |                |               |         |
| Primitive 8kx2                                                                                            |                |                |               |         |

(a) Pestaña “Basic ”

Component Name blk\_mem

| Basic                                                                                                        | Port A Options | Port B Options | Other Options | Summary |
|--------------------------------------------------------------------------------------------------------------|----------------|----------------|---------------|---------|
| <b>Memory Size</b>                                                                                           |                |                |               |         |
| Write Width 24 Range: 1 to 4608 (bits)                                                                       |                |                |               |         |
| Read Width 24                                                                                                |                |                |               |         |
| Write Depth 199692 Range: 2 to 1048576                                                                       |                |                |               |         |
| Read Depth 199692                                                                                            |                |                |               |         |
| Operating Mode Write First Enable Port Type Always Enabled                                                   |                |                |               |         |
| <b>Port A Optional Output Registers</b>                                                                      |                |                |               |         |
| <input checked="" type="checkbox"/> Primitives Output Register <input type="checkbox"/> Core Output Register |                |                |               |         |
| <input type="checkbox"/> SoftECC Input Register <input type="checkbox"/> REGCEA Pin                          |                |                |               |         |
| <b>Port A Output Reset Options</b>                                                                           |                |                |               |         |
| <input type="checkbox"/> RSTA Pin (set/reset pin) Output Reset Value (Hex) 0                                 |                |                |               |         |
| <input type="checkbox"/> Reset Memory Latch Reset Priority CE (Latch or Register Enable)                     |                |                |               |         |
| <b>READ Address Change A</b>                                                                                 |                |                |               |         |
| <input type="checkbox"/> Read Address Change A                                                               |                |                |               |         |

(b) Pestaña “Port A Options ”

Component Name blk\_mem

| Basic                                                                                                        | Port A Options | Port B Options | Other Options | Summary |
|--------------------------------------------------------------------------------------------------------------|----------------|----------------|---------------|---------|
| <b>Memory Size</b>                                                                                           |                |                |               |         |
| Write Width 24                                                                                               |                |                |               |         |
| Read Width 24                                                                                                |                |                |               |         |
| Write Depth : 199692                                                                                         |                |                |               |         |
| Read Depth : 199692                                                                                          |                |                |               |         |
| Operating Mode Write First Enable Port Type Always Enabled                                                   |                |                |               |         |
| <b>Port B Optional Output Registers</b>                                                                      |                |                |               |         |
| <input checked="" type="checkbox"/> Primitives Output Register <input type="checkbox"/> Core Output Register |                |                |               |         |
| <input type="checkbox"/> SoftECC Output Register <input type="checkbox"/> REGCEB Pin                         |                |                |               |         |
| <b>Port B Output Reset Options</b>                                                                           |                |                |               |         |
| <input type="checkbox"/> RSTB Pin (set/reset pin) Output Reset Value (Hex) 0                                 |                |                |               |         |
| <input type="checkbox"/> Reset Memory Latch Reset Priority CE (Latch or Register Enable)                     |                |                |               |         |
| <b>READ Address Change B</b>                                                                                 |                |                |               |         |
| <input type="checkbox"/> Read Address Change B                                                               |                |                |               |         |

(c) Pestaña “Port B Options ”

Component Name blk\_mem

| Basic                                                                                                                               | Port A Options | Port B Options | Other Options | Summary |
|-------------------------------------------------------------------------------------------------------------------------------------|----------------|----------------|---------------|---------|
| Pipeline Stages within Mux 0 Mux Size: 49x1                                                                                         |                |                |               |         |
| <b>Memory Initialization</b>                                                                                                        |                |                |               |         |
| <input checked="" type="checkbox"/> Load Init File                                                                                  |                |                |               |         |
| Coe File /Material_Memoria_ELO212/VGA_RELATED/img/mem.coe <input type="button" value="Browse"/> <input type="button" value="Edit"/> |                |                |               |         |
| <input type="checkbox"/> Fill Remaining Memory Locations                                                                            |                |                |               |         |
| Remaining Memory Locations (Hex) 0                                                                                                  |                |                |               |         |
| <b>Structural/UniSim Simulation Model Options</b>                                                                                   |                |                |               |         |
| Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.                           |                |                |               |         |
| Collision Warnings All                                                                                                              |                |                |               |         |
| <b>Behavioral Simulation Model Options</b>                                                                                          |                |                |               |         |
| <input type="checkbox"/> Disable Collision Warnings <input type="checkbox"/> Disable Out of Range Warnings                          |                |                |               |         |

(d) Pestaña “Other Options ”

Figura 3.9: Configuración de la BRAM

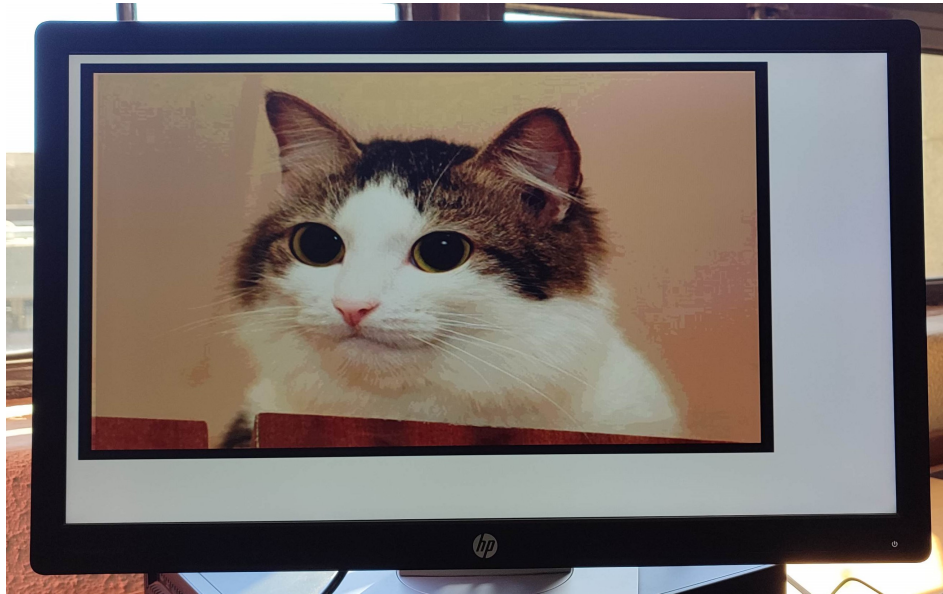


Figura 3.10: Imagen original mostrada en pantalla

### Color Scrambler

Habiendo logrado la visualización de la imagen a color, se propone la implementación del filtro color scrambler, el cual se debe implementar siguiendo las indicaciones de la Figura 3.11. Este filtro es relativamente sencillo y solo requiere operaciones de multiplexación, siendo el objetivo que los estudiantes puedan experimentar con los canales de colores y ver algunos efectos visuales. Por ejemplo, la Figura 3.12 muestra los efectos de aislar cada canal de color, que puede realizarse haciendo un *bypass* de uno de los canales de color, dejando el valor de los otros dos canales en 0.

En la Figura 3.13 se puede ver un diagrama del sistema hasta el momento con la inclusión del color scrambler.

### Grayscale

El filtro de grayscale hace uso de una suma ponderada de los valores de intensidad de los canales R, G y B, la cual es asignada por igual a todos los canales a la salida para generar una imagen en la escala de grises. Debido a que todos los canales toman el mismo valor, solo se tienen 4 bits para representar un total de 16 tonalidades en escala de grises. En primera instancia, se propone utilizar la siguiente fórmula para obtener la tonalidad de gris a partir de los valores RGB:

$$gray = 0,2126 * R + 0,7152 * G + 0,0722 * B \quad (3.3)$$

la cual representa la intensidad con la que nuestros ojos perciben cada canal en el espacio de color sRGB [17], proporcionando un resultado que debiese ser preciso frente al ojo humano. Una observación interesante es que al ingresar esta operación en SystemVerilog mediante el siguiente código:

```
assign pixel_out = 0.2126*R + 0.7152*G + 0.0722*B;
```

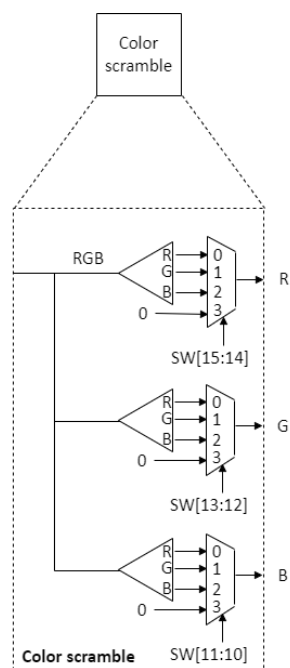
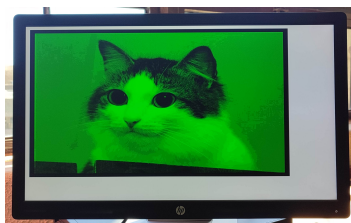


Figura 3.11: Funcionamiento interno del color scrambler



(a) Canal rojo de la imagen



(b) Canal verde de la imagen



(c) Canal azul de la imagen

Figura 3.12: Canales de color de la imagen separados utilizando el filtro color scrambler

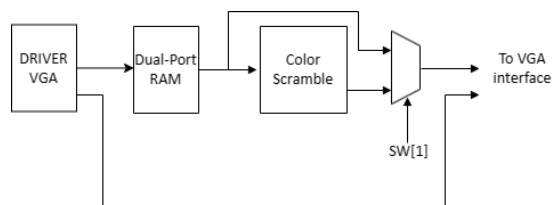


Figura 3.13: Sistema con la inclusión del color scrambler

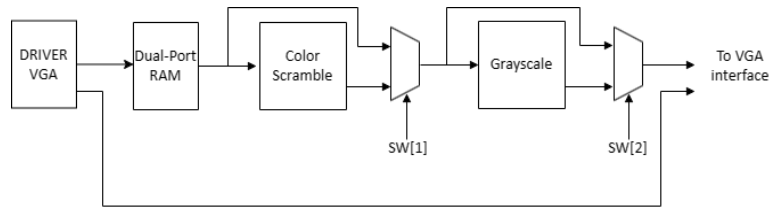
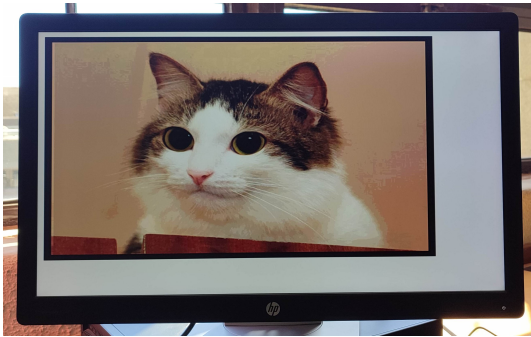


Figura 3.14: Sistema con la inclusión del grayscale



(a) Imagen original



(b) Imagen transformada a escala de grises

Figura 3.15: Comparación de la imagen normal y el filtro grayscale

la herramienta de síntesis Vivado arroja el siguiente error:

“[Synth 8-502] non-constant real-valued expression is not supported”

el cual indica que Vivado no soporta operaciones con números reales para síntesis lógica. Esto resulta interesante para reforzar algunas diferencias fundamentales entre HDLs y lenguajes de programación tradicional, además de los cuidados y restricciones a considerar en sistemas que se implementarán en *hardware*. Una forma alternativa de ingresar esta expresión es:

```
assign pixel_out = (1063 * R + 3576 * G + 361 * B)/5000;
```

la cual si sintetiza correctamente. Se hace notar que la expresión planteada apunta a representar fielmente la expresión matemática; sin embargo, como se verá más adelante, no es adecuada para su implementación en *hardware* debido a que impondrá limitaciones en términos de *timing* y desempeño cuando se requiera usar resoluciones más altas, haciendo necesario el considerar otros tradeoffs.

En la Figura 3.14 se puede ver la representación de alto nivel del sistema luego de la adición del módulo grayscale, y en la Figura 3.15 se puede ver una comparación de una imagen antes y después de activar el filtro de conversión a escala de grises.

## Dithering

Este filtro aplica ruido a una imagen de manera controlada para mejorar la presentación de esta en una pantalla con una paleta de colores de menor tamaño que la resolución original. En el caso de esta experiencia, se propone la implementación del algoritmo *ordered dithering*, este recibe la información de color del píxel original codificada con 8 bits por color junto con la

posición del píxel, para así calcular el valor del píxel a ser mostrado en pantalla utilizando solo 4 bits por color .

En el caso de esta experiencia la imagen cargada en la BRAM posee una profundidad de color de 24 bits, con 8 bits por color, pero nuestra salida solo cuenta con 4 bits por color (ver Figura 3.3). Aplicando el algoritmo de ordered dithering, para cada canal del píxel original, se utilizan los 4 bits más significativos para determinar un valor auxiliar para la magnitud del canal de color, y se usan los 4 bits menos significativas para determinar un ajuste a la magnitud auxiliar. Específicamente, los 4 bits menos significativos se comparan a un valor de *threshold* preconfigurado. En caso de que el valor formado por los 4 bits menos significativos sea mayor al *threshold*, entonces se incrementa en una unidad el valor auxiliar definido por los cuatro bits más significativos del canal de color; en caso contrario, el valor auxiliar se mantiene sin cambios. El valor del *threshold* cambia dependiendo de la posición del píxel en la imagen. Para esto, se define una matriz cuadrada llamada *threshold map*, cuya dimensión debe ser potencia de 2. Como ejemplo, se propone el siguiente threshold map de dimensión 4x4:

$$\begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix} \quad (3.4)$$

el cual corresponde a una matriz de Bayer [18], donde la distancia promedio entre dos elementos vecinos se maximiza, lo que proporciona una distribución óptima del ruido aplicado a la imagen.

Para cubrir todos los píxeles de la imagen, se pueden concatenar replicas del threshold map en las direcciones horizontales y verticales para obtener el threshold asociado a cada píxel dependiendo de su posición. En la práctica, no es necesario guardar directamente réplicas del threshold map en memoria, sino que resulta más eficiente implementar una expresión que mapee la posición absoluta del píxel en la imagen al índice del valor correspondiente en el threshold map. En este caso, considerando un threshold map de 4x4, basta con obtener el módulo 4 del índice vertical y horizontal del píxel en la imagen, los cuales se enumeran de la forma presentada en la Figura 3.16. Esto es equivalente a tomar los dos bits menos significativos de cada índice.

Para facilitar la obtención del threshold en base a la posición del píxel en el código, se hizo uso de la siguiente expresión para definir la matriz de thresholding

```
parameter [3:0] array [0:3][0:3] =  
'{'{0,8,2,10}, '{12,4,14,6}, '{3,11,1,9}, '{15,7,13,5}};
```

Esta expresión especifica un arreglo de 4x4 elementos de 4 bits cada uno. Se debe notar que los índices siguen una representan big endian, lo que permite acceder a la posición correspondiente del elemento en la matriz ingresando los índices de posición horizontal (x) y vertical ((y)), relativos a la matriz de thresholding, de la siguiente manera

```
assign threshold = array[x][y];
```

lo cual resulta más intuitivo al ser similar a cómo se realiza el manejo de arreglos en lenguajes de programación tradicionales. En el caso de que los elementos fueran ordenados utilizando notación little endian, el elemento (0,0) correspondería al (3,3), lo que resultaría confuso.

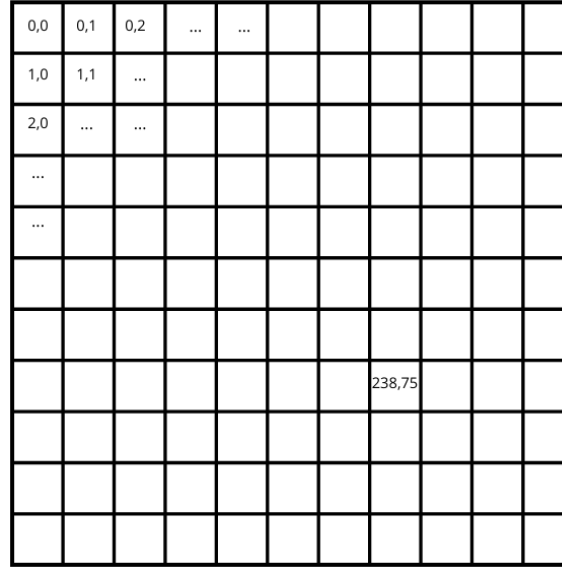


Figura 3.16: Enumeración de píxeles en la pantalla

Tabla 3.1: Caption

|            | Decimal | Binario           | Indice threshold map | Threshold condition (th=9) | Color en 4 bits |
|------------|---------|-------------------|----------------------|----------------------------|-----------------|
| Posición x | 238     | 1110 11 <b>10</b> | $10_2 = 2_{10}$      | -                          | -               |
| Posición y | 75      | 0100 10 <b>11</b> | $11_2 = 3_{10}$      | -                          | -               |
| Canal R    | 120     | <b>0111</b> 1000  | -                    | $1000_2(8) < th$           | 0111            |
| Canal G    | 30      | <b>0001</b> 1110  | -                    | $1110_2(14) > th$          | 0010            |
| Canal B    | 185     | <b>1011</b> 1001  | -                    | $1001_2(9) = th$           | 1011            |

Para ilustrar el procesamiento de un píxel, consideremos la información del píxel presentada en la Tabla 3.1. Como se mencionó previamente, el valor del módulo 4 de las posiciones horizontales y verticales corresponde a tomar los dos bits menos significativos de cada índice, los cuales se marcan en rojo en la tabla. En el caso del ejemplo, estos bits indexan al elemento de la tercera fila y cuarta columna del mapa de threshold, correspondiente al valor 9. Con esta información, se procesan los bits de cada uno de los canales de color. Para cada canal, se define como magnitud auxiliar de salida a partir de los cuatro bits más significativos (marcados en negritas en la columna), mientras que los 4 bits menos significativos se comparan con el valor del threshold derivado previamente a partir de la posición del píxel. En el caso del canal R, la magnitud de los píxeles menos significativos es menor al threshold, por lo que se mantienen los 4 bits más significativos del canal de color para la salida. En el caso del canal G, la magnitud de los 4 bits menos significativos es mayor al threshold, por lo que el valor auxiliar se incrementa en 1 al pasar a la salida. En el caso del canal B, la magnitud de los bits de ajuste es igual al threshold, por lo que el valor se mantiene.

Luego de describir estos cálculos en el código e implementar el filtro en la FPGA, el nuevo diagrama general del sistema es el de la Figura 3.17 y los resultados esperados del filtro pueden visualizarse en la Figura 3.18, donde se puede ver que las áreas afectadas por el truncado de los canales de color originales se suaviza gracias al efecto de la matriz de thresholding. Además de lo anterior, en la Figura 3.19, se puede ver cómo aplicar el filtro dithering en conjunto al de

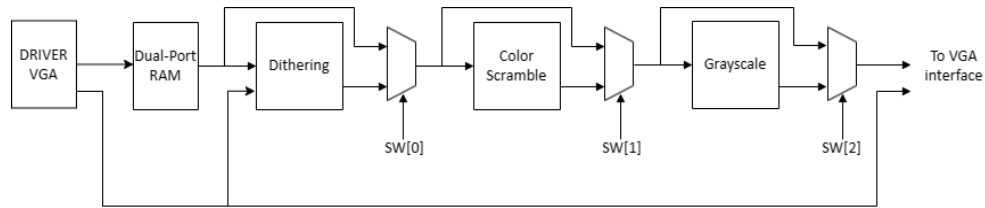
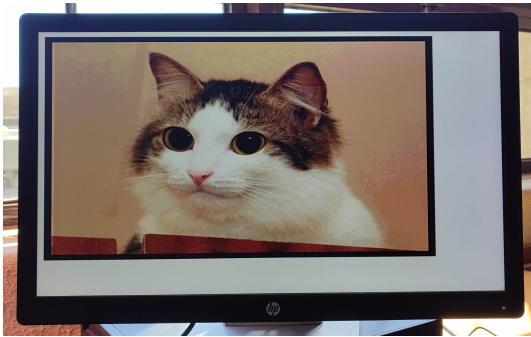
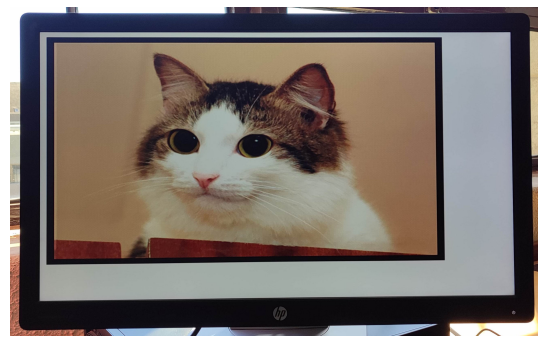


Figura 3.17: Diagrama del sistema con la inclusión del dithering



(a) Imagen original



(b) Imagen con el filtro dithering

Figura 3.18: Comparación de la imagen normal y el filtro dithering

grayscale puede mejorar la percepción visual de la imagen.

### Sobel

El algoritmo de Sobel detecta bordes de una imagen en escala de grises calculando las derivadas de los valores de los píxeles en los ejes horizontales y verticales de la imagen, cuanto más alto sea el cambio del color de un píxel a otro, más intensidad se le dará al píxel de salida, dándose un máximo de intensidad cuando un píxel pasa de negro a blanco o viceversa. El calculo anterior se realiza píxel a píxel, requiriendo conocer los valores de los píxeles vecinos para poder realizarlo, lo que precisa el uso de un buffer que permita contar con el valor de los píxeles vecinos al píxel que está siendo procesado en un ciclo de reloj dado.



(a) Imagen con grayscale



(b) Imagen con el filtro dithering y grayscale

Figura 3.19: Comparación de la imagen con grayscale y grayscale con dithering

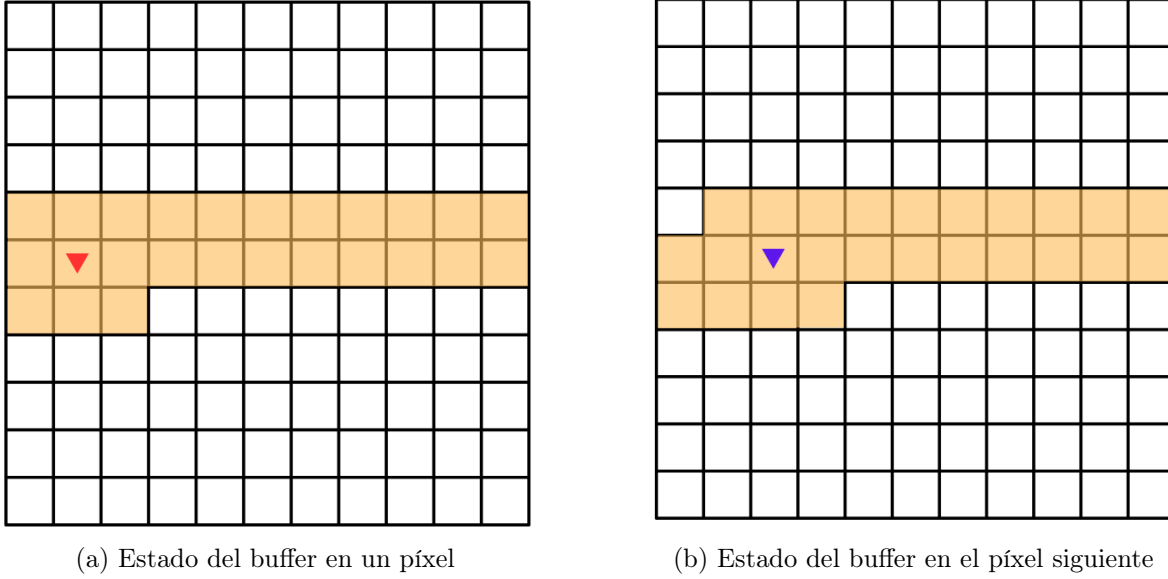


Figura 3.20: Representación del buffer

En la Figura 3.20a puede verse una representación del problema mencionado anteriormente. Si se desea calcular el valor del píxel marcado con el triángulo rojo, se necesitan conocer los valores de todos los píxeles que lo rodean. Para lograr esto, se utiliza un *buffer* que guarda todos los píxeles que están en amarillo. El número total de píxeles que se necesitan guardar se puede generalizar como:

$$\text{BUFFER\_LENGTH} = 2 \cdot \text{IMAGE\_WIDTH} + 3 \quad (3.5)$$

En la Figura 3.20b se puede observar que para calcular el valor de un nuevo píxel consecutivo al anterior (marcado con el triángulo azul), basta con ingresar al buffer un nuevo píxel y descartar el primero que había entrado al buffer y que permanecían en este. Este comportamiento se puede implementar por medio de un shift register con una estructura Serial-Input Paralel-Output (SIPO), del cual la salida son todos los píxeles correspondientes al cuadrado que encapsula al píxel objetivo.

El requerimiento de conocer el valor de los vecinos inmediatos en dirección horizontal y vertical para cada píxel impone dos requerimientos prácticos que requiere tomar decisiones de diseño:

- Dado que los píxeles se acceden ordenadamente en forma secuencial desde la memoria, la información necesaria para calcular el valor de un píxel leído de memoria solo estará disponible una vez hayan transcurrido el número de ciclos del pixel clock necesarios para recorrer toda una línea horizontal. Es decir, si se usa el mismo contador para desplazarse por la pantalla y para acceder los datos de memoria, el valor de un píxel dado se estaría desplegando con un desplazamiento de un píxel en la dirección vertical y uno en la posición horizontal con respecto a la imagen original.
- Los píxeles en los bordes de la imagen original no tienen vecinos en alguna de sus direcciones, por lo que no es posible realizar el cálculo en forma directa.

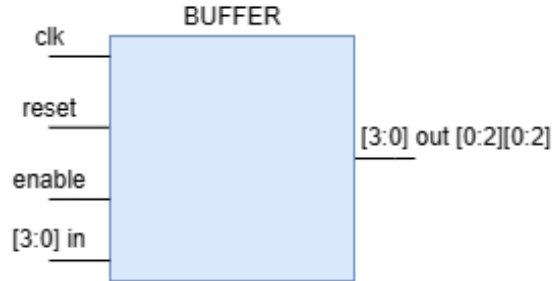


Figura 3.21: Entradas y salidas del buffer

Por simplicidad, para resolver los puntos anteriores se opta por permitir que la imagen generada por el filtro Sobel se despliegue desplazada en un píxel horizontal y un píxel vertical con respecto a la imagen original, y también se omite el procesamiento de los bordes de la imagen original, por lo que la imagen procesada tiene un píxel menos en ambas dimensiones. Si bien sería posible implementar lógica adicional para mantener la posición y dimensión de la imagen original, podría complicar innecesariamente la experiencia, sobretodo considerando que el efecto visual de estas simplificaciones es casi imperceptible al ojo humano.

La Figura 3.21 muestra un diagrama de alto nivel de entrada/salida para el buffer que se debe implementar. La entrada `in` corresponde al valor del píxel en 4 bits (escala de grises), el cual se guarda en una cola en el orden que se reciben. La salida del buffer corresponde a un arreglo de  $3 \times 3$ , donde el elemento (1,1) corresponde el píxel de interés, y los demás elementos corresponden a los píxeles vecinos en la imagen. La señal `enable` se utiliza para habilitar o deshabilitar el ingreso de nuevos píxeles a la cola.

En la descripción de *hardware*, un buffer de `BUFFER_LENGTH` elementos de `INFO_WIDTH` bits cada uno queda con la siguiente estructura:

```
logic [INFO_WIDTH - 1 : 0] buffer [BUFFER_LENGTH - 1 : 0];

always_ff @(posedge clock) begin
    if(reset) begin
        for (int i = 0; i < BUFFER_LENGTH; i = i + 1) begin
            buffer[i] <= '0;
        end
    end else begin
        if (enable) begin
            buffer <= {buffer[BUFFER_LENGTH-2:0], in};
        end
    end
end
```

La salida del buffer se puede expresar mediante la siguiente estructura:

```
output logic [INFO_WIDTH - 1 : 0] out [0:2][0:2]

assign out[0][0] = buffer [2 + 2 * IMAGE_WIDTH];
assign out[0][1] = buffer [1 + 2 * IMAGE_WIDTH];
assign out[0][2] = buffer [0 + 2 * IMAGE_WIDTH];
```

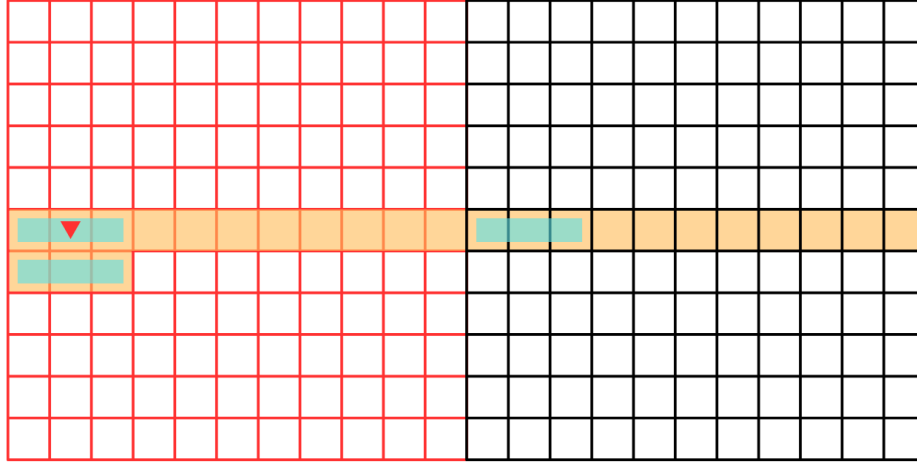


Figura 3.22: Representación del problema con el buffer

```

assign out[1][0] = buffer [2 + IMAGE_WIDTH];
assign out[1][1] = buffer [1 + IMAGE_WIDTH];
assign out[1][2] = buffer [0 + IMAGE_WIDTH];
assign out[2][0] = buffer [2];
assign out[2][1] = buffer [1];
assign out[2][2] = buffer [0];

```

donde `IMAGE_WIDTH` corresponde al ancho de la imagen en numero de píxeles. Se hace notar que el elemento `buffer[0]` corresponde al píxel guardado más recientemente en el buffer.

Otra consideración que debe tenerse con este buffer es que solo debe estar habilitado para guardar píxeles de la pantalla que efectivamente correspondan a la imagen a mostrar, la cual puede tener una resolución menor a la pantalla. En el caso de esta actividad, la pantalla está configurada para una resolución de 640x480 píxeles, mientras que la imagen a mostrar es solo de 516x387 píxeles. Esta situación puede generar el problema mostrado en la grilla de la Figura 3.22, donde la zona roja corresponde a píxeles de la imagen, mientras que la zona negra corresponde a píxeles de la pantalla cuyos índices están fuera del rango en que se despliega la imagen. En la Figura, los píxeles que almacena el buffer están pintados en amarillo, y como puede observarse estos se almacenan cuando nos encontramos dentro y fuera de la imagen. Al aplicar el procesamiento con estos datos se tendría un error de visualización. Para evitar este problema, se usa la señal `enable` para guardar solo los píxeles del área que contiene la imagen.

Una vez se tienen todos los píxeles vecinos se debe proceder a calcular las aproximaciones de las derivadas en los ejes horizontales ( $G_x$ ) y verticales ( $G_y$ ), las cuales están definidas como una convolución de un kernel aplicado a toda la imagen, donde la formula genérica de convolución es

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n - m] \quad (3.6)$$

y las operaciones para calcular ( $G_x$ ) y ( $G_y$ ) son

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I \quad (3.7)$$

donde I representa la matriz de la imagen completa. En este caso estamos interesados en como realizar los cálculos sobre una posición en específico de la imagen, para lo cual realizaremos una suma de las multiplicaciones elemento a elemento entre el kernel correspondiente y la región de 3x3 píxeles que se guardo en el buffer, cuyo elemento central corresponde al píxel de interés.

Por ejemplo, si tuviéramos la siguiente matriz de píxeles

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 5 & 7 & 10 \\ 6 & 10 & 15 \end{bmatrix} \quad (3.8)$$

donde el elemento central con valor 7 seria el píxel sobre el cual se desea calcular el operador Sobel y todo el resto serian sus vecinos, los cálculos de  $G_x$  y  $G_y$  se verían de la siguiente manera

$$G_x = -1(2) + 0(5) + 1(1) - 2(5) + 0(7) + 2(10) - 1(6) + 0(10) + 1(15) = 18 \quad (3.9)$$

$$G_y = -1(2) - 2(5) - 1(1) + 0(5) + 0(7) + 0(10) + 1(6) + 2(10) + 1(15) = 28 \quad (3.10)$$

Para implementar los cálculos anteriores en SystemVerilog, se declaran  $G_x$  y  $G_y$  como *logic signed* para dejar claro que deben ser tratados como valores con signo, además de esto el tamaño de estas señales se coloca en 7 bits, ya que un valor con signo de 7 bits en complemento a 2 tiene un rango desde -64 hasta +63, mientras que el rango de valores que pueden tomar  $G_x$  y  $G_y$  va desde -60 hasta +60, lo cual puede ser calculado considerando los siguientes casos en los cuales las sumatorias representan la suma de todos los elementos de la matriz.

$$A = \begin{bmatrix} 0 & 0 & 15 \\ 0 & 0 & 15 \\ 0 & 0 & 15 \end{bmatrix} \quad G_{x\_max} = \sum \begin{bmatrix} -1(0) & 0 & 1(15) \\ -2(0) & 0 & 2(15) \\ -1(0) & 0 & 1(15) \end{bmatrix} = 60 \quad (3.11)$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 15 & 15 & 15 \end{bmatrix} \quad G_{y\_max} = \sum \begin{bmatrix} -1(0) & -2(0) & -1(0) \\ 0 & 0 & 0 \\ 1(15) & 2(15) & 1(15) \end{bmatrix} = 60 \quad (3.12)$$

$$A = \begin{bmatrix} 15 & 0 & 0 \\ 15 & 0 & 0 \\ 15 & 0 & 0 \end{bmatrix} \quad G_{x\_min} = \sum \begin{bmatrix} -1(15) & 0 & 1(0) \\ -2(15) & 0 & 2(0) \\ -1(15) & 0 & 1(0) \end{bmatrix} = -60 \quad (3.13)$$

$$A = \begin{bmatrix} 15 & 15 & 15 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad G_{y\_min} = \sum \begin{bmatrix} -1(15) & -2(15) & -1(15) \\ 0 & 0 & 0 \\ 1(0) & 2(0) & 1(0) \end{bmatrix} = -60 \quad (3.14)$$

Teniendo los valores de  $G_x$  y  $G_y$  se calcula la magnitud del cambio en un píxel ( $G$ ) como

$$G = \sqrt{G_x^2 + G_y^2} \quad (3.15)$$

Este valor ( $G$ ) corresponde al último cálculo del operador Sobel y últimamente el valor que se asignara a los canales de color del píxel a la salida. Considerando que el valor máximo de  $G_x$  y  $G_y$  es 60, se obtiene que el valor máximo de  $G$  es aproximadamente 85, lo cual es mayor a los valores que podemos representar en la escala de grises.

Considerando lo anterior, debemos ajustar la formula de  $G$  para que el valor máximo sea 15 y quede dentro del rango de los valores de 4 bits que se utilizan para los canales de color. Para realizar esto, se divide la suma interna de la raíz por 32, lo cual al momento de implementar se puede aproximar con un right shift de 5 posiciones. Con este cambio el nuevo valor máximo de  $G$  es 15.

Para realizar el cálculo de la raíz cuadrada en el código, se propone, en primera instancia, utilizar el IP CORDIC disponible en el IP Catalog de Xilinx, el cual permite realizar varias operaciones trigonométricas como rotaciones, traslaciones, seno, coseno, además del cálculo de raíces cuadradas. La configuración de esta IP en Vivado 2023.1 se puede observar en la Figura 3.23.

$$(G_{x_{max}} + G_{y_{max}})/32 = (60^2 + 60^2)/32 = 225 = 11100001_2 \quad (3.16)$$

La salida del IP CORDIC es de 5 bits, de los cuales solo se consideran los 4 bits menos significativos para asignarlos a los valores de los canales R, G y B para obtener la imagen con el filtro Sobel aplicado.

En la Figura 3.24 se pueden ver dos comparaciones de imágenes antes y después de prender el filtro Sobel.

Como referencia, un diagrama de alto nivel del sistema con todos los filtros ya implementados puede ser visto en la Figura 3.25. Se realiza todo el proceso de síntesis, implementación y generación de bitstream del sistema completo, el cual demora aproximadamente 4 minutos en un computador de escritorio de gama media <sup>3</sup>, reforzando la importancia de una correcta planificación al diseñar, ya que se puede perder bastante tiempo al realizar intentos de corregir errores usando como método de verificación la correcta visualización en *hardware*, especialmente en computadores de menores especificaciones. El reporte de *timing* final puede observarse en la Figura 3.26, pudiendo verse que se dispone de un Worst Negative Slack positivo lo cual implica que los requerimientos de *timing* si se cumplen.

Con la implementación de todos estos filtros y la visualización de imágenes en pantalla, se espera que los alumnos ya estén acostumbrados a trabajar con las señales involucradas a las

---

<sup>3</sup>Procesador Intel i5-11600k, tarjeta gráfica RTX3070, 16 GB RAM

Component Name

**Configuration Options** AXI4 Stream Options

**Configuration Parameters**

Functional Selection

Architectural Configuration

Pipelining Mode

Data Format

Phase Format

**Input/Output Options**

Input Width  [8 - 48]

Output Width  [5 - 48]

Round Mode

**Advanced Configuration Parameters**

Iterations  [0 - 48]

Precision  [0 - 48]

☐ Coarse Rotation

Compensation Scaling

Figura 3.23: Configuración del CORDIC

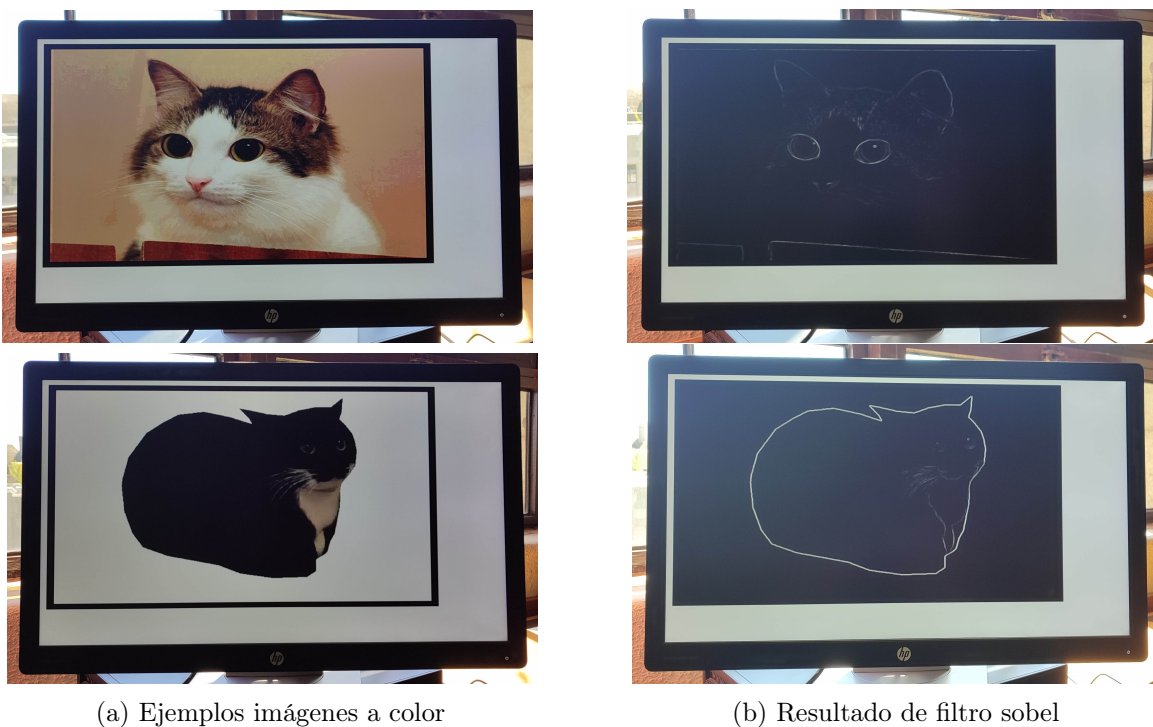


Figura 3.24: Comparación de imágenes antes y después de activar el filtro Sobel

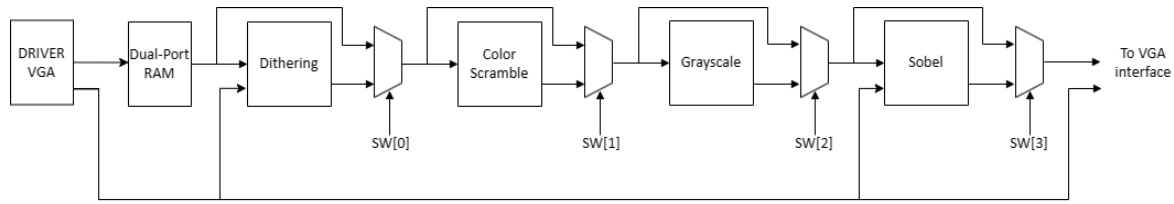


Figura 3.25: Diagrama del sistema con la inclusión del Sobel

| Setup                                | Hold                             | Pulse Width                                       |
|--------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): 0.140 ns | Worst Hold Slack (WHS): 0.109 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): 0.000 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0       | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 6580      | Total Number of Endpoints: 6580  | Total Number of Endpoints: 1117                   |

Figura 3.26: Reporte de *timing* en 640x480

interfaces de vídeo y filtros de imágenes, abriendo al posibilidad de enfrentarlos a escenarios más complejos, donde los filtros anteriores deberán ser optimizados para dar lugar a requerimientos de *timing* más severos.

### 3.3.5. Aumento de resolución de video e introducción de problemas de *timing*

El objetivo principal de esta guía es que los alumnos aprendan a identificar y resolver problemas de requerimientos de *timing*. Para que estos problemas aparezcan, se les indicara que aumenten la resolución de imagen con la que están trabajando, provocando que el *pixel clock* sea más rápido y que por lo tanto los ciclos sean más cortos, esto en conjunto a que los alumnos fueron guiados a una cierta solución en los filtros *grayscale* y *Sobel*, se tiene una alta probabilidad de que aparezcan de este tipo de problemas .

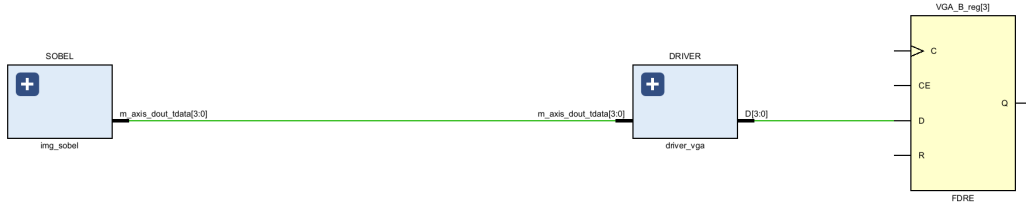
De la guía anterior los alumnos debiesen tener una resolución de imagen de 640x480 @75Hz con un *pixel clock* de 30.75MHz y un sistema sin problemas de requerimientos de *timing* como fue mostrado en la Figura 3.26. En la primera actividad se les pedirá que aumenten la resolución a 800x600 @75Hz, lo cual requerirá un aumento del *pixel clock* a 49MHz y cambios en los parámetros del driver, provocando que para el sistema de filtros diseñado en la actividad anterior no se cumplan los requerimientos de *timing* como puede ser observado en la Figura 3.27.

En la Figura 3.28, la cual muestra el esquemático del *path* crítico, el cual es el path que genera la mayor latencia, se observa que están presentes el modulo *Sobel* y el *driver VGA*, debido que el driver no puede ser optimizado, se analiza el modulo Sobel y se observa que el path crítico

| Setup                                  | Hold                             | Pulse Width                                       |
|----------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): -4.269 ns  | Worst Hold Slack (WHS): 0.024 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): -58.044 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 16        | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 8295        | Total Number of Endpoints: 8295  | Total Number of Endpoints: 2008                   |

Timing constraints are not met.

Figura 3.27: Reporte de *timing* en 800x600 que no cumple requerimientos de *timing*

Figura 3.28: Visualización del peor *path* en el esquemático a 800x600

| Setup                                 | Hold                             | Pulse Width                                       |
|---------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): -0.780 ns | Worst Hold Slack (WHS): 0.024 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): -8.993 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 16       | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 7439       | Total Number of Endpoints: 7439  | Total Number of Endpoints: 2022                   |
| Timing constraints are not met.       |                                  |                                                   |

Figura 3.29: Reporte de *timing* en 800x600 luego de aproximar Sobel

de este modulo está asociado al cálculo de la raíz cuadrada, por lo que se sugiere a los alumnos que replacen el módulo CORDIC, por una aproximación que consuma menos recursos, esta solución aproximada fue obtenida de la pagina web [19] y consiste en dos pasos, de los cuales el primero es calcular el valor absoluto de los factores  $G_x$  y  $G_y$  de las ecuaciones 3.7 y realizar una suma simple entre estos

$$G = |G_x| + |G_y| \quad (3.17)$$

Luego, al valor de  $G$  el cual tiene un ancho de 7 bits, se le realiza una manipulación de bits para calcular el valor final de salida, la cual puede ser visualizada en el código a continuación

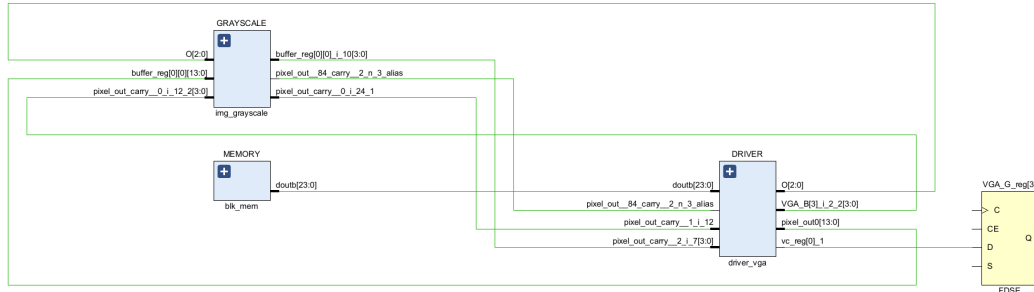
```
if (g[6])
    pixel_out = 4'b1111;
else
    pixel_out = g[5:2];
```

donde se puede observar que si el bit más significativo de  $G$  es 1, la salida se fija en el máximo valor posible, y en cualquier otro caso la salida se asigna como los bits que van desde la posición 5 hasta la posición 2 de  $G$ .

Luego de aplicar la aproximación de raíz cuadrada, se observa que el análisis de *timing* estático vuelve a fallar, como puede verse en la Figura 3.29. Sin embargo, en este caso el path crítico ya no corresponde al filtro de Sobel, sino que al filtro de escala de grises. En la Figura 3.30 se puede observar el esquemático del nuevo peor path, en el cual se aprecia que el único módulo que podría ser optimizado es el módulo grayscale, por lo que se deberá encontrar una nueva aproximación del cálculo del grayscale.

Buscando disminuir el tamaño de la lógica del modulo grayscale, se prueba utilizando la siguiente fórmula reducida

```
assign pixel_out = (54 * R + 183 * G + 18 * B)/256;
```

Figura 3.30: Visualización del peor *path* en el esquemático a 800x600 luego de aproximar Sobel

| Setup                                          | Hold                             | Pulse Width                                       |
|------------------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): 0.566 ns           | Worst Hold Slack (WHS): 0.080 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): 0.000 ns           | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0                 | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 6533                | Total Number of Endpoints: 6533  | Total Number of Endpoints: 1117                   |
| All user specified timing constraints are met. |                                  |                                                   |

Figura 3.31: Reporte de *timing* exitoso en 800x600

la cual prueba ser suficiente para cumplir los requerimientos de *timing* como se puede ver en la Figura 3.31, al revisar el consumo de recursos de esta versión en comparación a la anterior en la tabla 3.2 se puede ver una diferencia considerable en la utilización de recursos al igual que en el número de niveles lógicos.

Habiéndose logrado cumplir los requerimientos de *timing* a 800x600 @75Hz, se propone una segunda actividad donde se les indica que incrementen la resolución a 1024x768 @75Hz aumentando el *pixel clock* a 82MHz, lo cual es un 67 % más que la resolución anterior, el reporte de *timing* 3.32, como puede ser observado muestra que no se cumplen los requisitos de *timing*.

Al revisar el esquemático del peor path en la Figura 3.33, se aprecia que el módulo grayscale es nuevamente el único que puede ser modificado de todos los que se encuentran presentes, por lo que una solución sería aproximar la fórmula para calcular el valor de los píxeles, con lo cual se indicará a los alumnos que utilicen la siguiente fórmula

```
assign pixel_out = (1 * R + 2 * G + 1 * B) >> 2;
```

al realizar este cambio y observar el reporte de *timing* 3.34, se puede ver que nuevamente el problema se ha solucionado. La utilización de LUT de esta nueva versión y el número de niveles lógico en comparación a las versiones anteriores se puede ver en la Tabla 3.3 donde la disminución respecto a las versiones anteriores es sustancial, prácticamente reduciéndose a la mitad el número de niveles lógicos.

| Versión grayscale | Uso de LUT | Niveles lógicos |
|-------------------|------------|-----------------|
| Versión 1         | 121        | 23              |
| Versión 2         | 37         | 9               |

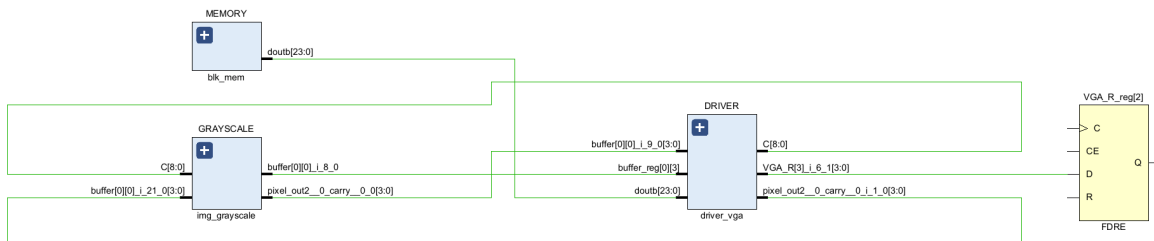
Tabla 3.2: Utilización de LUT y numero de niveles lógicos de versiones 1 y 2 de módulo grayscale

| Setup                                  | Hold                             | Pulse Width                                       |
|----------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): -1.299 ns  | Worst Hold Slack (WHS): 0.053 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): -16.872 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 16        | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 8307        | Total Number of Endpoints: 8307  | Total Number of Endpoints: 2005                   |
| Timing constraints are not met.        |                                  |                                                   |

Figura 3.32: Reporte de *timing* en 1024x768 que no cumple requerimientos de *timing*

| Versión grayscale | Uso de LUT | Niveles lógicos |
|-------------------|------------|-----------------|
| Versión 1         | 121        | 23              |
| Versión 2         | 37         | 9               |
| Versión 3         | 6          | 5               |

Tabla 3.3: Utilización de LUT y numero de niveles lógicos de versiones 1, 2 y 3 de modulo grayscale

Figura 3.33: Visualización del peor *path* en el esquemático a 1024x768

| Setup                                          | Hold                             | Pulse Width                                       |
|------------------------------------------------|----------------------------------|---------------------------------------------------|
| Worst Negative Slack (WNS): 0.152 ns           | Worst Hold Slack (WHS): 0.047 ns | Worst Pulse Width Slack (WPWS): 3.000 ns          |
| Total Negative Slack (TNS): 0.000 ns           | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0                 | Number of Failing Endpoints: 0   | Number of Failing Endpoints: 0                    |
| Total Number of Endpoints: 8284                | Total Number of Endpoints: 8284  | Total Number of Endpoints: 1996                   |
| All user specified timing constraints are met. |                                  |                                                   |

Figura 3.34: Reporte de *timing* final en 1024x768

Habiendo terminado el trabajo de optimización de los filtros, la última actividad consiste en instanciar un módulo UART, el cual se les entrega a los alumnos, este está pensado para recibir imágenes desde un computador y cargarlas durante el tiempo de ejecución, sin embargo para poder cargar las imágenes se requiere de un módulo entre el receptor UART y la BRAM que tome los datos del computador, los empaquete en píxeles individuales, y los cargue en las posiciones de memoria correspondientes a la BRAM.

El *script* de python que se encarga de enviar la información de la imagen a la FPGA, lo hace en paquetes de 1 byte de tamaño, donde cada uno de estos paquetes corresponde a un canal de color de 1 píxel. Primero se envía la información del primer píxel, empezando con la del canal rojo, luego el valor del canal verde y luego el del canal azul, posteriormente se realizará esto mismo para todos los píxeles de la imagen. El módulo UART que se proporciona captará toda esta información y a su salida indicará si la lectura de un paquete se completó, en conjunto a la información de este.

Los alumnos deberán crear una máquina de estados que pueda transitar entre captar la información del canal rojo, verde y azul de un píxel para luego enviársela a la BRAM en conjunto a la dirección en la que el píxel debe ser escrito, siendo la dirección 0 correspondiente al primer píxel, el diagrama de la maquina de estados, el cual fue generado utilizando TerosHDL [3], se puede ver en la Figura 3.35, donde se aprecia que las transiciones entre estados dependen de una señal `in_ready` la cual corresponde al aviso del módulo UART de que se terminó de leer un paquete, que existe un estado por cada canal de color donde se guarda temporalmente la información de cada uno y que al final se tiene un estado que guarda el píxel completo en memoria. Si se estipula también que cuando la dirección de escritura llegue al tamaño de la imagen, la dirección de escritura vuelva a 0, la maquina de estados queda automáticamente preparada para recibir una segunda imagen luego de haber recibido la primera.

Una vez el estudiante haya implementado la máquina de estados, habrá terminado de diseñar el sistema completo, el cual es capaz de cargar imágenes en cualquier momento, aplicarle distintos filtros que pueden ser activados en cascada proporcionando una gran variedad de efectos y funcionar adecuadamente a una resolución de 1024x768 @75Hz.

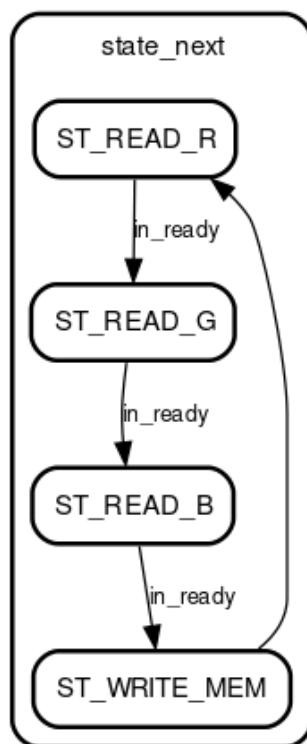


Figura 3.35: Diagrama de la máquina de estados generado por TerosHDL que guarda la imagen en la BRAM

## 4 | Material desarrollado

En este capítulo se listará todo el material desarrollado durante este informe y donde encontrarlo, con el propósito de facilitar el uso de este en experiencias futuras.

### 4.1. Listado de material

Todo el material desarrollado en esta memoria se puede encontrar en un repositorio manejado por profesor guía el cual se mantiene con acceso privado. En el caso de necesitar acceso a este repositorio se debe consultar al profesor guía. Este repositorio además de guardar el material desarrollado durante este trabajo de memoria, también recopila material generado anteriormente a modo de reunir todo lo necesario en un lugar. Un listado de todo el material recopilado en este repositorio se dejará a continuación.

Sobre la experiencia VGA se recopiló el siguiente material, el cual se dividió en tres carpetas las cuales representan tres posibles guías a realizar

- Guía 1
  - Archivos utilizados en 8bitworkshop
  - Cápsula driver VGA 8bitworkshop
  - Cápsula driver VGA en FPGA
- Guía 2
  - Script de Python para generar archivos COE
  - Script para cargar imágenes a la FPGA a través de UART
  - Hoja de calculos para calcular los parámetros del driver VGA
  - Códigos en SystemVerilog del sistema a 640x480
- Guía 3
  - Bitstreams de los sistemas optimizados a cada una de las tres resoluciones
  - Códigos en SystemVerilog del sistema a 800x600 y 1024x768

Sobre el nuevo flujo de trabajo IcarusVerilog más Visual Studio Code

- Documento con los códigos utilizados durante la cápsula y consejos de uso

- Cápsula del tutorial de instalación y uso del nuevo flujo de trabajo

Sobre las pruebas automáticas

- Tutorial del uso de pruebas automáticas para los ayudantes
- Información sobre el repositorio que contiene los archivos actualizados, el cual es manejado por <https://github.com/AlfonsoUSM> y requiere de su permiso para ser accedido

## 5 | Conclusión y trabajo futuro

Este capítulo resume los resultados obtenidos en este trabajo de memoria, indicando lo aprendido y las limitaciones de lo realizado, listando al final recomendaciones para trabajo futuro dentro de la asignatura.

### 5.1. Conclusiones

El trabajo aquí presentado expuso el análisis y material desarrollado en la búsqueda de aumentar el enfoque práctico del laboratorio de sistemas digitales del Departamento de Electrónica de la Universidad Técnica Federico Santa María. El análisis incluyó la actualización del estudio de cursos similares a ELO212 en búsqueda de posicionarlo en comparación a otros y encontrar posibles mejoras, la revisión de los *scripts* de revisión asistida y el estudio de herramientas alternativas a Vivado con el objetivo de entregar más opciones a los alumnos que no tengan las especificaciones técnicas para ejecutarlo. En el material desarrollado se encuentra un tutorial del uso de las pruebas de revisión asistida, un nuevo flujo de trabajo opcional utilizando herramientas alternativas a Vivado y un estudio comprensivo sobre la creación de una experiencia de la utilización del protocolo VGA y procesamiento de imágenes.

De los cursos observados en la comparación, se llegó a la conclusión de que la edición actual del curso ELO212 posee una buena estructura y proporciona un contenido sustancial para una asignatura introductoria, pudiendo destacarse algunos posibles cambios que pudieran introducirse de otros cursos los cuales fueron considerados como deseables pero no necesarios.

Se analizó la necesidad de generar documentación para nuevos usuarios de los *scripts* de revisión asistida y la facilidad de modificar las pruebas automáticas para ajustarlas a cambios menores en las guías. De lo primero se vio que existía una falta de material para los usuarios primerizos, decidiéndose generar un tutorial del uso de estos con base en el conocimiento acumulado de los ayudantes que lo han usado previamente, este tutorial fue entregado a los nuevos ayudantes de la versión del I semestre 2023 de ELO212 demostrando ser efectivo en enseñar a utilizar la herramienta. Respecto a la facilidad de modificar estas pruebas se comprobó manualmente durante la versión del curso señalada anteriormente, que la alteración de estas resultó ser factible cuando los cambios no tienen impactos en los *scripts* Tcl, los cuales requerían modificaciones en múltiples archivos que podían resultar complejos de manejar, por lo que se concluyó que sería necesario continuar trabajando en estas pruebas en el futuro.

La necesidad de encontrar una herramienta de trabajo alternativa a Vivado que requiriese de menos recursos computacionales y cumpliera con las necesidades del curso ha sido una problemática estudiada anteriormente a la cual hasta el momento no se había encontrado

solución, pero que durante el estudio de las herramientas gratuitas disponibles en el mercado realizado en este trabajo dio un paso hacia adelante en la dirección correcta, pudiéndose dar con un flujo de trabajo basado en Icarus Verilog el cual permite simular sistemas descritos en SystemVerilog, que además tiene facilidades como la generación automática de templates de instancias y simulaciones de módulos, asimismo como la inferencia de diagramas de maquinas de estados en base al código de estas. Este nuevo flujo es un complemento a Vivado, debido a que carece de la facultad de realizar los pasos de implementación y generación de bitstreams, por lo que los alumnos que lo utilicen todavía requerirán acceso a Vivado para las etapas finales de diseño. Con lo anterior en mente, Vivado seguirá siendo la herramienta principal del curso en el futuro cercano.

El proceso de reinventar la experiencia de interfaz de vídeo VGA y procesamiento de imágenes basándose en el flujo de trabajo de diseño de experiencias de laboratorio desarrollado en el trabajo [4] resultó en el diseño de variadas actividades con nuevos objetivos de enseñanza y un flujo optimizado para los estudiantes en comparación a la guía original del año 2019, la cual dejaba los problemas planteados de manera abierta y resultaba en un desarrollo mas complicado para los alumnos. Los nuevos objetivos de enseñanza fueron encontrados al momento de desarrollar la guía original al observar errores que fueron pasados por alto anteriormente y detalles que complicaban la resolución por parte de los estudiantes de las actividades. Las nuevas guías se fueron desarrollando naturalmente en base a estos nuevos objetivos, resaltando la importancia de iterar en el trabajo previo y trabajar teniendo los objetivos de enseñanza en mente.

El material desarrollado en esta tesis y los aprendizajes obtenidos en su creación proveerán una base para extender el enfoque práctico para la docencia en diseño de sistemas digitales, otorgando nuevas herramientas a los estudiantes con las que trabajar y nuevas guías que incorporan periféricos y funcionalidades que no han visto anteriormente y que tiene resultados visualmente llamativos. Más bien, si lo anterior es un avance en la dirección correcta, no representa el fin de la incorporación de un enfoque práctico a la asignatura, ya que todavía falta trabajo en el estudio de la integración de un enfoque mas práctico a la asignatura en general, realizando cambios en la estructura de este a lo largo de todas las guías de un semestre, analizando cada guía del curso en búsqueda de formas de incorporar mejoras. Además de lo anterior, el contenido de este documento siempre puede ser refinado y los cabos que hayan quedado sueltos pueden ser atados.

## 5.2. Trabajo Futuro

En esta sección se detallan los puntos que debiesen continuarse trabajando en futuros proyectos del laboratorio de sistemas digitales

Se debe continuar el estudio de las distintas herramientas disponibles en el mercado en búsqueda de nuevas alternativas viables a Vivado, también profundizando en las herramientas con las que ya se trabaja buscando encontrar funcionalidades útiles para los alumnos y *staff*.

Verificar el uso del flujo de trabajo basado en Icarus Verilog en grupos de estudiantes, recibiendo retroalimentación sobre su uso y realizando modificaciones en el caso de ser necesario, como lo podría ser el cambio de ciertas herramientas o ajustar las configuraciones de las que se están utilizando.

Actualizar constantemente el tutorial de las pruebas asistidas para incluir cambios a las pruebas o nuevas buenas prácticas que se encuentren.

Desarrollar nuevas funcionalidades para las pruebas de revisión asistida apuntadas a facilitar el uso y la modificación de estas.

# Bibliografía

- [1] “Documentacion IcarusVerilog.” <https://steveicarus.github.io/iverilog/>. Accedido: 2023-09-15.
- [2] “Pagina principal Visual Studio Code.” <https://code.visualstudio.com/>. Accedido: 2023-09-15.
- [3] “TerosHDL.” <https://terostechnology.github.io/terosHDLdoc/>. Accedido: 2023-09-15.
- [4] P. Henríquez, *Diseño de experiencias prácticas para diseño y verificación de sistemas digitales utilizando lenguajes de descripción de hardware y herramientas de simulación*. Tesis de pregrado inédita, Universidad Técnica Federico Santa María, 2021.
- [5] “Ieee standard for systemverilog—unified hardware design, specification, and verification language,” *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pp. 1–1315, 2018.
- [6] “Electrónica Digital Bit a Bit: Fundamentos, Verilog y FPGA.” <https://www.coursera.org/learn/electronica-digital-bit-a-bit-fundamentos>. Accedido: 2022-10-7.
- [7] “Ieee standard for verilog hardware description language,” *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pp. 1–590, 2006.
- [8] “Electrónica Digital Bit a Bit: Diseñando en Verilog para FPGA.” <https://www.coursera.org/learn/electronica-digital-bit-a-bit-disenando-circuitos-complejos>. Accedido: 2022-10-7.
- [9] “Introducción al diseño de hardware con Verilog.” <https://www.edx.org/es/learn/electronics/universidad-galileo-introduccion-al-diseno-de-hardware-con-verilog>. Accedido: 2022-10-7.
- [10] “Introductory Digital Systems Laboratory.” <http://web.mit.edu/6.111/volume2/www/f2019/index.html>. Accedido: 2023-09-21.
- [11] “FPGA computing systems: A Bird’s Eye View.” <https://www.coursera.org/learn/fpga-computing-systems>. Accedido: 2022-10-7.
- [12] “FPGA computing systems: Partial Dynamic Reconfiguration.” <https://www.coursera.org/learn/copy-of-fpga-intro>. Accedido: 2022-10-7.
- [13] “Ieee standard for systemverilog—unified hardware design, specification, and verification language,” *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, pp. 1–1315, 2013.

- [14] “Pagina principal GTKWave.” <https://gtkwave.sourceforge.net/>. Accedido: 2023-09-18.
- [15] “8bitworkshop.” <http://8bitworkshop.com/>. Accedido: 2023-09-15.
- [16] “Manual de referencia de la Nexys A7.” <https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual>. Accedido: 2023-09-25.
- [17] “Pagina de escala de grises en Wikipedia.” <https://en.wikipedia.org/wiki/Grayscale>. Accedido: 2023-10-03.
- [18] “An optimum method for two-level rendition of continous-tone pictures.” <https://web.archive.org/web/20130512190753/http://white.stanford.edu/~brian/psy221/reader/Bayer.1973.pdf>. Accedido: 2023-09-17.
- [19] “2D Convolution in Hardware.” <https://sistenix.com/sobel.html>. Accedido: 2023-02-16.