Fecha de entrega 9 de mayo de 2023.

Disclaimer This latex template has been obtained from IEEE ACCESS for educational purpose.

LEDerize!

ANTONIO DÁVILA[1], ELOY LÓPEZ[1]
[1]Pontificia Universidad Católica de Chile (e-mail: a.davila@uc., ealv.sic@uc.cl)

SI (POR FAVOR ESCOGER SI o NO) Autorizo que mi proyecto (tal como ha sido entregado, sin nota ni comentarios de evaluación) sea publicado en un repositorio para pueda servir de guía y ser mejorado en proyectos de futuros estudiantes.

Este proyecto ha sido desarrollado bajo el curso IEE2463: Sistemas Electrónicos Programables.

ABSTRACT Este proyecto tiene por objetivo la implementación de un juego de seguimiento de patrones de leds por medio de botones como diseño general por medio de Vivado y con lenguaje foco VHDL. Además de esto para la realización del proyecto se ocupa tanto AXI4 Lite como Advance mediado por ATGs para la transferencia de datos al circuito del juego.
Los resultados del proyecto fueron bastante positivos pues se logró el cometido original, tanto respecto juego en sí y su funcionamiento, como la transacción de datos desde los ATG implementados a las memorias que guardan dicha información.

INDEX TERMS VHDL, maquina de estados, patrones de leds, AXI Traffic Generator, AXI4 Lite, AXI4 Full, FSM, AXI perisferical.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

EN este proyecto, se implementó un juego el cual muestra una secuencia de leds encendiendose y apagándose, y luego recibe una entrada del usuario, el cual debe imitar correctamente el orden de la secuencia anteriormente mostrada, o de lo contrario pierde el juego. La implementación permite escoger entre 4 secuencias diferentes y 4 velocidades de juego.

Para lograr la funcionalidad del proyecto, se programaron numerosos bloques en Vivado, cuya interacción queda resumida en el siguiente diagrama:

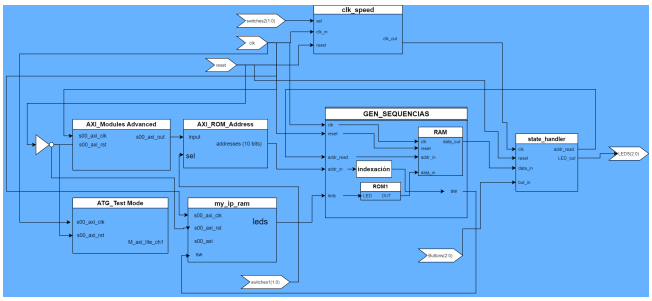

FIGURE 1: Implementación en diagrama de bloques.

Se tienen dos configuraciones de comunicación AXI. Primero, una serie de bloques implementados, los cuales fueron resumidos como el bloque "AXI Modules Advanced" con el objetivo de simplificar la visualización. Consiste de un bloque de ATG en test mode que envía información a una ATG en Advance Mode, donde ambos estan interconectados por medio de un módulo "smart connect". La información recibida por el ATG en Advanced mode es pasada a un ip core esclavo que actúa como buffer de comunicación AXI, y luego esta información para a ser escrita en un IP CORE esclavo creado por nosotros llamada ROM Address, la cual contendrá 4 vectores de 10 bits que definirán la secuencia que se mostrará en el juego. Los contenidos de esta configuración serán abordados más en detalle en el video adjunto al proyecto.

La segunda configuración consiste en un ATG en Test Mode que envía una transacción que guardará en una RAM esclavo los números "0001", "0010" y "0100", que representan las luces de la tarjeta y que luego serán almacenadas en el orden indicado por el módulo ROM Address. Esta RAM enviará la información al generador de secuencias, donde una ROM interna eliminará el bit más significativo, el cual en este caso son los ceros, para así llevarlos al tamaño correspondiente.

Un generador de secuencias lee la información contenida en distintas memorias, lo que almacena una secuencia determinada por la entrada "sel" del bloque ROM Adress a una RAM interna. Dicha entrada consiste en dos switches de la tarjeta Zybo Z7, los cuales fueron configurados como W13 y T16. Los memorias leídos son dos: Una RAM contiene bits que representan las luces (0001, 0010 y 0100), y que luego serán cambiadas de tamaño, como se indicó anteriormente. La segunda memoria es la ROM esclavo de los módulos comunicados por AXI Advance, y contiene distintos vectores

de bits que representan la dirección de la cual haya que extraer información de la primera memoria. Dichos vectores son de 10 bits de largo, donde cada 2 bits representan una dirección. Las direcciones contenidas en estos vectores están ordenados de tal forma que se guarde una combinación siguiendo tal orden en una memoria RAM contenida en el generador de secuencias, la cual será leida por el siguiente bloque principal.

El bloque que lee la información del generador de secuencias es un "state handler". Su propósito es leer la RAM interna del generador de secuencias y mostrar la secuencia de luces en la FPGA. También tiene como objetivo recibir la entrada del usuario y comparar el orden de tal entrada con el orden de la secuencia almacenada en la RAM leída. Si el orden es el correcto, el juego sigue, y en caso contrario el juego es reiniciado. Este bloque también controla los LEDS tanto para mostrar la secuencia como para mostrar el estado actual del juego.

Otro bloque define la velocidad del reloj bajo el cual va a funcionar el aspecto de control de luces LED del state handler, mediante un selector llamado "clk sel". El propósito de este bloque es establecer una dificultad basada en que tan rapido se muestra la secuencia de luces antes de recibir las entradas del usuario. Posee cuatro velocidades que son controladas por dos switches de la tarjeta, las cuales si son cambiadas generarán un reset en todo el circuito, gracias al bloque great reset.

Cualquier cambio en alguno de switches de secuencias o de velocidad generará un "reset" en el generador y en state handler. Este reset lo permite un bloque llamado "great reset", el cual lleva a que los resets de dichos bloques sean iguales a "1" si detecta flancos (de subida o de bajada) en los switches utilizados. Para que esto funcione, es necesario conectar todos los switches tanto al generador de secuencias como a state handler, pero esto no se muestra en el diagrama para efectos de visualización.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): El código cumple con lo propuesto por esta actividad, es decir, se implementó exitosamente el instanciamiento de 3 componentes en distintas entities. En particular, se instanciaron un bloque RAM, un bloque ROM y el bloque great reset descrito anteriormente. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AO2 (100%): Se incorporaron exitosamente más de 3 bloques en el block design final de nuestro proyecto. Dichos bloques son el generador de secuencias, una ROM, el state handler, y el regulador de clock. Cada uno de estos bloques posee parámetros genéricos que ayudaron en la definición de las señales correspondientes. Se incorporaron además numerosos bloques relacionados a la comunicación AXI con IP Cores esclavos. El resultado final fue el siguiente: INSERTAR BLOQUES DEL BLOCK DESIGN.

Nivel de Logro: 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AO3 (100%): Se Utiliza un ATG en Test Mode como maestro para escribir en los registros del esclavo que es un periférico AXI Lite creado y editado por nosotros, el cual entrega sus datos recibidos por un output a una RAM que es cargada para arreglar las secuencias que emitirán las led. También se ocupa un sistema en cascada que parte con ATG en Test Mode de bits 32 como maestro para escribir en la memoria del ATG en Advance Mode instrucciones de escritura para que al ese ATG inicializarse se ejecuten en un periférico AXI Full creado y editado por nosotros, el cual entrega sus 32 bits de datos recibidos por un output a una RAM que guarda las secuencias en sí disponibles, y tiene una entrada de selector entre las 4 secuencias y el output de la secuencia codificada.

AC1 (100%): Se incorporó un FSM de forma exitosa en la forma de la implementación de state handler. Este bloque es un FSM ya que cuenta con dos etapas de operación: mostrar las secunencias ubicadas en una memoria, y recibir entradas en un orden establecido. Además, existen subetapas, como por ejemplo, un caso donde se reciban las entradas en el orden correcto o otro caso cuando se reciban en orden incorrecto. Queda diferenciado además la lógica combinacional (estado de las LEDS) y la lógica secuencial (control de las LEDS en el tiempo y control de las entradas).

Nivel de Logro: 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC2 (100%): El proyecto utiliza 4 switches, 4 botones y 6 LEDS de la Zybo. Se utiliza un LED para indicar que se está esperando una entrada, se utilizan 3 botones para entregar entradas, un boton para resetear, 2 leds R y G para indicar etapas del juego, 3 leds para mostrar la secuencia a imitar, 2 switches para elegir entre distintas secuencias, y 2 switches para elegir diferentes velocidades de juego. Además, esto queda evidenciado en el video del proyecto.

Nivel de Logro: 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC3 (50%): El código del proyecto solamente cuenta con la implementación de 5 operadores que ejercen un rol activo en la operación. Estos consisten en los operadores lógicos "and", "or" y "not" y las operaciones aritméticas "+" y "-". No se lograron implementar atributos de una forma productiva.

Nivel de Logro: 50%. Parcialmente logrado. Los operadores utilizados fueron capaces de generar bitstream, pero no se implementaron atributos.

AC4 (100%): El código del proyecto utiliza en la gran mayoría de sus procesos la definición de variables que pudan ser actualizadas instantaneamente. Variables como contadores, o variables lógicas temporales que auxilian el

funcionamiento del proceso son algunos ejemplos de los usos dados a dichas variables.

Nivel de Logro: 100%. Completamente logrado, las variables implementadas han compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC5 (100%): Se utiliza codigo secuencial en la gran mayoria de procesos gobernados por un clock, como lo son memorias RAM, y los procesos de state handler y el generador de secuencias. Se utiliza codigo concurrente en bloques tales como la ROM interna del generador de secuencias. Quedaron explicitas sus diferencias de uso, y se llega hasta el bitstream correctamente.*Nivel de Logro:* 100%.

AC6 (100%): se utilizan functions y procedures. Se explicitan sus diferencias pues se utilizan functions en el momento en el que conviene obtener una sola salida, mientras que se utilizan procedures para aprovecharse de la posibilidad de obtener multiples salidas. *Nivel de Logro:* 100%. Se llega hasta la generación de bitstream satisfactoriamente.

AC7: (0%): No se investigaron funcionalidades no vistas en clase y no se implementaron de ninguna forma. Actividad no lograda.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

Para simulaciones al proceso correspondiente del bloque "sequence gen", se obtuvieron los siguientes resultados de delay post-implementación:

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,115 ns	Worst Hold Slack (WHS):	0,141 ns	Worst Pulse Width Slack (WPWS):	3,500 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:	243	Total Number of Endpoints:	243	Total Number of Endpoints:	108

All user specified timing constraints are met.

FIGURE 2: Delays de tiempo para bloque generador de secuencias.

La presencia de estos slacks de tiempo muestra la existencia de distintos delays los cuales no son posibles de observar en simulaciones de comportamiento.

No obstante, la simulación de comportamiento para este bloque si permite ilustrar la diferencia que existe entre las actualizaciones de variables y señales, como se muestra en la figura 3.

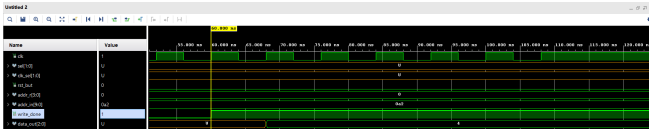

FIGURE 3: Delay entre una señal (data out) y una variable (wait done).

En esta figura, "write done" es una salida que esta gobernada por la adquisición de un cierto valor para una variable contadora definida dentro del proceso principal del generador de secuencias. Por otro lado, data out es una salida que depende del estado de una señal de lectura de una RAM interna cuando la variable contadora llega a un cierto valor. Como el contador esta definido como variable, este se actualiza primero antes de gatillar la actualización correspondiente a la señal que permite que data out salga.

Finalmente, es de interés analizar los resultados de las transacciones AXI implementadas. Para esto, se debe simular directamente los ATG en Test Mode y Advanced Mode que fueron implementados en el proyecto. Su simulación post-implementación arroja el siguiente gráfico:

A continuación presentan los gráficos que muestran las transacciones AXI a la RAM con las secuencia, se seria el sistema en cascada ATG Test (Driver) -> ATG Advance (DUT) -> Periférico: El sistema de handshakes del protocolo AXI funciona con 2 señales:

READY: establece que el canal está disponible para recibir información, estos son identificados con un cuadrado ROJO.

VALID: indica que está disponible una lectura válida para que el receptor reciba, estos son identificados con un cuadrado BLANCO.

- El primer cuadrado ROJO representa la activación del canal de lectura de direcciones (ADDR) con la señal de DUT a Driver ARREADY.
- El primer cuadrado BLANCO representa la indicación de disponibilidad de una dirección lectura válida, con la señal de Driver a DUT ARVALID.
- El segundo cuadrado ROJO representa la activación del canal de lectura de data con la señal de Driver a DUT RREADY.
- El segundo cuadrado BLANCO representa la indicación de disponibilidad de data para una lectura válida, con la señal de DUT a Driver RVALID.

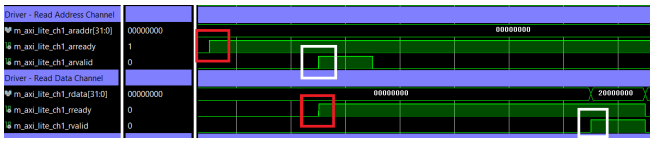

FIGURE 4: Simulación de señales para lectura entre Driver y DUT.

- El primer cuadrado ROJO superior NO indica un HANDSHAKE sino el encendido de los bloques haciendo reset '1'. Los 2 cuadrados ROJOS bajo este si lo son y representan la activación de los canales de escritura de direcciones y data con las señales de DUT a Driver AWREADY y WREADY respectivamente.
- El rectángulo AMARILLO largo contiene la secuencia de direcciones de escritura donde se guardan los datos, mientras los cuadrados BLANCOS bajo este indican la disponibilidad de las direcciones de escritura validas para recibir, con la señal de Driver a DUT AWVALID.
- El rectángulo NARANJA largo contiene la secuencia de datos de escritura, mientras los cuadrados BLANCOS bajo este indican la disponibilidad de las datos de escritura válidos para recibir, con la señal de Driver a DUT WVALID.
- La seguida de cuadrados ROJOS en la parte inferior corresponden a las activaciones del canal de confirmación de

recepción de datos, con la señal de Driver a DUT BREADY. Mientras que los cuadrados BLANCOS pequeños a sus lados representan la disponibilidad de una respuesta de confirmación de recepción válida para recibir, DUT a Driver BVALID.

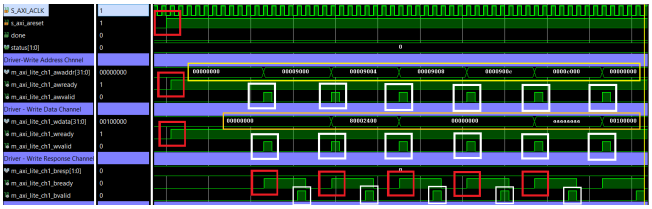

FIGURE 5: Simulación de señales para escritura entre Driver y DUT.

- El primer cuadrado BLANCO representa la indicación de disponibilidad de una dirección de escritura válida, con la señal de DUT a Periférico AWVALID. Donde el canal de Periférico a DUT AWREADY ya se encontraba activado anteriormente.

- El segundo cuadrado BLANCO representa la indicación de disponibilidad de un dato válido, con la señal de DUT a Periférico WVALID. Donde el canal de Periférico a DUT WREADY ya se encontraba activado anteriormente.

- El tercer cuadrado BLANCO pequeño en la parte inferior representa la indicación de disponibilidad de una respuesta confirmación de recepción válida, con la señal de Periférico a DUT BVALID. El cuadrado ROJO en la parte inferior corresponde a la reactivación del canal de confirmación de recepción de datos que se desactiva al recepcionar como en la parte 2), con la señal de DUT a Periférico BREADY.

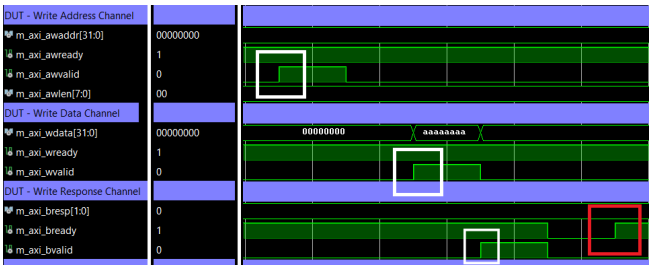

FIGURE 6: Simulación de señales para escritura entre DUT y Periférico.

Finalmente se presenta una Post-Implementation Timing Simulation de la implementación en sí con al salida de 32 bit volviendose 'AAAAAAAA' como fue comentado luego de toda la operación explicada anteriormente:

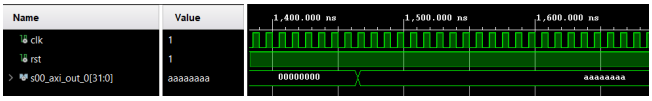

FIGURE 7: Post-Implementation Timing Simulation de salida de 32 bits.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

La implementación del proyecto planteado originalmente se logro casi en su absoluta completitud, logrando así implementar correctamente el juego descrito en el informe. Los principales desafíos que debimos solventar durante la realización del proyecto fueron los descritos a continuación.

En primera instancia tuvimos complicaciones en la elaboración de la FSM, pues en su estado de espera de input de botones no definimos correctamente por flancos de subida del botón en el módulo de debouncer hecho, como no pudimos hacerlo funcionar bien tuvimos que optar por quitarlos y los botones comenzaron a funcionar correctamente. El segundo gran problema que enfrentamos fue en editar el periférico en AXI Full para tener de output los 32 bits recibidos, originalmente la idea era enviar 4 datos de 32 bits o acceder directamente a la memoria del periférico por medio de un selector, pero lamentablemente esto no se pudo y tuvimos que trabajar solo con los 32 bits que logramos obtener del proceso en cascada AXI con ambos ATG.

V. CONCLUSIONES(0.2)

Enumere las conclusiones más importante de su proyecto. Recuerde que:

- Con la transmisión de 32 bits ocurre ocasionalmente un error en el cual parte de la secuencias que enviábamos por AXi Full no siempre llegaban correctamente al circuito final, llegando en vez de los 32 bits en hexadecimal "62199988" secuencias diferentes a las efectivamente enviada, esto se atribuye a un algún modulo no AXI con un error que no pudimos encontrar.
- Con la elaboración del proyecto pudimos notar la complejidad que significa trabajar en la edición de un periférico AXI, siendo el caso de Full particularmente desafiante, pues implicaba cambiar estructura desconocidas para tratar de obtener la información solicitada.

Si sigue estas reglas para escribir las conclusiones tendrá los 0.25 puntos.

VI. TRABAJOS FUTUROS (0.2)

Para posibles iteraciones futuras de este proyecto o parecidos, se hace fundamental establecer la maquina de estados como eje fundamental del proyecto, pues en nuestra realización nos enfrentamos a diversos problemas que nos obligaron a cambiar múltiples veces este módulo.

En cuanto construir sobre nuestro proyecto, se presentan las siguientes oportunidades:

Aumentar largo del patrón a seguir: Para esto se deberían alterar las memorias RAM simples para tener mas bits que representen dicha secuencia.

Obtener acceso a la memoria interna del periférico: puede ser pertinente solventar el adquirir acceso a las memorias del periférico AXI Full de manera directa, para implementar la RAM dentro del mismo Ip-core pues este tiene la estructura interna ya existente, pero no pudimos acceder a ella.

REFERENCES

[1] G. O. Young, "Synthetic structure of industrial plastics," in Plastics, 2nd ed., vol. 3, J. Peters, Ed. New York, NY, USA: McGraw-Hill, 1964, pp. 15–64.
[2] W.-K. Chen, Linear Networks and Systems. Belmont, CA, USA: Wadsworth, 1993, pp. 123–135.
[3] J. U. Duncombe, "Infrared navigation—Part I: An assessment of feasibility," IEEE Trans. Electron Devices, vol. ED-11, no. 1, pp. 34–39, Jan. 1959, 10.1109/TED.2016.2628402.
[4] E. P. Wigner, "Theory of traveling-wave optical laser," Phys. Rev., vol. 134, pp. A635–A646, Dec. 1965.
[5] E. H. Miller, "A note on reflector arrays," IEEE Trans. Antennas Propagat., to be published.