Fecha de entrega 9 de Mayo de 2023.

Zyborice

IGOR ARAUS[1], AGUSTÍN BUDD[1]
[1]Pontificia Universidad Católica de Chile (e-mail: igor.agustin@uc.cl, agustin.budd@uc.cl)

SI 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 El proyecto consiste en la creación de un juego llamado "Memorice", el cual muestra una secuencia de luces que el jugador debe repetir correctamente para ganar. El juego tiene cuatro niveles de dificultad, donde cada nivel tiene su propia secuencia predefinida, almacenada en una RAM. El diseño del proyecto incluye la creación de una máquina de estados para controlar los modos del juego, un IP Core que controla la lógica principal del juego, y un IP Core PWM que controla un LED RGB de la ZYBO Z7-10. Además, se utilizó comunicación AXI para cargar las secuencias de leds en una RAM, desde un ATG en modo prueba. También se implementó un ATG en modo avanzado para guardar en una RAM las ponderaciones de puntajes según el nivel seleccionado, los cuales se sumarán al puntaje obtenido en la ronda. En caso de perder, se muestra nuevamente la secuencia, y si se gana, se despliega el puntaje en los leds y vuelve al inicio del juego. Los resultados muestran que el juego funciona correctamente y cumple con los objetivos propuestos. El jugador puede elegir diferentes niveles de dificultad y el puntaje se actualiza en tiempo real. El uso de tecnologías como la comunicación AXI y la creación de IP cores, utilizando la herramienta block design permitió una implementación eficiente y robusta del juego. En conclusión, se logró crear un juego divertido y desafiante, utilizando tecnologías avanzadas de diseño de sistemas digitales.

INDEX TERMS Memorice, VHDL, Vivado, Block design, Máquina de estados, Código secuencial y recurrente, FPGA, ZYBO z7-10

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

EL proyecto presentado consiste en el diseño e implementación de un juego de memorización llamado "Memorice" en la FPGA de una placa ZYBO z7-10. El objetivo del juego es memorizar una secuencia de 4 LEDs mostrada en la placa y luego replicarla presionando 4 botones disponibles. El juego cuenta con 4 niveles distintos, donde cada uno tiene una secuencia distinta y una dificultad asociada, la cual es la velocidad con la que se prenden y apagan los LEDS, además de tener un ponderador de puntaje asociado a cada nivel.

Al iniciar el juego, se encienden 4 LEDs, indicando que la tarjeta está en modo de espera. Luego, al activar y desactivar el switch "sw2", se avanza al siguiente nivel y se encienden 3 LEDs para indicar que se ha entrado en el juego. Posteriormente, al accionar nuevamente el switch "sw2", se avanza a otro estado donde se espera que se ingrese el nivel deseado mediante los switches "sw0" y "sw1". Al accionar nuevamente el switch "sw2", se comienza a mostrar la secuencia del nivel correspondiente.
Una vez que se muestra la secuencia, se enciende una luz azul mediante un LED RGB disponible en la placa para indicar que el jugador debe presionar los botones en el orden mostrado previamente. Si el jugador falla, se enciende una luz roja y se vuelve a mostrar la secuencia para volver a intentarlo. En caso de ganar, se muestra solo una luz verde durante un segundo y luego, sin apagar el RGB en color verde, por tres segundos se muestra el puntaje de la ronda en binario en los LEDs, según el nivel, la dificultad y el ponderador. El puntaje máximo que se puede alcanzar es b'1110, mientras que el mínimo es b'0100. Si el jugador desea volver al comienzo del juego, puede subir y bajar el switch "sw3" que funciona como reset, o subir el switch "sw2" que permite cambiar de estado en cualquier momento.

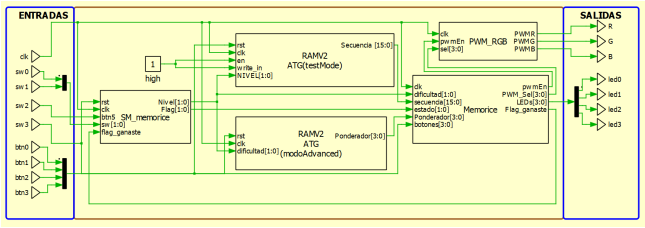

FIGURE 1: Diagrama de bloques detallado

La estructura del proyecto se resume en cinco grupos, compuesto de tres módulos y dos conjuntos de módulos, mostrados como bloques en la figura **1**, junto con las entradas y salidas respectivas. A continuación se muestra un listado de cada grupo/módulo, junto con una breve descripción del flujo de información dentro del proyecto:

- **SM_MEMORICE:** Es uno de los módulos principales del juego, en formato de IPCore, donde se gestiona el menú del juego, mediante una máquina de estados, controlada por el switch "sw2", transitando por estado de espera, el inicio de menú del juego, la selección de nivel y dificultad, y por último, el inicio de juego, que despliega la secuencia. Este módulo entrega la información de nivel y dificultad a los grupos de módulos RAM y al módulo memorice, al que también entrega el estado actual.

- **RAM(ATG Advanced Mode):** Este grupo de módulos se encarga de entregar el ponderador de puntaje asociado a un nivel, y está compuesto por tres IPCores distintos. El primero es un ATG en "test Mode" que tiene la función de cargar instrucciones de escritura en la CommandRAM y una data de "ponderadores de puntaje" en la MasterRAM de un ATG en "advance mode". Este último ATG corresponde al segundo IPCore, que una vez inicializado carga la data en un tercer IPCore esclavo, que contiene una memoria RAM. Este último IPCore recibe la difultad, y en base a esto, entrega el ponderador de puntaje correspondiente al módulo "MEMORICE" descrito en los siguientes puntos. Este último IPCore se basó en parte al trabajo desarrollado en [1].

- **RAM(ATG Test Mode):** Este conjunto de módulos se encarga de entregar todas las secuencias de prendido de los LEDs para el nivel asociado en un vector de largo 16. Está compuesto de 2 IPcores, un IPCore es un ATG en "test mode" que al inicializar carga la data de secuencia en un segundo IPCore esclavo que almacena en registros la data, y entrega al módulo "MEMORICE" la secuencia. Este último IPCore fue adaptado de la ayudantía 4 del Curso.

- **MEMORICE:** Este IPCore corresponde al bloque más importante para el desarrollo del juego, pues dentro está la lógica para mostrar la secuencia de LEDs según el nivel y dificultad escogida, se recibe y maneja la secuencia de botones ingresada por el jugador, se determina el puntaje y si se ganó o perdió. Este bloque recibe todas las señales entregadas por la Máquina de estados "SM_MEMORICE", además del ponderador de puntaje y la secuencia de LEDs a mostrar desde los módulos RAM mencionados en los puntos anteriores. Como salidas, genera una señal que es recibida por la "SM_MEMORICE" para avisar que el jugador ganó el nivel, y así volver al estado de espera; la salida para los LEDS, que variará según el estado del juego en el que se encuentre (mostrar la secuencia, apretar los botones y mostrar el puntaje); y enviar datos de control al módulo "PWM_RGB", descrito a continuación. Este bloque contiene varios components, functions y procedures, que en conjunto logran hacer funcionar la lógica deseada, los cuales serán mencionados y detallados mejor en la section II.

- **PWM_RGB:** Este último módulo IPCore, corresponde a un controlador PWM, que recibe las señales del módulo "MEMORICE", y muestra en el LED RGB de la tarjeta los colores deseados. El Módulo fue extraido y ligeramente adaptado de los ejemplos vistos en la ayudantía 3 del curso.

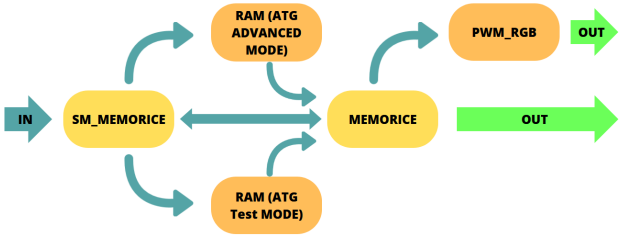

FIGURE 2: Diagrama de flujo

En resumen, se muestra en la figura 2 el diagrama de flujo, para entender las direcciones del flujo de datos y señales entre los módulos descritos en los puntos anteriores.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): *Descripción:* El código cumple con las funciones que habíamos planteado en nuestro proyecto las cuale son: XXXX. De acuerdo a la arquitectura presentada en la sección I, esto se implementó en los bloques XXX, cuya funcionalidad fue XXXX. *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.
AO1 (100%): *Descripción:* El código cumple con la implementación completa y funcional de **3 components** dentro de una entity. Esto se logra en el módulo *memorice* que se observa en las figuras 1 y 2, ya que utiliza los components

btn_sample (para que los botones tengan efecto por un solo período de clock) y mux_clock (código concurrente que elije la frecuencia del clock), luego, dentro de este último, se instancian 4 componentes que tienen la misma función de dividir la frecuencia del reloj, pero con distintas frecuencias según la entrada de mux_clock. *Nivel de Logro:* 100%. Se logró completamente, el código simula lo esperado, sintetiza e implementa sin problemas, pudiendo también generar el bitstream al unirlo con el resto de los bloques. Además, antes de juntarlo con la totalidad del proyecto, se le hicieron modificaciones para probarlo por si solo en la ZYBO y realizaba lo esperado.

AO2 (100%): *Descripción:* El proyecto en su totalidad logra implementar **5 IP CORES** principales con parámetros génericos, los cuales sintetizan, implementan y cargan bitsream sin problemas. Los **5 IP CORES** creados por nosotros se pueden observar en la figura 1, $SM\_memorice$ con parámetros genérico la cantidad de switches; $memorice$ con parámetros genéricos la cantidad de switches (NUM_SW), leds (NUM_LEDS), largo de la información de la RAM($LARGO\_INFO\_RAM$), la cuenta máxima para el tiempo que dura encendido el PWM ($CUENTA\_PWM$) y para el tiempo que se muestra el puntaje de la ronda ($CUENTA\_PUNTAJE$); y por último, el módulo PWM_RGB que tiene parámetros genéricos el número de colores máximo que puede mostrar (en largo de bits), el cual es $CANTIDAD\_COLORES$.
Los dos restantes, son las memorias RAM que se encuentran dentro de los bloques **RAMV2 ATG(TEST MODE)** y **RAMV2 ATG(ADVANCED MODE)**, que se crearon como periféricos AXI con los parámetros genéricos que estos traen por defecto, siendo modificados para que cumpla nuestros requerimentos. *Nivel de Logro:* 100%. Todos los **IP CORES** funcionan en su totalidad y en la práctica no tuvieron problemas en la sintetización, implementación y bitstream. En la teoría se tiene un critical warning sobre los tiempos de requerimento durante la implementación, pero en la práctica y efectos de esto proyecto no influye.

AO3 (100%) *Descripción:* Mediante el bloque **RAMV2 ATG(TEST MODE)** de la figura 1 se logró enviar mediante un ATG maestro en test mode las secuencias del nivel (elegido según la entrada "sw") en un solo vector, hacia un IP CORE esclavo, el cual es una memoria RAM que escribe y guarda la secuencia de leds asociada al nivel seleccionado, para así, enviarla al módulo $memorice$. Esto se puede observar en la figura 3.

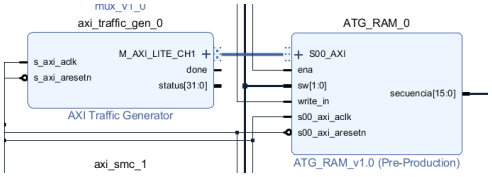

FIGURE 3: ATG test mode

Luego, mediante el bloque **RAMV2 ATG(ADVANCED MODE)** de la figura 1 se logró enviar mediante un ATG maestro en advanced mode un ponderador de puntaje a $memorice$, el cual varía según la entrada "nivel", tal como se observa en la figura 4. Es importante mencionar que para poder realizar esto, es necesario un ATG en test mode que configure en el ATG advanced con archivos .coe, para así cargar las instrucciones que realizará el ATG en modo avanzado, esto se observa en la figura 5.

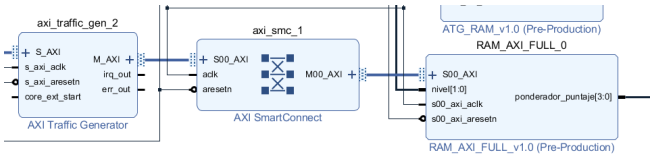

FIGURE 4: ATG advanced mode

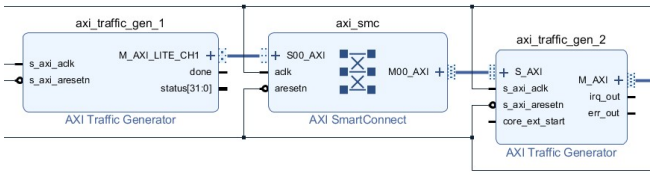

FIGURE 5: ATG TEST de configuración

Nivel de Logro: 100% Fue posible sintetizar e implementar estos bloques en conjunto al resto del proyecto, consiguiendo el resultado esperado al generar el bitstream y cargarlo en la ZYBO.

AC1 (100%) : *Descripción:* El módulo $SM\_memorice$ de la figura 1 es una máquina de estados controlada por la entrada, es decir, por cada vez que se eleve el switch, esta cambiará de estado, enviando de salida en que estado se encueetra (estos fueron explicados en la sección I). También $memorice$ es una máquina de estados, con la diferencia que fue hecha a base de condicionales if, esta es controlada por el estado que le llegue de $SM\_memorice$ y también por la acción del jugador al apretar los botones, ya que dependiendo de si acierta o falla, se moverá a uno u otro estado. Específicamente, si el estado de $SM\_memorice$ es "00", la salida serán los 4 leds encendidos; si es "01" la salida serán los 3 leds encendidos de izquierda a derecha; si el estado es "10" serán los 2 primeros leds; por último, si es "11", se inicia el juego y mostrará la secuencia de leds entregada por la RAM, para que cuando termine, pase al estado de jugar, que interpretará los botones que el usuario apriete, llegando a un estado de perder donde muestra el RGB en color rojo y luego vuelve al estado de mostrar la secuencia, mientras que si acierta, muestra el color verde y el puntaje en los 4 leds, para luego enviar una salida a la primera máquina de estados para volver al inicio del juego.
Nivel de Logro: 100%. Las dos máquinas de estado funcionan

correctamente enviando lo correcto según el estado en el que se encuentren, además de reaccionar como corresponde según los distintos estímulos, esto se cumple tanto para las simulaciones y paraa la implementación en la tarjeta.
AC2 (100%) : *Descripción:* Los 4 botones son utilizados para repetir la secuencia de leds del nivel puesto (realizado en el módulo $memorice$), los 4 switches son utilizados para el reset, cambio de estado y, los dos últimos, para la selección de nivel y dificultad ($SM_memorice$), de izquierda a derecha respectivamente. Los 4 leds son utilizados para mostrar la secuencia ($memorice$), el estado de $SM\_memorice$, y el puntaje de la ronda ($memorice$). Además los LEDS RGB muestran azul que indica el momento de apretar los botones, rojo si fallas, y verde si aciertas la secuencia ($memorice$).
Nivel de Logro: 100%. Todas las entradas de switches y botones, junto a la salida de los LEDS funcionan correctamente en la implementación del proyecto en la ZYBO.

AC3 (100%) : *Descripción:* Fueron utilizados los operadores $=$, $\neq$, $+$, $<=$ (asignación), $<$, $\leq$, $\geq$, entre otros, durante la totalidad del proyecto. Sobre los atributos, fueron utilizados justo 5, en el módulo $memorice$, para la función "conv_integer" se ocuparon **HIGH**, **LOW**, **LENGTH**, mientras que para hacer rising_edge(clk), se reemplazó por: if not clk'**STABLE** and clk = '1'. Para el faltante, en el módulo $SM\_memorice$ se utilizó clk'**EVENT** and clk = '1'.
Nivel de logro: 100%. Fue posible implementar 5 atributos y operadores diferentes en la totalidad del proyecto.

AC4 (100%) : *Descripción:* En el módulo $memorice$ fueron utilizadas constantemente las variables para actualizar a estas mismas dentro de la lógica, además de actualizar las salidas mediante estas mismas, para así lograr que la actualización de los valores no tenga que esperar a que termine el proceso.
Nivel de Logro: 100%. Fue posible implementar las variables en el proyecto y permitir menores retrasos de tiempo a como podría haber sido con señales.

AC5 (100%) : *Descripción:* El código secuencial es utilizado en la totalidad del proyecto, ya que todo es controlado por un reloj (máquinas de estado, axi full y lite), ya sea reducido o el base de la tarjeta. Excepcionalmente en el módulo $memorice$ se llama a un componente llamado mux_clk, que a pesar de que recibe la entrada del reloj, no es usada por él, sino que es ocupada por los divisores de clock que este llama como componentes. La funcionalidad que tiene este módulo es estrictamente concurrente, ya que no es controlado por un clock, por ejemplo, si el clock no existiera, este seguiría llamando según la entrada que llegue al componente requerido, con la diferencia que lo que no funcionaría sería el divisor de clock, debido a que este si es secuencial y necesita un clock para funcionar.
Nivel de Logro: 100%. Como se explicó fue correctamente realizado y no existieron problemas en la sintesís e implementación, el divisor de clock funciona correctamente y el multiplexor es capaz de elegir que frecuencia ocupar según la entrada.

AC6 (100%) : *Descripción:* En el módulo $MEMORICE$ se crearon 2 procedures: $suma1$, este es un procedure simple que "modulariza" la acción de sumarle un 1 a una misma variable, lo que hace más legible el código; $arreglo_puntaje$, este lo que hace es como dice su nombre, arreglar el puntaje cuando el nivel seleccionado es el "00", ya que si no se realiza, el puntaje obtenido sería 0, por lo que para cuando el nivel es "00", se asigna a la salida de este procedure "01", mientras que mantiene la salida para cualquier otro nivel.
Para el uso de functions, se creó $conv\_integer$, extraída de [2], esta lo que hace es transformar un standar logic vector en su valor entero, entonces así es posible convertir el nivel que llega como vector en un entero para poder sumarlo.
A pesar de que en este caso tanto los procedures como las functions tienen una sola salida, es posible diferenciar claramente que si se requiriera, el procedure podría tener más salidas, mientras que la función solo puede tener una, que es la dada por "return".
Nivel de Logro: 100%. Fue posible implementar procedures y functions en el código de $memorice$, funcionando correctamente al momento de las simulaciones y la implementación, ya que se obtuvo lo esperado.

AC7 (100%) : *Descripción:* Como se observa en la figura 4, para poder conectar el AXI Traffic Generator en advanced mode con un IPCore esclavo, es necesario un bloque entre medio, el cual es AXI Smart connect, este bloque es una caja negra que permite la conexión entre un puerto maestro en advanced con un puerto esclavo en test y viceversa.
Nivel de Logro: 100%. Fue posible implementar este bloque extra permitiendo la conexión de los modos advanced y test de distintos bloques. La implementación en la ZYBO funcionó correctamente y se reciben los datos correctos.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

A continuación se mostrarán algunos resultados de la simulación post implementación del módulo memorice, el cual contiene la lógica principal que permite el funcionamiento del hardware en su totalidad. Se realiza esta simulación con el fin de observar retrasos en la actualización de variables y sus efectos, lo cual es algo que no ocurre en las simulaciones de comportamiento.

En la figura 6 la variable $contadorbtn$ debería activarse en el flanco de subida del reloj que viene primero luego del accionamiento de $btn1$, pero debido a que en la realidad existen retrasos debido a la cantidad de compuertas que puedan haber y su distancia entre ellas, esta es activada con un retraso de $15\ ns$ aproximadamente.

En la figura 7 es posible observar que este delay de actualización de variables, puede provocar una fluctuación entre el cambio del valor, esto se puede observar en que, para que

contadorbtn pase de 1 a 2, primero pasa por 3, esto es fácil de entender si nos damos cuenta que en bits, debe pasar de 001 a 010, por lo que los retrasos hacen que se actualize primero el cero a uno, y luego el uno al cero.

La razón explicada de esta oscilación, es posible demostrarla con el caso contrario, en la figura 10 es posible observar que ocurrió el retraso en el efecto, pero la oscilación no existe. Esto es debido a que el contador pasó de 2 a 3, es decir, de "010" a "011", por lo que solo tuvo que actualizar el bit que pasó de 0 a 1, sin provocar esa oscilación.

Estas simulaciones también permiten darse cuenta de la razón por la cual se recomienda usar variables para actualizar valores por encima de las señales. En las figuras 8 y 9 se observa la comparación de la salida actualizada con variables frente a la misma, pero asignada mediante señales. A pesar de que tienen un comportamiento parecido, ya que en los dos casos existe una fluctuación, si es posible darse cuenta que la salida mediante variables se actualiza antes que la asignada por señales, esto debido a que las señales se actualizan cuando el proceso termina, mientras que las variables son actualizadas instantáneamente durante el proceso.

Para efectos de el hardware descrito en este módulo, al no ser un código muy complejo y largo, la diferencia es bastante despreciable (del orden de 100 ps), pero en casos donde los códigos sean más largos y complejos, los retrasos en cadena podrían hacer que el funcionamiento no sea el esperado.

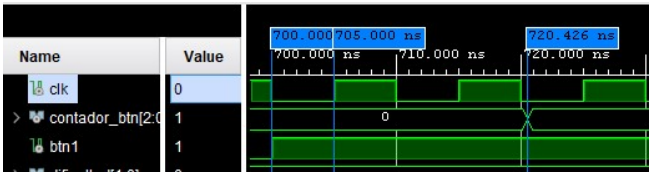

FIGURE 6: Delay en la variable contador de botón.

FIGURE 7: Efecto delay de una variable.

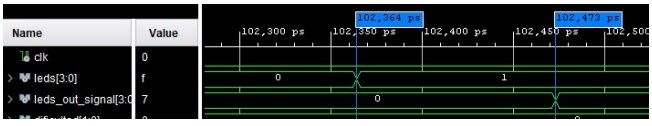

FIGURE 8: Actualización de señales v/s variable parte 1.

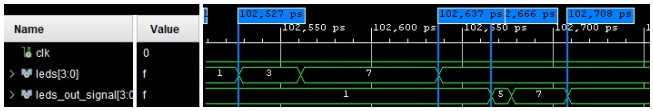

FIGURE 9: Actualización de señales v/s variable parte 2.

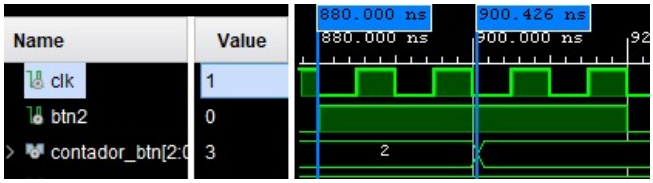

FIGURE 10: Delay de otro botón

Para comprobar el correcto funcionamiento del protocolo AXI LITE y AXI FULL en nuestro proyecto, se escogió los módulos del grupo RAM(ATG ADVANCED), ya que en ellos se realiza una operación con AXI FULL y también una operación con AXI LITE, como se describió anteriormente. Las pruebas se realizarón en los módulos correspondientes al ejemplo que viene incluido en los IPCores de "AXI traffic generation", de manera de poner visualizar mejor las transacciones con el testbench implementado en él.
Los archivos coe que fueron cargados en el ATG en test mode, que inicializa y carga el ATG en advance mode, contenían básicamente una línea de lectura de comprobación para que el ATG en test mode compare el modelo del otro ATG, y luego le siguen 4 líneas de instrucciones que deben ser escritas en la CommandRAM del otro ATG, finalmente, una data enviada que es un h'aaaaaaaa y un comando para iniciar la operación del ATG en advanced mode. En las simulaciones post implementación se obtuvo lo que se muestra en las figuras 11,12,13,14,15.

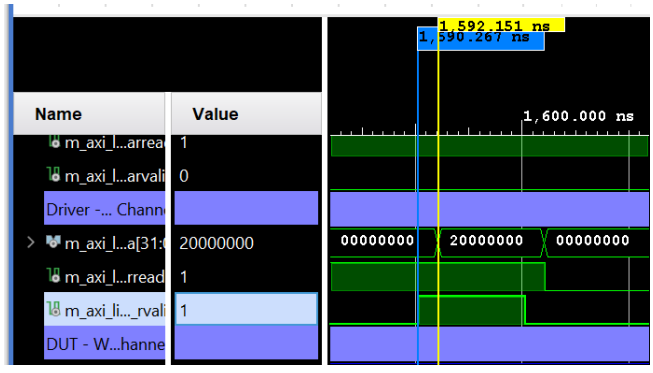

FIGURE 11: Lectura inicial ATG TEST MODE - AXI lite

En la figura 11 se muestra que se recibe un flanco de subida de la señal rvalid, que indica que se habilita la lectura. Luego se recibe un 2000000, que corresponde a lo esperado, pues es la lectura de comprobación inicial. Se puede notar que existe un desfase entre la el flanco de subida y cuando inicia la lectura, debido a la implementación real, el cual es del orden de 2ns.

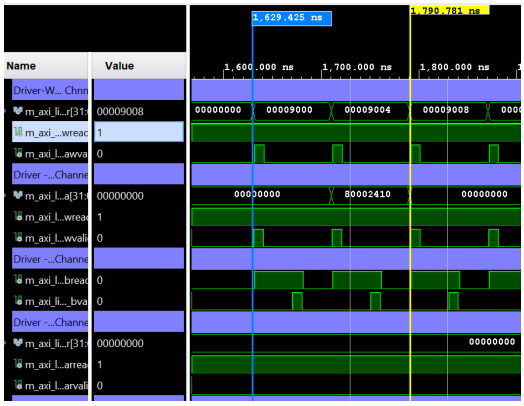

FIGURE 12: Escritura en la CommandRAM de ATG en Modo advanced

Luego, en la figura 12 se puede ver que se activa el canal de escritura, y se escribe en la dirección 9000 4 datos, que corresponden a las instrucciones de escritura. Se escriben en la dirección 9000 porque esa es la parte de la CommandRam que contiene ese tipo de instrucción.

FIGURE 13: Escritura de dato h'aaaaaaaa en C0000

Posterior a esto, en la figura 13 se puede ver que en la dirección c0000 se escribe un dato h'AAAAAAAA. Esto es lo esperado, pues la dirección c0000 es donde comienza la memoria de la MasterRAM del ATG en advanced mode.

FIGURE 14: Inicio operación de ATG ADVANCED

Finalmente, en la figura 14, el ATG en test mode escribe en la dirección h'00000000 la data h'00100000, lo que inicia la operación del ATG en Advanced Mode, finalizando la transacción mediante protocolo AXI LITE. Luego de esto, marcado con el marcador amarillo, se indica en el canal de WRITE Address del ATG en modo ADVANCED, que comienza la operación de escritura del ATG hacia la RAM externa, mediante AXI FULL. En este caso, se levanta una flag que indica que se mandará una ráfaga de un dato.

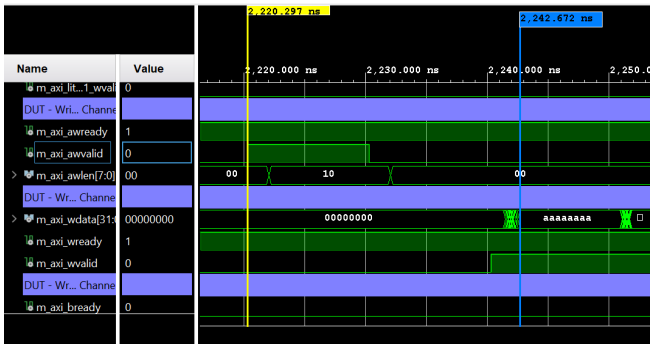

FIGURE 15: Escritura de ATG ADVANCED en dirrección 0 de la RAM

Luego, en la figura 15, el ATG en modo advanced escribe en la dirección 0 de la RAM el dato h'aaaaaaaa, finalizando la transacción en AXI FULL del ATG en modo advanced en la RAM. Con esto se comprueba que la simulación de implementación funciona correctamente y no se pierden datos que son parte del mensaje que se espera recibir. Se comprueba además que los datos que implementamos en los archivos coe, junto con las instrucciones fueron guardados en la sección de memoria del ATG en modo ADVANCED adecuada. El mensaje que finalmente enviamos en nuestro proyecto no es el usado en la simulación, pero su adaptación no influye en las pruebas realizadas.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

En un principio, la implementación tenía pensado el uso de los 6 botones que trae la ZYBO, pero al momento de realizar la implementación, existieron problemas con los botones MIO50 y MIO51, ya que estos se encuentran conectados al microprocesador, y para acceder a ellos se requiere un uso que supera el trabajar con la lógica programable. En consecuencia, se tuvo que adaptar el uso de esos 2 botones a 2 switches, afectando a la idea principal que se tenía de ocupar 2 para dificultad del juego y 2 para el nivel a jugar, por lo que ahora el nivel y la dificultad van de la mano con los mismos switch.

Otra consecuencia fue que para poder escribir en las memorias ram mediante axi full y lite, se tuvo que realizar internamente un módulo que esté constantemente enviando un 1 a los enable y write in de estos estos, para así evitar el uso de botones o switches que permitieran la escritura

en las memorias. Luego de solucionar estos problemas, la implementación funcionó tal como se esperaba.

V. CONCLUSIONES(0.2)

Las conclusiones más relevantes del proyecto son las siguientes:

1) El $btn1$ no se actualiza en el flanco de subida del reloj como se esperaba, si no que después de 15 ns. Esta desfase se debió a la cantidad de compuertas y la distancia física entre ellas.
2) La diferencia entre el uso de variables y de señales con respecto al tiempo de actualización es del orden de los $100ps$. Presentando un menor desfase lo realizado con variables.
3) Para la comunicación con el protocolo AXI FULL, se esperaba recibir 32 bits de datos, correspondientes a un mensaje, que contiene los ponderadores por nivel en los 16 bits menos significativos. Se recibieron los 32 bits de información esperados.
4) Para la comunicación con el protocolo AXI LITE, se esperaba recibir 64 bits distribuidos en 4 mensajes. Se almacenaron en la RAM y se recibieron los 64 bits de datos, correspondientes a las secuencias.

VI. TRABAJOS FUTUROS (0.2)

Para trabajos futuros se suguieren las siguientes ideas:

- Se puede incluir al proyecto a utilización del microprocesador Zynq, que viene dentro de la placa de trabajo Zybo Z7-10, de manera de poder ocupar los botones MIO50 y MIO51, liberando dos switches, permitiendo mayor cantidad de niveles a jugar, como también dificultades.

- Otra idea puede ser que los IPCores asociados a salidas tengan puertos AXI, de manera que en un futuro sea posible conectar salidas de alguno de los bloques a un periférico disponible en la tarjeta.
- Optimizar los bloques del juego y la lógica, para optimizar su respuesta en el tiempo. Esto puede mejorarse agregando mayor cantidad de variables y disminuyendo las señales.
- Se puede aumentar el largo de los bits de secuencias, y en vez de usar 4 secuencias, usar 8. De esta manera aumentar la dificultad del juego.

REFERENCES

[1] M. B. Aykenar, “HOW TO CREATE an AXI4-FULL CUSTOM IP with AXI4-LITE and UART INTERFACES in VIVADO”. Recuperado de: https://www.mehmetburakaykenar.com
[2] Clases VHDL curso sistemas electrónicos progamables, “FUNCTIONS and PROCEDURES SYSTEM DESIGN” pp. 4.