Fecha de entrega 08 de mayo de 2023.

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

Juego Caliente/Frío en Zybo Z7-10

GUSTAVO BARREZUETA[1], CATALINA SIERRA [1]

[1]Pontificia Universidad Católica de Chile (e-mail:gabarrezueta@uc.cl, catalina.sierra@uc.cl)

Autorizo que mi proyecto (tal como ha sido entregado, sin nota ni comentarios de evaluación) sea publicado en un repositorio para que 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 presente proyecto consiste en un juego que se basa en encontrar a lo largo de un tablero de 5x5 un objetivo que se encuentra en una posición al azar, observando el color que indica un led RGB. El proyecto fue desarrollado en el lenguaje de descripción de hardware VHDL, haciéndose uso de IPs y protocolos de comunicación AXI-lite y AXI-full; y este se implementó en la tarjeta Zybo Z7-10, haciendo uso de sus 4 switches, 4 leds, 4 botones y led RGB. El juego se logró implementar exitosamente, permitiendo que el usuario se pueda desplazar con los botones y guiar mediante un led que transiciona de rojo a verde a medida que se va acercando al objetivo, además de los switches que permiten reinicio y acceso a información como coordenada X, coordenada Y y distancia total al objetivo.

INDEX TERMS VHDL, Zybo, Zybo Z7-10, VHDL project, AXI4, IP design, Vivado block design

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

En la arquitectura de hardware simplificada mostrada en la figura 1 encontramos 5 bloques principales, además de las entradas y salidas (que se señalan en color azul), los cuales se explican a continuación:

1) GAME_manager: Maneja la actualización de posición del jugador a medida que se desplaza con los botones, despliega en leds la información que se consulta con switches y calcula constantemente la distancia que se tiene respecto al objetivo, cuya posición ingresa como input semi aleatorio desde el generador de posiciones. Una breve descripción sobre cómo desplazarse en el juego y cómo consultar información:
   - btn3 → movimiento vertical hacia arriba
   - btn2 → movimiento vertical hacia abajo
   - btn1 → movimiento horizontal hacia la izquierda
   - btn0 → movimiento horizontal hacia la derecha
   - sw3 → resetea el juego
   - sw2 → consulta distancia del objetivo
   - sw1 → consulta la coordenada X jugador
   - sw0 → consulta coordenada Y jugador

   Siendo relevante levantar cada switch de forma individual, es decir, sólo uno a la vez.
2) RGB_manager: Maneja la led RGB, generando señales PWM para canales R, G y B de acuerdo a distancia que calcula GAME_manager. Su funcionamiento está condicionado por un *flag* que depende de la transacción AXI-full que ocurre en el módulo Gatillante AXI-full.
3) Generador de posiciones con AXI-lite: Realiza transacciones AXI-lite hacia un módulo que almacena posiciones para el objetivo del juego, las cuales son leídas y entregadas a GAME_manager de acuerdo al valor de un contador (se entrega sólo 1 posición a la vez, la cual cambia cada vez que se resetea el juego).
4) Gatillante AXI-full para RGB_manager: Realiza una transacción de escritura hacia una RAM mediante protocolo AXI-full, cuyo éxito condiciona el *flag* que saca hacia RGB_manager.
5) Debouncers: Permiten evitar *glitches*.

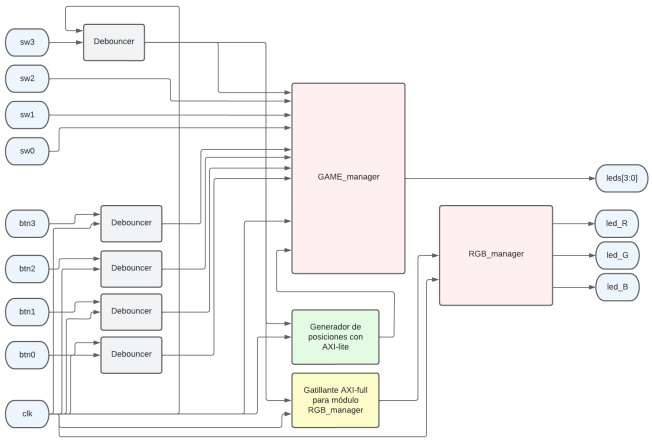

FIGURE 1: Diagrama explicativo del proyecto

A continuación se presenta el diagrama de bloques directamente desde el Block Design del proyecto en Vivado, donde se puede apreciar en mayor detalle el proyecto en general, además de los módulos con transacción AXI que fueron resumidos en el diagrama previo, siendo posible observar bloques como AXI Smart Connect, ATGs, IPs de memoria y Clocking Wizard.

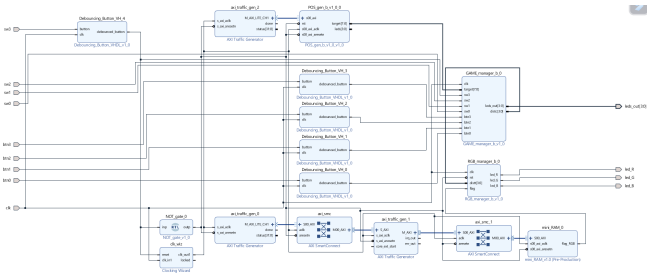

FIGURE 2: Block Design del proyecto en Vivado

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%):

Descripción: El proyecto cumple con haber utilizado `component` para instanciar al menos 3 componentes dentro de otra `entity`, lo cual lleva a cabo en el archivo final_design.vhd donde se instancian:

- final_design_axi_traffic_gen (3 en total)
- final_design_mini_RAM_0_0
- final_design_axi_smc (2 en total)
- final_design_clk_wiz
- final_design_NOT_gate_0_0
- final_design_POS_gen_b_v1_0_0_0
- final_design_Debouncing_Button_VH (5 en total)
- final_design_RGB_manager_b_0_0
- final_design_GAME_manager_b_0_0

Nivel de Logro: 100%. Completamente logrado, se instancian múltiples módulos IP usando `component`, siendo la mayoría creados por nosotros y algunos del repositorio de Vivado.

AO2 (100%):

Descripción: El proyecto cumple con haber generado al menos 3 IPs en Vivado block design para incorporar a nuestros códigos, además de utilizar parámetros genéricos en algunas de ellas. Algunos de los IPs generados fueron:

- RGB_manager_b
- GAME_manager_b (tiene como parámetro genérico `clk_max_cnt` para modificar frecuencia del clock que se genera en el módulo para parpadeo de leds)
- POS_gen_b
- mini_RAM

Nivel de Logro: 100%. Completamente logrado, se crean y empaquetan al menos 3 IPs en Vivado block design, las cuales son incorporadas a los códigos del proyecto.

AO3 (100%):

Descripción: El proyecto cumple con utilizar comunicación AXI para comunicar un IP core esclavo creado por el equipo con un ATG maestro en Test Mode, y un IP core esclavo creado por el equipo con un ATG maestro en Advanced Mode. Las conexiones entre módulos y ATGs respectivos se enlista a continuación y pueden ser notados explícitamente en el diagrama de la figura 11:

- axi_traffic_gen_2 (ATG maestro en **Test Mode**) y POS_gen_b_v1_0_0 (IP core esclavo creado con puerto de entrada **AXI-lite**)
- axi_traffic_gen_0 (ATG maestro en Test Mode que carga instrucciones), axi_traffic_gen_1 (ATG maestro en **Advanced Mode** que ejecuta instrucción que recibe en bloque mini_RAM) y mini_RAM_0 (IP core esclavo creado con puerto de entrada **AXI-full**)

Nivel de Logro: 100%. Completamente logrado.

AC1 (100%):

Descripción: El proyecto cumple con generar una máquina de estados, la cual está asociada a parte del proyecto. Específicamente en el módulo GAME_manager.vhd nos encontramos con una máquina de 2 estados, descrita por la variable `player_state`, que permite el funcionamiento del sistema de acuerdo a si el jugador está en estado *playing* (`0`) o *winner* (`1`), lo cual gatilla acciones específicas.

Nivel de Logro: 100%. Completamente logrado.

AC2 (100%):

Descripción: Se utilizan en el proyecto los 4 botones, 4 switches y 4 leds de la Zybo Z7-10 vistos en el curso.

Nivel de Logro: 100%. Completamente logrado.

AC3 (60%):

Descripción: El proyecto cumple con utilizar al menos 5 operadores, entre los que se encuentran `+`, `<`, `-`, `<=`, `:=` en GAME_manager.vhd, pero sólo se utiliza 1 atributo, no 5. El atributo utilizado es `'event` en el proceso ASSIGN_DIST de RGB_manager.vhd

Nivel de Logro: 60%. Parcialmente logrado.

AC4 (100%):

Descripción: El proyecto cumple con utilizar variables para actualizar valores instantáneamente en alguna parte de la lógica programada. En RGB_manager.vhd, en los procesos R_CTRL, G_CTRL y B_CTRL se utilizan variables para determinar el *duty cycle* para las PWM respectivas a cada canal del led RGB.

Nivel de Logro: 100%. Completamente logrado.

AC5 (100%):

Descripción: El proyecto cumple con utilizar código secuencial y concurrente. Esto se puede evidenciar, por ejemplo, en los procesos R_CTRL, G_CTRL y B_CTRL de RGB_manager.vhd, donde se utiliza código secuencial para actualizar los contadores según un *clock* y código concurrente para determinar el valor de la variable de conteo máximo, lo cual se realiza usando sentencia WHEN.

La diferencia entre código secuencial y concurrente es principalmente que este último es asíncrono, ya que la salida sólo depende del estado actual de las entradas, a diferencia de un circuito con lógica secuencial, donde la salida depende de

la historia de entradas pasadas, requiriendo memoria, además de un reloj.

Nivel de Logro: 100%. Completamente logrado.

AC6 (50%):

Descripción: El proyecto cumple con hacer uso de *functions* pero no de *procedures*. Se usan funciones de conversión como `to_integer` en módulo RGB_manager. Se intentó en el mismo implementar un *procedure* pero no funcionó, por lo que finalmente se excluyó.

Nivel de Logro: 50%. Parcialmente logrado.

AC7 (100%):

Descripción: El proyecto cumple con implementar alguna función no presentada en el curso. Se utilizó el led RGB de la Zybo para reflejar un espectro de color que transiciona de rojo a verde en función de la distancia a la que se está del objetivo en el juego, lo cual se realiza generando PWMs en base a la distancia horizontal y vertical.

Nivel de Logro: 100%. Completamente logrado.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

RGB_manager: Se llevaron a cabo **simulaciones post implementación de tiempo y comportamiento** para analizar los procesos R_CTRL, G_CTRL y B_CTRL del módulo, para identificar posibles retardos y observar la actualización de variables y señales.

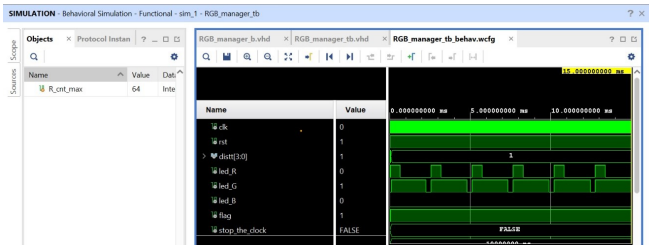

FIGURE 3: Simulación de comportamiento para procesos R_CTRL, G_CTRL y B_CTRL de RGB_manager

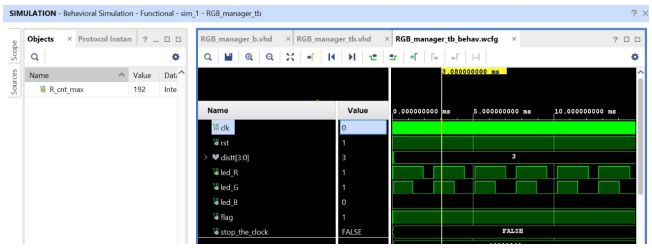

FIGURE 4: Simulación de comportamiento para procesos R_CTRL, G_CTRL y B_CTRL de RGB_manager

Se puede apreciar cómo las señales led_R, led_G y led_B generan PWMs de acuerdo a la distancia a la que nos encontramos del objetivo, siendo la distancia (distt) 1 en la figura 3 y 3 en la figura 4, pudiendo apreciarse como en el primer caso led_G tiene mayor *duty cycle* que led_R porque se busca desplegar un color verde ya que se está cerca del objetivo. En el segundo caso, al aumentar la distancia, led_G disminuye su *duty cycle* y led_R lo aumenta levemente, para poder desplegar un color más verde-amarillo.

Adicionalmente, se puede notar en ambos casos cómo la variable `R_cnt_max` se actualiza o cambia de valor (pasa de 64 a 192) de acuerdo a la distancia (pasa de 1 a 3), la cual aumenta de un caso a otro. Claramente que haya aumentado esta variable es lo esperado ya que al alejarnos del objetivo se tiende a mostrar en el led RGB un color cada vez más rojo y menos verde.

La figura 5 muestra la simulación de comportamiento del bloque RGB_manager. Se puede apreciar que en el instante 2.555 ms de la simulación, la señal led_R toma el valor 1. En la figura 6, que muestra la simulación de tiempo post-implementación, la misma señal toma el valor de 1 en el instante 2.555009520 ms, es decir, existe un *delay* de 9.520 nanosegundos.

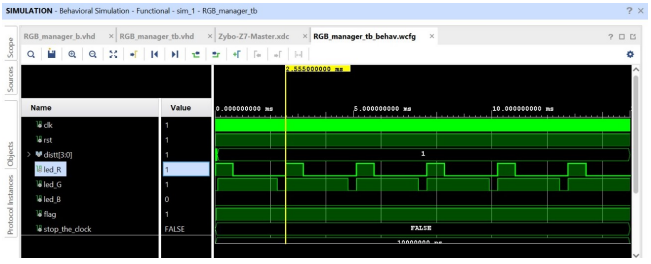

FIGURE 5: Simulación de comportamiento para procesos R_CTRL, G_CTRL y B_CTRL de RGB_manager

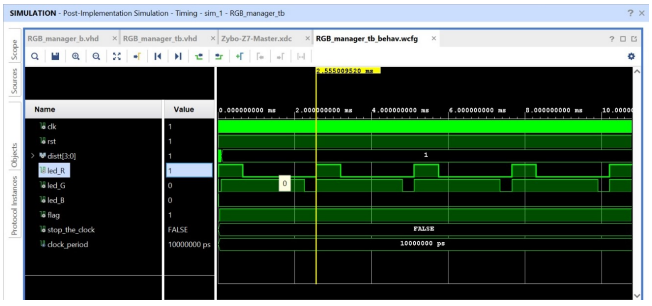

FIGURE 6: Simulación post implementación para procesos R_CTRL, G_CTRL y B_CTRL de RGB_manager

Adicionalmente, se obtuvo el reporte de la simulación de tiempo post implementación, donde se detalla el retardo neto para las variables asociadas al led RGB. En la figura 7 se muestran los *path* con los *delays* más importantes. Específicamente, en el diseño se muestra el *path* desde el botón de *rst* hasta el led_R. La relación entre estas dos señales se ve en el *proccess PWM_R_GEN* del bloque RGB_manager. La figura 8 muestra todo el camino que debe recorrer la señal iniciada en el switch 3 hasta que produce un efecto en el led_R.

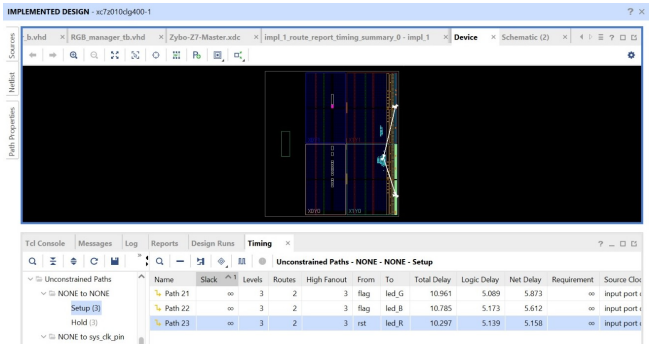

FIGURE 7: Timing Report para simulación post-implementación de RGBE_manager, con su *layout* incluido

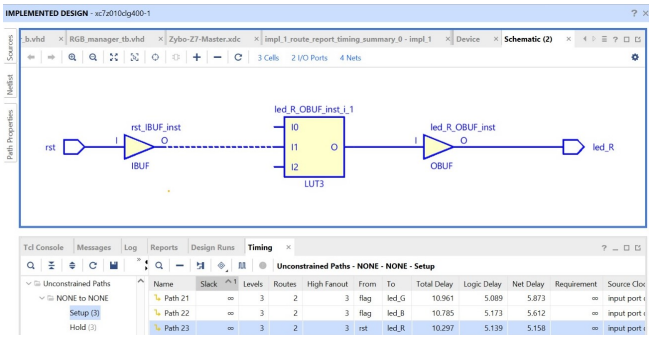

FIGURE 8: Timing Report para simulación post-implementación de RGB_manager, con circuito equivalente que representa el *path*

Por último, en la figura 9 se muestra el *max delay path* para *slacks*. En este caso, el *delay* es de 3.665ns que es inferior a los 9.520 ns que se obtuvieron en la simulación. Si bien, no se logró entender bien a qué se refieren los valores mostrados en la figura 9, se cree que la diferencia entre estos valores y la simulación se debe a que esta última posee procesos que son síncronos, por tanto, no solo se debe considerar los *delays* debido a *paths* sino también, el *delay* debido al *clock*.

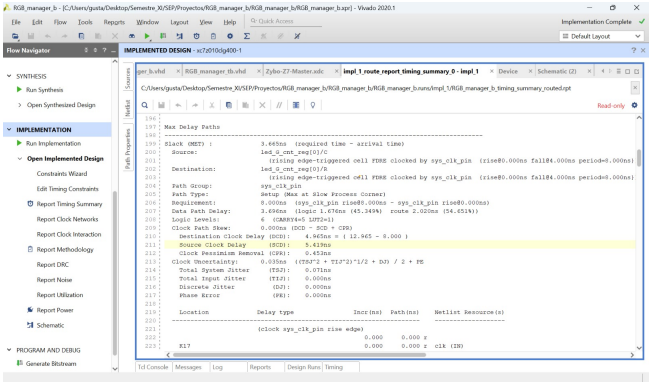

FIGURE 9: Timing Summary Report para simulación post-implementación de RGB_mana

GAME_manager: Por otro lado, se realizaron simulaciones de comportamiento al bloque más importante del circuito. El resultado se muestra en la figura 10. En la simulación, primero se simula un reseteo con el switch 3, a causa de lo anterior, la posició en x cambia de 1 a 0. Luego, se muestran las posiciones con los leds (primero la de y y luego la de x), notando que se obtiene 0 en *leds_out*. Seguido a lo anterior, se realizan dos movimientos hacia arriba con el botón 3. Luego, 4 movimiento hacia la derecha y, por último, 2 de nuevo hacia la derecha. En este último acto, se alcanza el objetivo definido en la posición $x = 4$, $y = 4$. Por lo que, se puede notar que los leds parpadean según lo esperado.

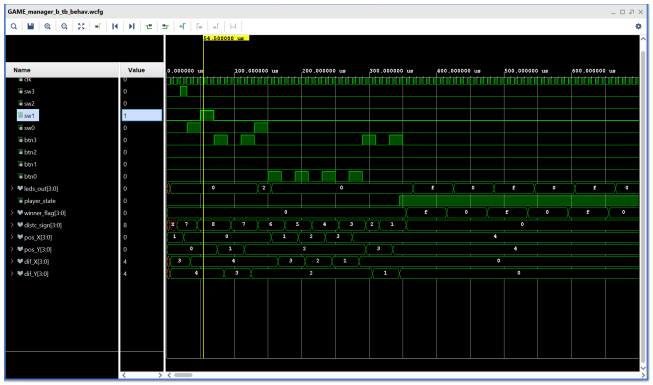

FIGURE 10: Simulación de comportamiento de bloque GAME_manager

Transacciones AXI: Se llevaron a cabo **simulaciones de comportamiento** para este efecto, las cuales se muestran ahora:

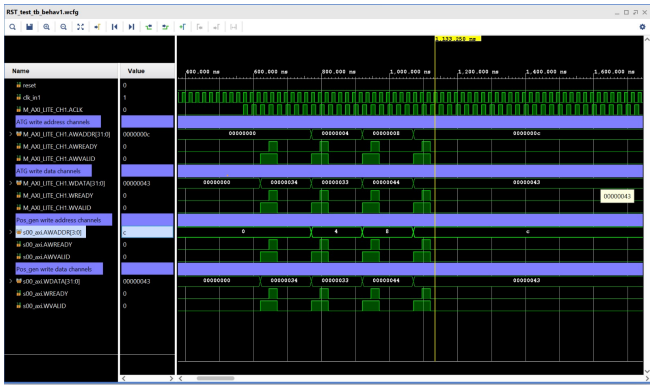

FIGURE 11: Simulación de comportamiento para bloques axi_traffic_gen_2 y POS_gen_b_v1_0_0 (AXI-lite)

Se puede apreciar claramente tanto en el ATG maestro como en el IP esclavo POS_gen que la transacción de información es exitosa, ya que se envían/reciben los datos de posición 34, 33, 44 y 43, que son las posiciones ""aleatorias"" que se escribieron en los archivos coe de configuración del ATG para almacenarlos en POS_gen_b; además de que estos son escritos en las direcciones de memoria especificadas 0, 4, 8 y C. También se puede apreciar cómo se levantan de

acuerdo a lo esperado los canales AWREADY y AWVALID, tanto del ATG como de la IP.

Cabe mencionar que, ya que sólo se implementaron transacciones de escritura mediante los archivos coe, sólo se muestran las variables de escritura relevantes.

Finalmente, no se logró realizar la simulación de comportamiento para el caso de los módulos axi_traffic_gen_0 (ATG maestro en Test Mode que carga instrucciones), axi_traffic_gen_1 (ATG maestro en **Advanced Mode** que ejecuta instrucción que recibe en bloque mini_RAM) y mini_RAM_0 (IP core esclavo creado con puerto de entrada **AXI-full**), ya que el programa arrojó un error de *mismatch* de parámetros (en el `generic`) que no se pudo resolver. De todas formas, el proyecto final funciona sin problemas y ello depende de que las transacciones de escritura AXI-full desde axi_traffic_gen_1 (ATG maestro en **Advanced Mode**) a mini_RAM_0 sean exitosas, ya que se levanta un *flag* de funcionamiento en base a lo anterior.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Las principales dificultades identificadas en la implementación fueron el *debounce* de los botones, además de un problema con la lógica del *reset* correspondiente al switch 3. Para solucionar estos problemas se incorporaron módulos *debouncer*, recuperados de [1], para evitar posibles *glitches* y señales auxiliares para retener el último estado de las variables, para simular lo que sería la condición *if rising_edge*, la cual no fue posible usar de forma directa al ya estarse usando para el *clk*.

Otra dificultad encontrada, la cual no se pudo resolver, fue poder implementar el *process* CALC_DIST en GAME_manager como un *procedure* definido por el equipo en la `architecture` del módulo. Si bien se logró que corriera sin problemas, la distancia no se calculaba correctamente, para lo cual no se encontró una causa.

V. CONCLUSIONES(0.25)

- Para el caso de la simulación de GAME_manager se notó que la señal que representa la distancia desde la posición al jugador hasta el objetivo se actualiza luego de dos *clocks* desde que se actualiza alguna de las posiciones. Este comportamiento no es el esperado, debido a que la actualización de la distancia se verifica en cada *rising_edge* del *clock*. Ampliando el tiempo de duración de cada ciclo del *clock* no cambia el fenómeno anterior, por lo que, se descarta la posibilidad que el problema sea que cálculo de la distancia se demora más de un ciclo del reloj.

VI. TRABAJOS FUTUROS (0.25)

Queda como trabajo futuro para el equipo:

- Lograr llevar el `process` CALC_DIST explicado en la sección IV a un `procedure` definido al incio de la arquitectura, que reciba las coordenadas de posición X e Y tanto del jugador como del objetivo, y que entregue la distancia total al objetivo.
- Ampliar el juego, de forma que el tablero en que se desplace el jugador sea de tamaño considerablemente mayor que 5x5, ojalá de 16x16 que es lo máximo que permiten reflejar las 4 leds de la Zybo.
- Cargar una lista de posiciones ""semi aleatorias"" de mayor extensión al bloque POS_gen para aumentar aún más el dinamismo y aleatoriedad del juego. Actualmente se cuenta con 4 posiciones posibles para el objetivo.
- Agregar “modos de juego”. Por ejemplo, implementar un modo de juego, en el que el jugador tenga que llegar a la mayor cantidad de posiciones objetivo en un tiempo limitado.

REFERENCES

[1] Loi Le, V. (s.f.). VHDL code for debouncing buttons on FPGA. fpga4student. https://www.fpga4student.com/2017/08/vhdl-code-for-debouncing-buttons-on-fpga.html
[2] AMD Xilinx. (s.f.). Xilinx Support. https://support.xilinx.com/s/?language=en_US
[3] Digilent Reference (s.f.). Edit the Address Map. https://digilent.com/reference/programmable-logic/guides/vivado-manual-address-editor