Fecha de entrega 9 de mayo de 2023.

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

Calculadora con juego del gato

MATÍAS SOLÍS[1], SOFÍA URZÚA[1]

[1]Pontificia Universidad Católica de Chile (e-mail: m.solis.lara@uc.cl, sofia.urzua@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 Este proyecto consiste en una calculadora lógico-aritmética con un juego “gato" integrado. Se utilizaron los 4 botones, 4 switches para que el usuario interactúe, y los 4 leds y el led RGB para que reciba información. Internamente, se utilizó comunicación AXI-Full en el ATG Advance Mode para una rutina preestablecida en la calculadora, y AXI-Lite para la configuración del ATG Advance Mode y el ATG Test Mode para la selección de colores de jugador en el juego. Además, se utilizaron IP-Core no revisados en el curso, tales como *DDS Compiler* y *Adder/Substracter*. En lo que respecta a la implementación, se lograron el 100% de las transacciones de información deseada para AXI-Full y AXI-Lite, pero no se lograron cumplir con los requisitos de tiempo establecidos por Vivado, de modo que la señal más lenta necesita de casi 7.5 ns estable para evitar errores. A pesar de lo anterior, el funcionamiento final del circuito implementado no presentó errores o bugs notables durante las pruebas.

INDEX TERMS VHDL, máquina de estados, calculadora, gato, *tic tac toe*, AXI.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

En este proyecto se realiza una calculadora con un juego “gato" integrado, inspirado en calculadoras como la Casio MG-880 [1], y la configuración en la Casio fx-991ES PLUS para jugar “gato" o *tic tac toe* [2]. Con eso en mente, el proyecto se puede separar en dos mini-proyectos: la calculadora y el gato, cada uno con un modo manual (estado por defecto) y un modo “AXI" en alguna de sus versiones, lite o full.

El proyecto alterna entre calculadora y “gato" por medio de la conservación del estado anterior del botón 2 (K19) de la ZYBOZ7 gracias al modulo *altState* [3], lo cual envía una señal a *package* que demultiplexa las entradas desde la ZYBOZ7 y multiplexa las salidas de cada mini-proyecto. En la figura 3 corresponde al modulo llamado **mux_supreme**.

Con respecto a la interacción para cada subproyecto, se ocupa universalmente el botón 3 (Y16) para resetear los datos ingresados. La presencia del reset se muestra en las conexiones de color rosado en el diagrama 3. Luego, para pasar al modo “AXI" se ocupa el botón 1 (P16) (en burdeo en el diagrama 3): para la calculadora, se carga una instrucción predefinida, cuyo resultado se mostrará en los 4 leds arriba de los switches y, para el juego, se cargan colores predeterminados para personalizar el led RGB. Para poder seleccionar o interactuar con valores ingresados manualmente a través de los 4 switches, ya sea números y operaciones para la calculadora, o posiciones y colores para el gato, se debe presionar el botón 0 (K18) (en verde agua en el diagrama 3). Este botón también sirve para recorrer la instrucción de secuencia Fibonacci. El modo manual está manejado por las máquinas de estado, mostradas en las figuras 1 y 2.

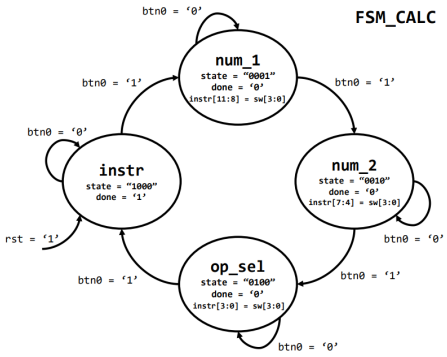

FIGURE 1: Máquina de estado de modo manual de la calculadora.

Para el modo Gato, se requiere indicar si un jugador ganó y quién fue ese jugador. Para ello, se ocupan los IP-Cores *DDS (Direct Digital Synthesizer) Compiler* [5] y *Adder/Substracter* [6], para emitir una señal oscilante distintiva para cada jugador cuando gane, inspirada en la implementación de Andrew Nguyen en Youtube [4]. El IP-

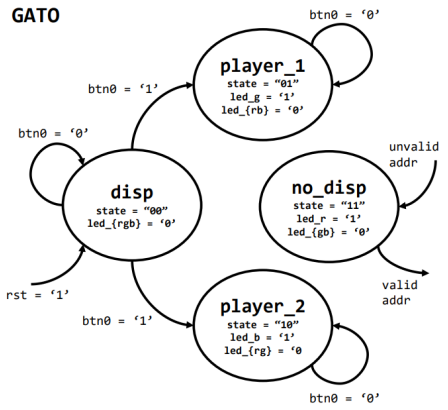

FIGURE 2: Máquina de estado de modo manual de la calculadora.

Core *DDS Compiler* sirve para la creación de sinusoides y el principal parámetro a configurar fue el incremento de fase $\Delta\theta$ (en binario) para obtener la frecuencia deseada, como se indica en la ecuación (1) (incluida en el *datasheet* [5]), donde f_{out} es la frecuencia deseada, $B_{\theta(n)}$ es el número de bits de la fase, que determina la resolución de la frecuencia de salida, y f_{clk}. La resolución de la frecuencia es un parámetro muy importante porque indica la frecuencia mínima de la señal que puede general el IP-Core, la cual disminuye (o bien, puede crear señales de un periodo más largo) al aumentar el número de bits de la fase. El IP-Core *Adder/Substracter*, por su parte, permite desplazar la sinusoide creada para poder modularla con PWM. Este efecto se resume en el diagrama 3 en el bloque **DDS_sin_PX**.

$$\Delta\theta = \frac{f_{out} \cdot 2^{B_{\theta(n)}}}{f_{clk}} \tag{1}$$

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): Fue posible el diseño e integración de tres *components* en una *entity*, siendo esta la **ALU** de la calculadora, dentro de la cual se encuentran un multiplexor, una unidad aritmética y una lógica. También, en el módulo **Gato** se llaman dos componentes, y en el modulo **PWM** otro. *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.

AO2 (100%): En la implementación del proyecto se utilizan ocho diferentes *packages*, sumados a otros seis *packages* pertenecientes a la librería de Vivado. Además, las *entities* empaquetadas en el proyecto Top y el package **PWM** tienen parámetros genéricos. Los primeros los utilizan para el largo de las entradas y la salida, aunque carece de sentido modificarlo por las limitaciones de la tarjeta para expresar resultados de más de cuatro bits. Mientras tanto, el segundo los utiliza para la cuenta máxima del divisor de reloj interno para crear la señal moduladora y el ancho del dato de entrada.

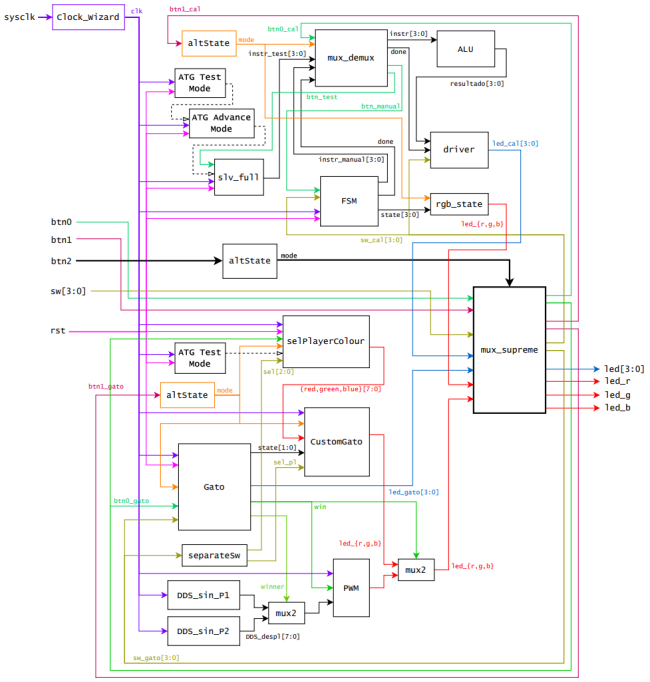

FIGURE 3: Diagrama de bloques del proyecto.

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.

AO3 (100%): Se utiliza un ATG maestro configurado en *Test Mode* para cargar las opciones de colores del bloque **CustomGato**, con una palabra de 24 bits (habilitados los 32 bits), y otro para configurar el ATG *Advance Mode* con una instrucción de escritura, una de lectura y una palabra de 32 bits, la cual escribe por medio de su puerto maestro a un periférico esclavo AXI4 *full* donde se escribe en una memoria BRAM, para posteriormente ser utilizado con lógica del usuario por la calculadora. *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.

AC1 (100%): Se generó una maquina de estados, **FSM_calculator**, para la construcción de la instrucción que se entrega a la *ALU* en la calculadora. Esta cuenta con cuatro estados, los primeros dos añaden a la instrucción los números en binario a operar, mientras que el tercero recibe el operador junto al selector (modo unidad aritmética o modo unidad lógica), y el cuarto entrega el resultado. Todos los valores ingresados son por medio de los *switches* y se alternan (confirman los valores) con el botón cero, K18. También es posible identificar una lógica de máquina de estados en el tablero de **gato**, identificando un estado inicial donde no se ha jugado la celda y otro en donde se jugó (ya sea por el jugador 1 o 2), además de un estado fijo que sin importar su entrada no podrá ser jugada (o bien, cambiar de ese estado). *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%): Se utilizaron los 4 botones, los 4 *switches* y los 4 leds para interactuar y mostrar datos relevantes. De acuerdo a la arquitectura presentada en la sección I, los botones se implementaron en los bloques **mux_supremo**, **mux_demux**, **gato**, **selectPlayerColour**, **altState**, etc., cuyas funcionalidades fueron ser *Reset*, para cambiar entre modos (calculadora o gato, y manual o AXI), y para confirmar entradas de números, operaciones, jugar la casilla y selección de colores. Los *switches* se implementaron en los bloques **FSM_calculator**, tanto para entradas de números de 4 bits y selector de operación en el modo Calculadora, como en el bloque **gato** para direcciones del tablero del minijuego y en **selectPlayerColour** y **CustomGato** como selector de color para un jugador en el modo Gato. Por otro lado, los leds se ocuparon como salidas: el led de 4 bits fue ocupado para mostrar tanto las entradas y las operaciones, como el número de casillas disponibles para jugar, mientras que el led RGB fue utilizado para mostrar el estado de la **FSM_calculator** e indicar la carga de datos por medio de AXI *full* en el modo Calculadora, y el estado de la casilla del tablero, el color seleccionado en el modo de personalización en **CustomGato** y quién ganó el juego en el bloque**PWM** del modo Gato. *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 (100%): Se utilizaron mínimo 5 operadores, entre lógicos, de asignación y aritméticos, y 5 atributos. Sobre los operadores, solamente en la **ALU** se utilizan al menos 16 operadores distintos entre lógicos, aritméticos y de asignación. Además, de acuerdo a la arquitectura presentada en la sección I, esto se implementó en diversos bloques del proyecto, en donde cuya funcionalidad fue principalmente poder parametrizar los módulos a través de los atributos **LENGTH** (largo en bits del dato, usado para funciones de conversión a std_logic_vector), **RANGE** (para ciclos *for* y crear señales con un mismo rango que otra), **LOW** y **LEFT** para acceder a índices fijos dentro de std_logic_vectors. También se utilizó **EVENT** en limitadas oportunidades, ya que su uso está vinculado al reloj/*clock* y es más simple ocupar la función *rising_edge*. *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.

AC4 (100%): Se utilizan *variables* porque a veces conviene actualizar una señal más de una vez dentro de un mismo *process*, además de ocupar ese valor actualizado para definir otras señales sin tener que esperar a que termine la rutina. De acuerdo a la arquitectura presentada en la sección I, nuevamente esto se implementó en diferentes bloques. Para este caso, su funcionalidad estuvo, por ejemplo, en el bloque **gato**, en un ciclo *for* para contar el número de casillas disponibles; o el indicador de turno en el *component* **casilla** del modo Gato. Sin embargo, cuando no es necesario actualizar en tiempo real un registro, es posible ahorrarse asignaciones y ocupar directamente *signals*. También, en menor medida, fue utilizado dentro de declaraciones *case* para generar una variable que solo existiría en esa instancia. *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.

AC5 (100%): En el código el código concurrente tiene la finalidad de crear conexiones directas entre señales para que retornen un valor, mientras que el código secuencial crea *flip-flops* para la asignaciones de señales (*signals* o *variables*). De acuerdo a la arquitectura presentada en la sección I, el código concurrente se implementó en los bloques **NOT_GATE**, **mux2** (multiplexor implementado en gato), **separate_Switches** y **led_driver**, cuya funcionalidad fue ahorrar tiempo, ya que los *flip-flops* agregan un *delay* para las señales. El código secuencial se ocupó en el resto de módulos, sin considerar *packages* propios de Vivado. Esto debido a que en general se necesitaba que los módulos conservaran la información de los *inputs* u *outputs* para tomar las siguientes decisiones, como en las máquinas de estado. *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.

AC6 (100%): Se hace uso de *functions* y *procedures*, principalmente para repetir código dentro de una entity. Por un lado, de acuerdo a la arquitectura presentada en la sección I, se utilizan *functions* casi siempre que se use un clock, con la función *rising_edge*. También se crea una función para determinar si un jugador ha ganado o no en el bloque **gato** (en el component **win_gato**). Por otro lado, se usa un *procedure* en el bloque **CustomGato**. Sus usos son consistentes con sus diferencias, ya que para *function* solo hay una salida y sus argumentos solo pueden ser señales *in* u *out*, por lo que es suficiente para usar sus resultados en condicionales, como se hace en el código. En cambio, un *procedure* sí permite otros tipos de argumentos, y más salidas, lo cual es útil para aumentar la cuenta del led RGB al definirlo como *inout* y la salida de cada componente del led. *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.

AC7: Para este proyecto, se utilizó un IP-Core no visto, cuyo objetivo en nuestro código es crear una señal distintiva para la victoria de un jugador u otro en el modo Gato. Se usa el IP-Core *DDS (Direct Digital Synthesizer) Compiler*, que permite la creación de sinusoides (senos y/o cosenos) y fases, ya sea en conjunto o por separado. Esta IP se configuró a través de parámetros de *hardware*, ingresando la cantidad de bits de la amplitud de la onda sinusoidal y de la fase, y el incremento de fase $\Delta\theta$ según la ecuación (1). En primera instancia, se deseaba utilizar este IP-Core para emitir un sonido distintivo (con armónicos) para cada jugador al configurarlo según una tabla de frecuencias, pero para comprobar que

efectivamente esté funcionando la modulación, se disminuyó la frecuencia (aumentando el largo de la fase) y se dejó un solo *DDS* para que sea visible en el led RGB como una luz blanca oscilante. Para eliminar las componentes negativas de la sinusoide, se instancia el IP-Core *Adder/Substracter*, que permite sumar o restar, con o sin signo, dos entradas o una entrada con una constante. En el diagrama 3 de la sección I se muestra su efecto como el bloque **DDS_sin_PX**. *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.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

- A través de las figuras 4 y 5 se puede comparar los resultados de una simulación de comportamiento y una simulación real, que replican el comportamiento de pasar directamente al modo Gato, cambiar los colores de los jugadores y la victoria del segundo jugador. Se puede identificar a simple vista de que los cambios en una simulación de comportamiento son más limpios y ocurren antes que en la simulación real, en especial para las salidas. En la figuras 6 y 7 se aprecia el momento en que al presionar el botón 0 se espera un cambio en los leds. Para la simulación de comportamiento (7), la demora entre la entrada y las salidas de los 4 leds y el led RGB es de 6.10 ns y 16.10 ns, respectivamente. Mientras, para la simulación real (6), la demora entre la entrada y las salidas de los 4 leds y el led RGB es de 16.38 ns y 27.87 ns, por lo que el delay de las salidas es de aproximadamente 10 ns. Además, en esta última, es posible observar de que la salida de los 4 leds se estabiliza en 1.55 ns, luego se pasar por 3 estados inesperados. Esto se puede estar produciendo por el cambio de cada bit del bus completo de los leds, cuyas rutas no necesariamente son las mismas dentro de la implementación del proyecto (es decir, no cambian al mismo tiempo), ni siguen una proporción fija al observar que estas salidas intermedias tampoco se mantienen por el mismo tiempo, aunque este es mucho menor a un ciclo de *clk*. En ambos casos, la luz del led RGB demora más que la salida de los 4 leds, porque la entrada debe "pasar" por más bloques para llegar la salida del led RGB que para los 4 leds.

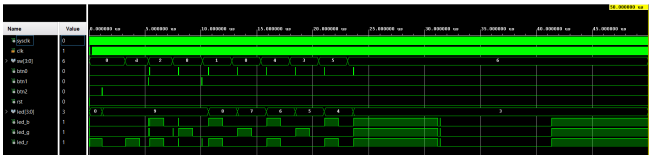

FIGURE 4: Simulación real para el modo Gato.

También se puede acotar que los primeros *testbenches* en la simulación temporal no funcionaban correctamente, debido a que el *sysclk* se asignó erróneamente a 1 ns, cuando en realidad su periodo es de 8 ns. Además, las entradas se asignaban en menos de 20 ns. En suma, el retraso de las señales, sumado a la frecuencia de reloj aún más alta que la realidad, provocaba asignaciones erróneas, como no alternar entre jugadores, realizar el *duty cycle* especificado, etc. Por ello, se aumentó el tiempo entre interacciones y se obtuvieron resultados que se corresponden a lo observado posteriormente al probarla en la ZYBOZ7.

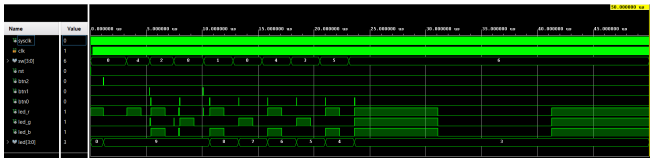

FIGURE 5: Simulación de comportamiento para el modo Gato.

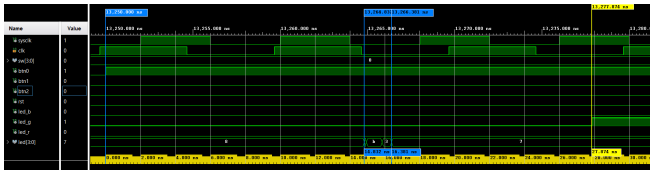

FIGURE 6: Detalle de simulación real para el modo Gato.

- Se aisla la entity **gato**, en donde está el component **win_gato** al cual se le pueden asignar valores mediante variables o señales, ya que al ejecutar la simulación post-implementación del *wrapper*, se borraron algunas de las señales y variables declaradas en ella y fueron reemplazadas por otras. En la figura 8, se observa que la diferencia de actualización entre una señal y una variable, que para esta entity (y component) es de 145 ps.
- En las figuras 9 a 12 se ven los handshake de las transacciones de escrituras hechas para configurar la información de los coe, 10 mensajes vistos desde la perspectiva del *Test Mode*, en el esclavo *Advance Mode*. Vale destacar que se puede observar en las 10 y 11 como el *AXI Protocol Converter* actúa a modo de puente.
  Luego, en las figuras 13 y 14 se aprecia el final de la transmisión de datos en AXI-Full. Note que aquí si importó observar el handshake de escritura, puesto que en el proceso el dato se escribe y se lee para habilitar su posterior recuperación. Además, al ser el envío de un solo, a diferencia de las partes anteriores del proceso, este tiene solo una seguidilla de *flags* para escritura y lectura.
  De modo similar a la calculadora, en el gato se implementó la transferencia de información con un ATG *Test Mode* como maestro. Los handshake, figuras 15 y 16 son más claros debido a que el *waveform* de Vivado ofrece indicadores propios.

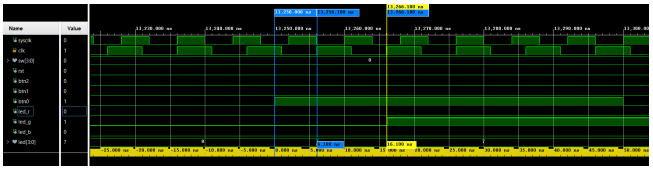

FIGURE 7: Detalle de simulación de comportamiento para el modo Gato.

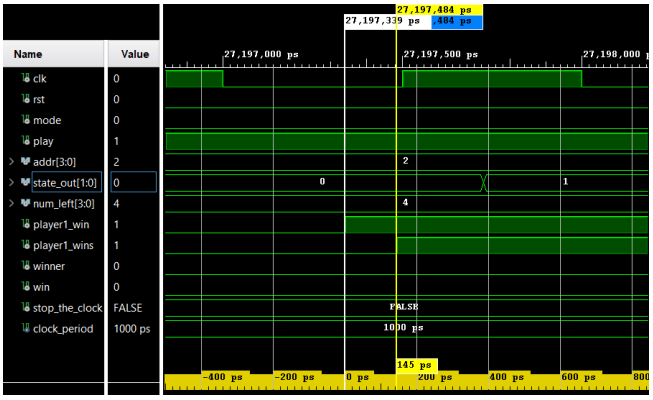

FIGURE 8: Detalle de simulación real en el modo Gato, que compara una *signal* (*player1_win*) y una variable (*player1_wins*).

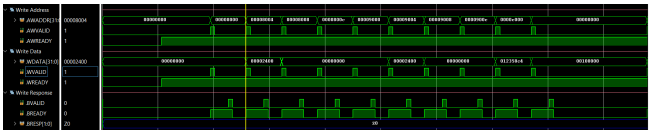

FIGURE 9: Handshakes de ATG Test Mode puerto maestro en la calculadora

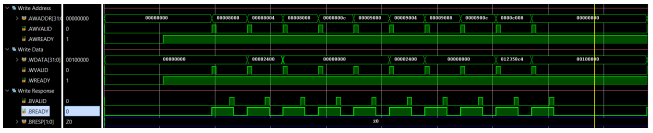

FIGURE 10: Handshakes de ATG Protocol Converter puerto esclavo en la calculadora

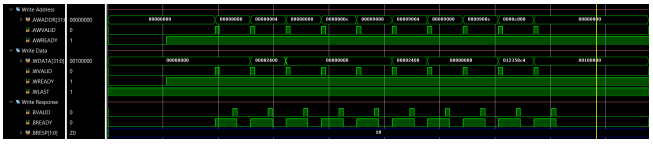

FIGURE 11: Handshakes de ATG Protocol Converter puerto maestro en la calculadora

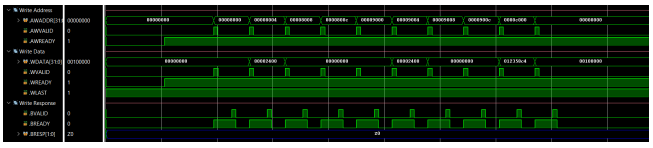

FIGURE 12: Handshakes de ATG Advance Mode puerto esclavo en la calculadora.

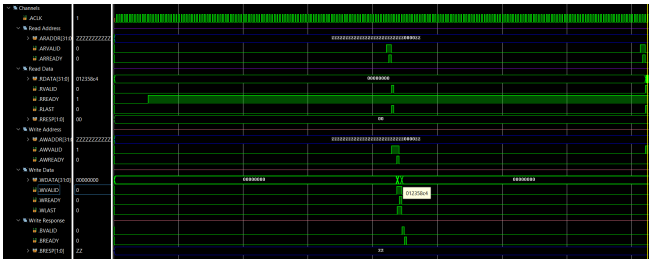

FIGURE 13: Handshakes de ATG Advance Mode puerto maestro en la calculadora.

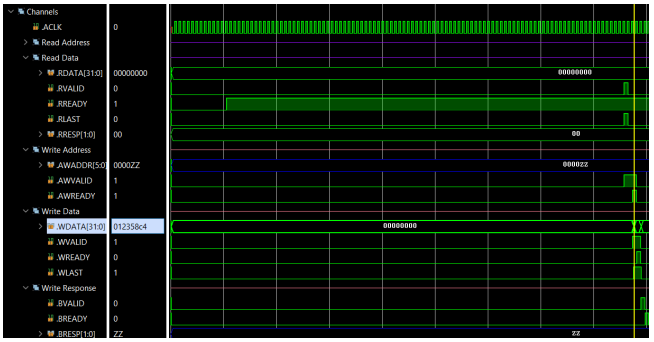

FIGURE 14: Handshakes de periférico esclavo AXI-Full en la calculadora.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

A pesar de que la implementación fue completada sin errores que impidan la creación del *bitstream*, el reporte de tiempo falló porque los retardos de las señales son mayores a los que define Vivado como razonables para el *clock* maestro (125MHz) y el *clock* distribuido a los bloques (100MHz). Esto resulta en que la interacción con la ZYBOZ7 deba ser, por precaución, un poco más pausada para evitar errores. Las rutas que presentan este problema están relacionados al juego del gato: el conteo de celdas disponibles para jugar (loop con if-else), y el conteo de los led RGB (*procedure* con *inout*, actualizado en cada flanco de subida de reloj), ambos descritos en código secuencial y condicionales if-else. Este retardo provocado por condicionales genera, por lo tanto, que el tiempo de estabilidad entre los flancos de subida que se debe respetar no se logre.

Particularmente, al implementar el proyecto, el fallo en el cumplimiento de tiempo no se traduce en errores significativos o apreciables. Hay que considerar que incluso el *setup time* de la señal más lenta es de menos de 8 ns, por lo que, a menos que se logré introducir una señal nueva al botón cero más rápido que eso, no habría problemas.

V. CONCLUSIONES(0.2)

En conclusion, este proyecto:

- Incumple los requisitos de tiempo recomendados por Vivado. Se obtuvieron WNS de -7.429 ns, TNS -4.7 μs aproximadamente, además, se sabe que de todos los *endpoints* el 4.97% fallan el requisito de tiempo en

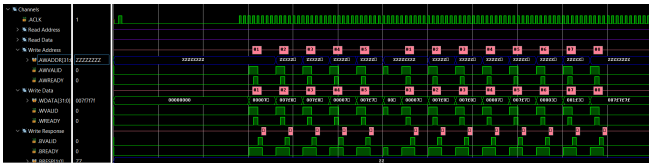

FIGURE 15: Handshakes de ATG Test Mode puerto maestro en el gato.

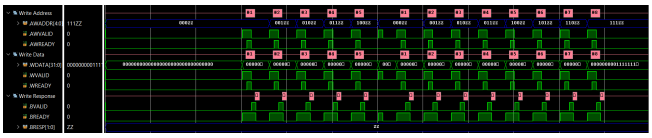

FIGURE 16: Handshakes de periférico esclavo AXI-Lite en el gato.

setup. No obstante, ninguna de las rutas falla en llegar a destino y se cumplen los requisitos de *hold slack*, WHS 0.020 ns, y *pulse width slack*, WPWS 2 ns.

- En un *process*, para una entity del proyecto, la asignación mediante señales se demora 145 ps más que al hacer la misma asignación con variables.
- Se cumplieron al 100% tanto las actividades obligatorias como las complementarias. Destacar el éxito de todas las transferencias de datos por medio de los protocolos AXI *full* y AXI *lite* haciendo uso de módulos ATG en modo *Test* y *Advance*. Esto se pudo comprobar en las implementaciones finales a la tarjeta ZYBOZ7.
- Todas las transacciones necesarias para llevar a cabo la transmisión AXI en Full y Lite tomaron 1.65 μs y 0.76 μs respectivamente. Considerando que, en las implementaciones de Full y Lite, las diferencias son la necesidad de un módulo conversor de protocolo y que, intrínsecamente, para AXI Full se necesita un módulo extra que programe al maestro, entonces se puede inferir que el tiempo extra requerido tiene causa en estas distinciones.
- Sobre las simulaciones, al comparar las hechas para comportamiento y las reales, fue posible identificar que el proceso real era alrededor de 10 u 11 ns más lento cuando se revisaba la salida de los leds para alguna entrada. Además, a partir de la simulación real se observó que los 4 leds necesitan 1.55 ns para estabilizar su valor. Se atribuye esto último a que los cambios en cada bit del bus no tienen porque tener la misma ruta, por lo que no cambian a la vez y se generan los estados transitorios.
- Por último, por medio de la simulación real del modulo **gato**, particulamente su componente **win_gato**, fue posible notar que la diferencia entre el tiempo de actualización de la señal y la variable *board* es de aproximadamente 145 ps.

VI. TRABAJOS FUTUROS (0.2)

Tal como se analizó en la sección IV, el principal desafío es optimizar las capas de código lógico e instanciaciones del proyecto para lograr cumplir con los requerimientos temporales según analiza Vivado. Los datos actuales que no cumplen se encuentran en la sección de *setup*, es decir, tiempo mínimo de estabilidad entre flancos de subida.

También se podría pulir la comunicación del ATG *Advance Mode* para adquirir la capacidad de escribir y leer más de un dato dentro de un modulo periférico. Esto implica por supuesto el envío de *burst* de datos con largo mayor a uno, su almacenamiento y recuperación completa e integra. Por último, podria ser interesante la programación del modulo ATG *Advance Mode* como maestro por medio de un periférico con puerto maestro en vez de depender de un ATG en *Test Mode*, de modo que no se necesite un convertidor de protocolo.

Dentro del mismo foco de pulir el ATG, podría ser interesante mejorar el tiempo de inicialización que necesita el sistema. Durante las simulación fue posible notar que los procesos de *Advance* y *Test Mode* necesitan 1.65 μs y 0.76 μs aproximadamente para realizar la transferencia completa de datos.

REFERENCES

[1] Anónimo. [boysandcakes]. (17 de febrero, 2015). *Casio MG-880 Calculator with Melody and "Invaders" Game* [Video]. YouTube. https://www.youtube.com/watch?v=BP3snsh7D5U

[2] Anónimo. [PRO FILMS99]. (23 de julio, 2019). *How To Play TIC TAC TOE On Calculator* [Video]. YouTube. https://www.youtube.com/watch?v=AUWmXyXGVWg&ab_channel=PRO-FILMS99

[3] OpenAI. [ChatGPT]. (s.f.) *Quiero que escribas un código simple para que alterne la entrada entre std_logic = '0' y '1' según la entrada de un botón. Es decir, la salida inicia en '0' y apreto el botón, por lo que cambia a '1' y se mantiene en '1' hasta que presiono de nuevo el botón y la salida vuelve a '0'.*

[4] Nguyen, A. [andrew nguyen]. (25 de noviembre, 2015). *EE432: Lab 5 - Vivado DDS Audio and BRAM Audio* [Video]. YouTube. https://www.youtube.com/watch?v=pxIbiUlpGck&ab_channel=andrewnguyen

[5] Xilinx. (2011). *LogiCORE IP DDS Compiler v5.0*. https://docs.xilinx.com/v/u/en-US/ds794_dds_compiler

[6] Xilinx. (2011). *LogiCORE IP Adder/Subtracter v11.0*. https://docs.xilinx.com/v/u/en-US/addsub_ds214

[7] Garcés Cristian. (2023). *Experiencia 2 - Sistema TxRx Infrarroja. IEE2473*. [PDF]