Fecha de entrega 11 de mayo de 2024.

Operador de Funciones

PATRICIO HINOSTROZA[1], VICENTE NAVARRETE[1]
[1]Pontificia Universidad Católica de Chile (e-mail: patricio.hinostrozav@uc.cl, vnavarretea@uc.cl)

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

ABSTRACT Se implementó un programa en la tarjeta ZYBO Z7-10 capaz de generar 4 tipos de funciones diferentes y operar matemáticamente con ellas, entregando el resultado de operaciones como suma y multiplicación de señales. Para ello se utilizaron distintas herramientas de vivado y lenguaje de programación VHDL. Se implementaron distintos IPCores creados y disponibles en vivado, y se utilizó protocolo de comunicación AXI para interconectar algunos de ellos.

INDEX TERMS VHDL, AXI4, Zybo Z7-10, Vivado block design, Generador de funciones, TestBench.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

El objetivo de este proyecto es funcionar como generador y operador de funciones, por lo que para su implementación la arquitectura de Hardware se planteó como sigue. Para una información ver figura 1.

Lo primero es escoger la frecuencia con la que se van a generar las frecuencias, para esto se utiliza un bloque divisor de clock y un dato escalar envíado por protocolo Axi Advance, el cuál tiene por función multiplicar el divisor para así aumentar o reducir la frecuencia del clock de salida. Esta frecuencia sirve después, para generar un contador de 8 bits. Para luego, con este contador, que va de forma ascendente, generar 4 funciones principales, una onda sinusoidal, una cuadrada, una triangular y una dientes de sierra, todas ellas de 8 bits.

Cabe destacar que la función sinusoidal se genera ya que el contador sirve como "address" para la lectura de una memoria que contiene los datos de la onda, los cuales son subidos mediante protocolo Axi-lite en configuración Test Mode. Además, mediante la instrucción de un bloque VIO, se le puede ajustar la referencia del comparador que genera la onda cuadrada, para cambiar su ciclo de trabajo. Por último, para las funciones dientes de sierra y triangular, solamente se lee el contador ascendente, y el contador ascendente y luego descendente, respectivamente.

Una vez generadas las funciones, se ordenan los bloques encargados de la operación entre funciones. El primero se encarga de recibir inputs a través de los botones de la tarjeta, para administrar las funciones a su salida. Éste por defecto no saca nada, pero en cuanto se presiona algún botón, saca su función respectiva en la salida "función a". Cuando se aprieta un segundo botón, si es que no es la misma función anterior, esta se copia en la salida "función b". Si luego de esto se presiona otro botón, se reemplaza la función más antigua por la nueva, y así sucesivamente. Existe la posibilidad de reiniciar las salidas con el bloque VIO.

Por último se encuentra el bloque "math", el cual se encarga de realizar las operaciones entre las funciones a y b seleccionadas anteriormente. Las operaciones se seleccionan mediante los switches de la tarjeta, y siguen un orden lógico según el orden de los bits. El bit de menos significativo representa la presencia de la función a, el siguiente la presencia de la función b, el siguiente la presencia de una constante c, la cual entrega el bloque VIO, y el bit más significativo alterna entre la función suma o multiplicación. Las instrucciones se resumen en la tabla 1. La salida es el resultado de la operación expresada en 16 bits.

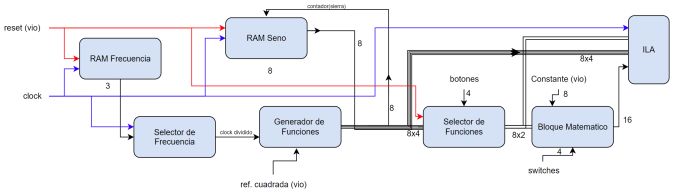

FIGURE 1: Esquema de Proyecto

Instrucción	Salida
0000	Apagado (0 constante)
0001	a
0010	b
0011	a + b
0100	c
0101	a + c
0110	b + c
0111	a + b + c
1000	Apagado (0 constante)
1001	a
1010	b
1011	a · b
1100	c · (a + b)
1101	c · a
1110	c · b
1111	c · a · b

TABLE 1: Tabla de instrucciones dadas por switches

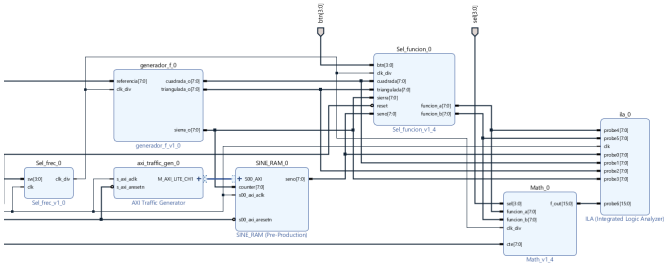

FIGURE 2: Diagrama de bloques (Vivado)

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

Se presentan las actividades implementadas y sus especificaciones.

AO1 (100%): *Descripción:* Fuera del código de wrapper del block design del proyecto final, se utilizaron 3 components dentro de la entity para el generador de funciones. Este bloque, cumple con lo explicado en la sección I. Es decir, recibe un señal clock con la cuál genera un contador que a su vez es la salida dientes de sierra (primer component), el cuál se compara con una referencia generando una señal cuadrada (segundo component) y también se cuenta de manera ascendente y descendente, con lo que se genera la señal triangular (tercer component).
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%): *Descripción:* A lo largo de la implementación del proyecto, se trabajó con muchos IPCores creados e implementados, se pueden observar en la figura 2. En varios de éstos, se utilizaron parámetros genéricos como por ejemplo en el caso del bloque "math". En el cual se utilizan para el manejo del largo de bits de las entradas, salidas y funciones internas.
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%): *Descripción:*Se implementó y testeó Axi-Full Advance Mode y Axi- Lite Test Mode, para ello se crearon los archivos COE necesarios para ambos, ya que el bloque Axi-Full fue programado utilizando un bloque extra de Axi-Lite en Test Mode, en el primero se escribió una rutina simple que se basa en publicar a la dirección xC00 el numero 1, el cual es recibido por una RAM al recibir las condiciones de AWVALID y AWREADY para guardarlo y luego publicarlo, por otra parte, el bloque Axi-Lite se conectó a una RAM que gracias a un contador entrega el numero correspondiente al registro como salida, creando así una función seno.
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%): *Descripción:* Se implementaron varias máquinas de estados en los distintos IPCores, del proyecto. Un ejemplo es el caso del bloque "Selector de funciones", el cuál implementa la máquina de estados cuyo esquema se aprecia en la figura 3. Sus estados se resumen en: no entrega ninguna función, entrega una sola función y entrega dos funciones a la salida. El cambio entre estos es mediando los botones de la tarjeta y la instrucción reset que entrega el bloque VIO, tal como se explicó en la sección I.
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.

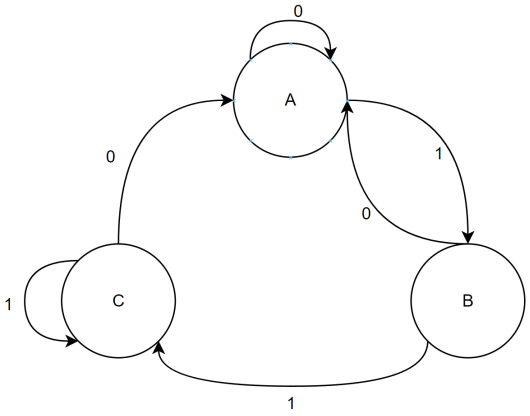

FIGURE 3: Diagrama de Máquina de Estados

Descripción: Los estados son A: ninguna función en la salida, B: 1 sola función a la salida, C: 2 funciones a la salida. La entrada 0 significa que se apretó el botón reset y la entrada 1 significa que se apretó alguno de los botones que representan una función.

AC2 (100%):*Descripción:* Se utilizaron los bloques VIO e ILA a lo largo del proyecto. Como se explicaba en la sección I, el bloque VIO genera varios output, los cuales utilizan otros bloques, tales como: la referencia como porcentaje de ciclo de trabajo de la onda cuadrada, la señal reset del bloque selector de funciones y la constante que utiliza el bloque math. Por otra parte, el bloque ILA se utiliza para medir las salidas y señales de interés, lo cuál es esencial para el objetivo de generador y operador de funciones. Concretamente, se leen las señales de las 4 funciones generadas originalmente, las funciones seleccionadas a y b, el valor de la constante enviada y la salida final del sistema.
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%): *Descripción:* A lo largo del proyecto se usaron varios tipos de atributos y operadores. Ejemplos de estos se pueden encontrar en el bloque math, en el cual, dentro de las definiciones de funciones que posee, se utilizan operadores para sumar y multiplicar las funciones recibidas. Dentro de este mismo, se pueden encontrar atributos como "Event" y "Length" usados para el manejo de la Arquitecture.
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%): *Descripción:* Se utilizaron variables en distintos casos necesarios, o para optimizar código. Dentro de los IPCores se pueden encontrar tanto señales como variables dentro de la Arquitecture. Un ejemplo podría ser el bloque "math", en el cuál se usan variables dentro de las functions, y luego una señal para manejar la salida.
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%): *Descripción:* En diferentes IPCores se utilizó tanto código secuencial como código concurrente. Ejemplos de código secuencial son que los bloques se coordinan mediante el mismo clock o la máquina de estados implementada en el bloque "selector de funciones". Mientras que un ejemplo de código concurrente se puede encontrar en el bloque "math" el cuál realiza una operación según las instrucciones de los switches.
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 (50%): *Descripción:* Se utilizaron functions en varias partes del proyecto, un ejemplo es dentro del bloque "math", en el cuál se definieron las funciones para la suma y multiplicaciones de vectores logicos de 8 bits, que entregan resultados de 16 bits.
Nivel de Logro: 50% El código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7. Pero solo se utilizaron functions y no procedures.

AC7 (100%): *Descripción:* Se implementó el bloque AXI SmartConnect, el cuál en clases nunca se usó, ya que para menor complejidad se eliminaba del block design cuando aparecia por la función "auto-connect". En esta ocasión se utilizó para optimizar las comunicaciones por AXI 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.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

A continuación se hace la comparación de la simulación del bloque generador de funciones con la simulación post-implementation del mismo.
Esta es la simulación de comportamiento:

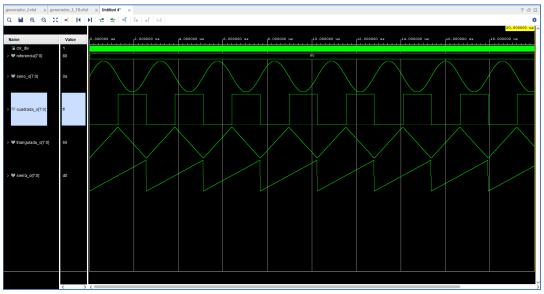

FIGURE 4: Simulación de comportamiento bloque Generador de Funciones

Y esta la simulación post-implementation:

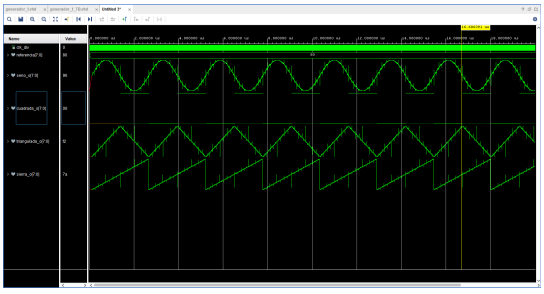

FIGURE 5: Simulación post-implementation bloque Generador de Funciones

Se puede notar que las funciones parecen ser más "ruidosas" en la segunda imagen. Además, la señal menos afectada es la cuadrada. Se puede notar además, que los "errores" que se producen o muestran en la onda dientes de sierra, se reflejan en las demás, esto es porque esta onda es la base de las otras. Estas "interferencias" se deben principalmente a 2 razones. La primera tiene que ver con problemas de sincronización con el clock original, ya que se recibe en varios bloques a la vez y también puede estar expuesto al efecto de un jitter. La segunda tiene que ver con el uso de "signals" y "variables", ya que unas se actualizan más rápido que las otras y las distintas funciones se generan con el uso de unas u otras. Por ejemplo, la señal triangular solo usa "signals" y se puede notar como tiene más imperfecciones.

Por otra parte, las visualizaciones utilizando ILA con AXI-Full y AXi-Lite fueron exitosas, se ve que se publica lo solicitado por el driver y archivos coe de forma correcta, al levantar bien las flags correspondientes en cada transacción.

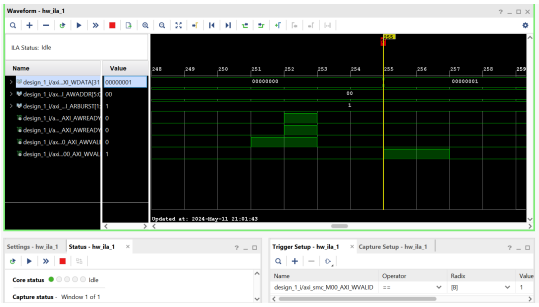

FIGURE 6: ILA para Axi-full advance mode

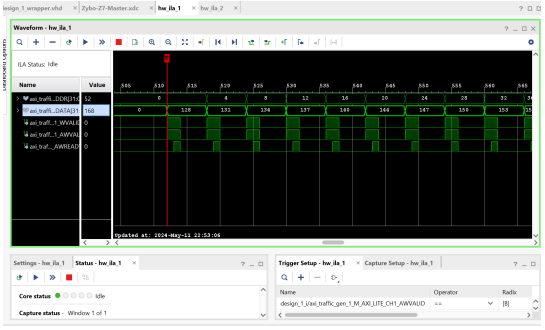

FIGURE 7: ILA para Axi-full test mode

En las imágenes previas se muestra el funcionamiento y lectura mediante el bloque ILA de lo relacionado a protocolo de comunicación AXI. Se aprecia que funcionaron cada uno de manera independiente.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Al momento de la implementación se separó el proyecto en 2 partes para finalmente unirlo todo. La primera parte consistió en todo lo que no incluía AXI, para lo cuál no se tuvieron grandes problemas. Podrían mencionarse como obstáculos el manejo de bits para evitar overflow y la sincronización de clocks, los cuales se ahondarán en las conclusiones.

La segunda parte se relaciona al testeo de lo relacionado con AXI lo cuál también funcionó en su totalidad tanto en configuración Test-Mode como en Advance-Mode. Finalmente, se implementó todo en el proyecto en su totalidad, y aunque la simulación fue posible, no se logró reproducir con la tarjeta, el error posiblemente es de sincronización de AXI.

V. CONCLUSIONES(0.2)

- Los resultados vistos en la ILA fueron bastante similares a las simulaciones e incluso mejores que las simulaciones post-implementation, ya que no se observó ruido u errores de coordinación entre ciclos. Un problema habitual al momento de ver los resultados se encontró en la onda cuadrada y se debía a la forma de registrar datos de manera analógica, ya que interpolaba datos y en lugar

de una cuadrada se observaba una dientes de sierra. A pesar de esto, se puede notar que es una cuadrada ya que sus datos son 0 y 255, que corresponden al máximo y al mínimo.

- Al momento de operar con las funciones en el bloque "math", idealmente se querían salidas de la misma cantidad de bits que la entrada. Pero, las funciones originales abarcaban rangos en que llegaban a valores máximos, por lo que se necesitaría al menos un bit más para la suma, y al menos el doble para la multiplicación. Es por esto que la salida es de 16 bits, aunque se dejó el caso de overflow cuando se añade también la constante a la suma o multiplicación de 2 funciones, para mostrar el efecto que tiene.
- respecto a AXI-Full se puede concluir que se logró comunicar en un entorno de prueba lo solicitado por el driver, que consistió en 1 instrucción de escritura, la cual se demoró 2 ciclos de clocks en enviarse y recibirse.
- respecto a AXi-Lite se testearon sus capacidades al enviar 256 instrucciones de escritura, de las cuales todas fueron exitosas y no se perdió ninguna información, cada una se demoró 6 muestras de la ILA en enviarse y recibirse.

VI. TRABAJOS FUTUROS (0.2)

Se plantea que el programa generado puede ser mejorado al implementar otro tipo de procesamientos a las señales creadas, tales como puede ser desfase, variación de frecuencia por señal, resta, etc. otra posibilidad de mejora viene en la utilización de periféricos integrados en la placa para la visualización de las diferentes ondas u aumento en la cantidad de frecuencias diferentes a almacenrar con la Ram de axi full.

REFERENCES

[1] Rojas, " IEE2463 Sistemas Electrónicos Programables," GitHub, 2023. https://github.com/IEE2463-SEP
[2] Xilinx, inc., "AMD Technical Information Portal," 2023. https://docs.amd.com