Fecha de entrega XX de YYYY de 20CC.

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

Ledmáforo: Simulador de Autos frente a un Semáforo en VHDL para ZYBO Z7-10

MANUEL ALVAREZ COFRÉ[1], JUAN IGNACIO VARGAS FERNÁNDEZ[1]
[1]Pontificia Universidad Católica de Chile (e-mail: manalvarezc@uc., jvargasf@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 simula, de forma simple, el comportamiento de una calle con autos frente a un semáforo. Los autos son representados por los leds de los switches, el semáforo por el led RGB. Mediante los switches se elige la velocidad de los autos. Para hacerlo se utilizó una placa ZYBO Z7-10, especifica para programación de Hardware, la cual fue programada en su Programmable Logic (PL) mediante el software Vivado de Xilinx, ocupando el lenguaje VHDL. Se utilizó también el protocolo de comunicación AXI, en su versiones Lite y Full, para la comunicación entre ciertos bloques que forman este hardware, principalmente para guardar datos en ciertas memorias RAM de estos bloques y despues para leerlos y confirmar su correcta carga. Esta información se utiliza para guardar datos tanto de las velocidades que los autos tienen para andar, como la duración del semaforo cuando está configurado para que cambie automáticamente.
La simulacion behavioural muestra un correcto funcionamiento para todos los bloques pero, al implementarlo en la tarjeta Zybo, funciona correctamente todo excepto por el modo automático.

INDEX TERMS VHDL, maquina de estados, memoria, AXI Lite, AXI Full, ATG, packages, simulador, Zybo-z 10, IEEE

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

El proyecto consta de siete bloque principales. Los tres bloques principales de elaboración propia y específica para el proyecto son : **Translator**, **RAM Velocidades**, **Semaforo** y **LED Driver**, además de tres ATG; dos en Test Mode y uno en Advanced. La conectividad de los bloques se puede ver en el diagrama de bloques de la figura 1.

El ATG en test mode **AXI Lite 1** inicializa al ATG **AXI Full** en advanced mode, que a su vez guarda las distintas velocidades disponibles en **RAM Velocidades**. Esto lo hace mediante la carga de archivos .coe en el **AXI Lite 1**. Estos poseen las instrucciones para configurar al **AXI Full**. Este **AXI Full** también tiene un comando de lectura mediante axi de la RAM simplemente para que, al simular, se verificara la correcta carga de los datos en la **RAM Velocidades**. Este bloque, si el switch de entrada enable está activado, selecciona una de estas velocidades mediante los switches, traducidos a una dirección de la **RAM Velocidades** por el bloque translator, y la entrega a **LED Driver**.

El ATG **AXI Lite 2** inicializa la RAM de **Semáforo** con los tiempos activos de la luz roja o verde del semáforo para cuando este esté en modo automático.

Semáforo, por su parte, utiliza estos valores para determinar el tiempo de encendido y apagado de la luz roja y verde de estar en modo automático. Este modo se seleccionado mediante la entrada del botón auto toggle, que cambia entre los modos cada vez que es presionado. De estar en modo manual, el color de la luz es seleccionado con los botones green y red. Este bloque entrega a **LED Driver** el estado actual del semáforo.

LED Driver, que maneja la salida final, representa en los LED de la tarjeta a los autos avanzando según la velocidad recibida y traducida interiormente a un período que divide el clock de la Zybo. Muestra también el estado del semáforo, que es evidenciado en el color del LED RGB. Los autos avanzan mientras la luz se encuentre en verde y se detienen al estar esta en rojo.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

Todas las actividades logradas pudieron ser correctamente sintetizadas, simuladas y cargadas a la tarjeta mediante el bitstream. La única actividad que no fue lograda en su totalidad fue la AC2.

AO1 (100%): Se crearon 3 *components*, cada uno en un

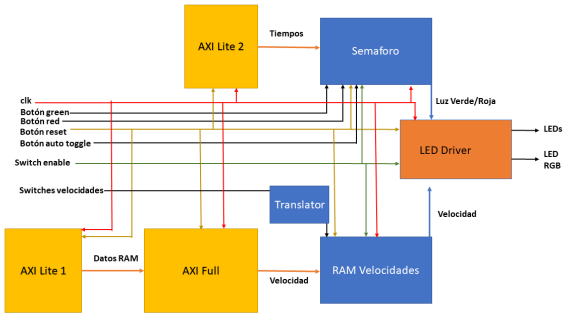

FIGURE 1: Diagrama de bloques del proyecto

package, que son integrados en el bloque de **LED_Driver**. Estos componentes, Multiplexor, cars_sim y vel_translator manejan la lógica de **LED_Driver**. El funcionamiento de cada componente se explica en el siguiente punto.

AO2 (100%): Se generaron 3 packages. El primero, pkg_utils contiene el multiplexor que selecciona el valor de GR y a la *function* vec2int, que convierte un valor de std_logic_vector a integer. El segundo package pkg_vel, contiene vel_translator, que traduce el valor decibido de velocidad a valor de período para el divisor de reloj que determina la velocidad de cambio de los LEDs que representan el auto. Final,ente, pkg_car contiene cars_sim, que consiste de una máquina de estados que guarda la posición del auto, para luego ser mostrada en los LEDs. Los packages son usados en **LED_Driver**.

AO3 (100%): Se utilizó un ATG maestro en Test Mode para entregar los valores del divisor de clock del bloque **Semaforo**. También se utilizó ottro ATG maestro en test mode para configurar y darle instrucciones a un ATG en modo Advanced. Este ATG en Advanced Mode se utilizó como maestro de un periférico AXI Full para cargar en este datos de las distintas velocidades posibles de los autos (LEDs), para que luego en este los switches, tras traducirse, seleccionen una ubicación de la memoria **RAM Velocidades** y esta entregue el valor de la velocidad al bloque **LED_Driver**.

AC1 (100%): Se Generaron dos máquinas de estado. Una en cars_sim que guarda y entrega la posición de los autos, representada con que LED se encuentra prendido. La máquina rota entre los estados "0001", "0010", "0100" y "1000" de tal forma que se evidencie un traslado del LED encendido. La otra máquina, de dos estados, se encuentra en **Semáforo**. Esta rota entre los estados que representan el modo automático y manual.

AC2 (90%): Se utilizaron los 4 botones, 4 switches y 4 leds de la ZYBOZ7 en la planificación y escritura del hardware. Los switches se usaron para controlar la velocidad de los auto y los botones para controlar el semáforo. Los leds vistos en el curso se usaron para representar los autos. Si bien todos los botones tienen un comportamiento asignado, el que cambia el modo automático a manual no funciona correctamente en al implementarlo en la tarjeta. Sin embargo, en las simulaciones "behavioral" si funcionaba correctamente, por lo que se puede decir que estan todos los botones, leds y switches utilizados, pero que no todos se pudieron aplicar en la tarjeta.

AC3 (100%): A lo largo del proyecto, se utilizan más de 5 operadores diferentes, entre ellos: /, +, :=, <=, =, * y **. Se utilizan 5 atributos diferentes: *length*, *high* y *low* son utilizados en la *function* para convertir un vector a escalar mientras que *event* y *stable* son utilizados como alternativa para detectar el *risingedge* del clock en cars_sim y multiplex, respectivamente.

AC4 (100%): Se utilizan variables para actualizar valores instantaneamente en los módulos vel_translator y pkg_utils. En vel_translator, la variable t se utilizó como auxiliar para guardar el valor que se entregará como período mientras que la variable mega tiene el propósito de cambiar entre modo "testing" y normal. El modo testing disminuye considerablemente el período para hacer visibles los cambios al momento de simular el protecto. Por otro lado, en pkg_utils se utiliza la variable result para guardar el resultado de la conversión entre vector y escalar.

AC5 (100%): Se da uso a código sequencial, escrito dentro de processes a lo largo del proyecto, por ejemplo, en los módulos vel_translator y cars. Se utiliza también código concurrente tanto en el **Semaforo**, donde el calor de GR se acctualiza independientemente de los process haciendo uso de WHEN. También existe código secuencial y concurrente en los código pre hechos de Vivado cuando se crean los ATG.

AC6 (100%): Se hace uso de una *function* y un *procedure*. La *function*, vec2int, traduce el vector de input de **LED_Driver** a un escalar, que se entrega luego a vel_translator para seleccionar el período. El *procedure*, update_output, se encuentra en el módulo multiplex y actualiza los valores de los leds en base a la entrada GR. Se diferencia de una *function* porque puede entregar varios outputs al mismo tiempo, que en este caso son los valores de led_g y led_r.

AC7 (100%): Como función no vista en clases, se implementó el uso del LED RGB de la Zybo Z7-10. Este representa al semáforo en el simulador y se aplicó con colores verde y rojo.

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

Si bien la simulacion Behavioral es una útil herramienta de debugueo, esta asume componetes y comportamientos ideales. En la simulación post implementación es posible observar comportamientos, interacciones y delays más cercanos al esperable en la realidad. A continuación se presenta un análisis de lo observable.

- En la figura 2. se puede observar un delay de al rededor de 16 ns entre la acción de presionar el boton green a la activación del led verde y desactivación del rojo. En la simulacion, esto corresponde a 4 ciclos de reloj.
- Si bien, las variables no son directamente visibles en las simulaciones que realiza Vivado, estas debiesen no mostrar el delay relacionado a las señales. Las señales,

sin embargo, son fáciles de observar y se evidencia un delay, somo se expicó en le punto anterior.

- Comenzando con la transacción desde el maestro ATG Test Mode hacia el esclavo ATG Advanced, se ve como en la figura 4 las distintas señales que representan los *handshakes* de AXI se van activando. Se ve en un comienzo como *awready* se mantiene en valor 1 y el *awvalid* va intermitentemente activándose y desactivándose a medida que los distintos datos se escribían en el ATG Advenced. Se puede ver como la señal de direcciones *waddr* también va cambiando en cada una de las escrituras ya que, al estar en modo Test, solo se puede cargar un dato a la vez, a diferencia del Advanced donde se pueden enviar ráfagas de datos (también implementado en este proyecto). Ests direcciones representan en un comienzo, escritura en la CMDRAM en las direcciones 0x8000 para lectura y 0x9000 para escritura, para despues llegar a las 0xC000 donde ya se está escribiendo los datos en la MSTRAM. Al final, los únicos datos que pasan del ATG Test hacia el periférico RAM son estos últimos escritos en la MSTRAM, que en este caso representan las velocidades.
  Vale destacar como también se pueden identificar el correcto funcionamiento de las señales de lectura que hace el ATG Test del ATG Full. Si bien estas no tienen relación con la funcionalidad del proyecto, si que nos permiten confirmar la correcta implementación AXI Test. Para AXI Full
  Como se puede ver en la figura 3, se realiza una correcta transacción de datos mediante AXI Full. Se puede ver cómo inicialmente *awvalid* y *awready* se activan para comenzar la transacción, luego cambian las distintas señales de wvalid y wready, además de mostrarse el largo de la ráfaga (03 que representa largo 4), para finalmente, al terminar la escritura, se active el *wlast*, la cual es característica del modo Advanced para señalar el fin de la ráfaga, en este caso de 4 datos, y el *bready* de axi, indicando que se terminó la escritura. Los distintos datos escritos también se muestran, y vale destacar como por la implementación real, estos no tienen el mismo tiempo de carga (el dato 00000001 tarda más que el 00000002), lo cual puede deberse a un retraso de la actualización de la siguiente señal en un ciclo del clock. Además, en laa figura 5 se evidencia el funcimaniento del AXI Lite 2, donde las señales *awvalid* y *wvalid* se activan en un comienzo al partir la escritura, para después *awready* y *wready* se vean activadas. Mostrar la correcta ejecución de las transacciones AXI Lite y AXI Full. Identifique claramente los handshake asociados a cada transaccion.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

La implementación fue un éxito en un 90%. Hubo dos problemas: en primer lugar, las salidas de led_g y led_r estaban intercambiadas. Para la implementación en el video, para que el auto simualdo "avance" cuando la luz se encontraba en verde y se "detuviera" con rojo, se intercambió la asignacion en el RGB de la señal de verde y roja en el archivo de constraints. Al hacer esto, claramente la asignación de los botones verde y rojo también se modifica (lo que estaba programado como verde ahora sale como rojo y viceversa), pero esto resulta en una correcta traducción a la tarjeta: verde es verde y rojo es rojo.

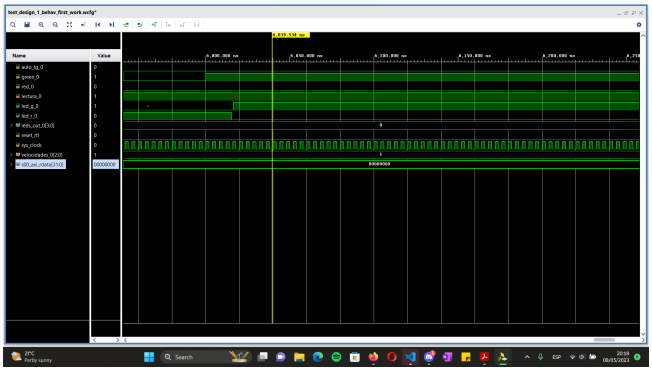

FIGURE 2: Delay en Timing Simulation

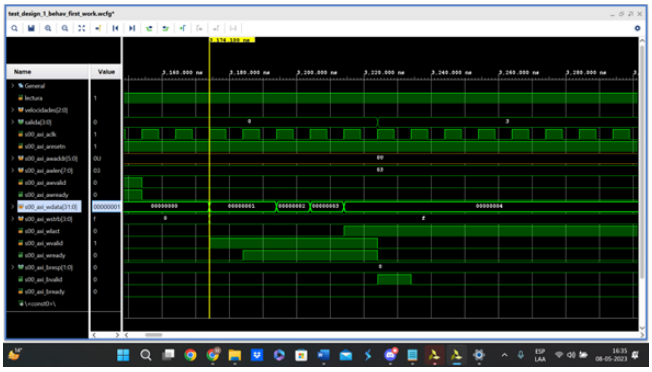

FIGURE 3: Transacciones AXI Full

El único problema que no se logró solucionar es que el modo automático, implementado con el botón auto toggle, no funciona al mandarle el bitsream a la RAM. Esto puede deberse a la falta de un debouncer en el botón o algún error de implementación. Vale decir que en la simulación behavioral que se ejecutó antes de generar la síntesis, implementación y bitsream. Esta simulación muestra un funcionamiento correcto, como se puede ver en la figura 6. Se puede ver el cambio en el modo de funcionamiento al final, donde las luces verde y roja toman turnos de acuerdo a los datos guardados en el **Semaforo**.

V. CONCLUSIONES(0.2)

Se lograron transmitir los 4 datos de 32 bits con AXI Lite 2 al bloque de Semáforo, además de 19 datos entre AXI Lite 1 y AXI FUll y 4 entre este último y RAM Velocidades. Estos bloques se pueden ver en la figura 1.

Los handshakes, tanto de AXI Lite como de AXI Full pudieron observarse y sus resultados fueron de acuerdo a lo esperado, permitiendo la correcta comunicación.

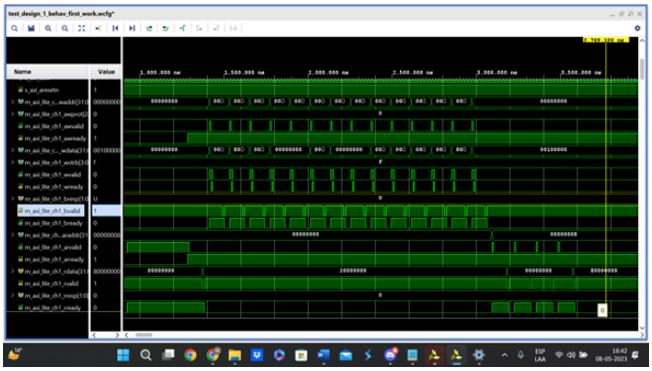

FIGURE 4: Transacciones AXI Lite 1

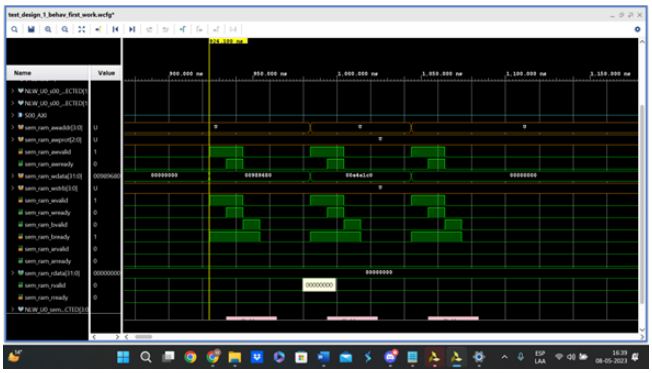

FIGURE 5: Transacciones AXI Lite 2

En la implementación real, se encontraron ciertos retrasos en la actualización de las señales, las cuales fueron del orden de los 16 ns. Esto se evidencia en la figura 2. No obstante, no se pudieron observar las variables dado a que no aparecían en el Scope de Vivado, por lo que no se podían agregar al waveform.

El comportamiento final del modo automático no fue el deseado. Se esperaba que, al presionar el botón asociado, comenzara a cambiar la luz verde a roja y viceversa automáticamente. Esto puede deberse a la falta de un debouncer para el botón o a fallas en la implementación real que en la simulación Behavioral no se revelaban.

También la salida de luces rojas y verdes del RGB estaban intercambiadas al hacer la implementación en la tarjeta, lo cual peude deberse al mismo problema de estar implementando ya no algo simulado sino que algo real.

VI. TRABAJOS FUTUROS (0.2)

En primer lugar, para trabajos futuros el primer paso sería arreglar el modo automático. Actualmente, no se cambia correctamente entre modos o, si se hace, esto no es evidente. Posibles soluciones serían implementar un debouncer, de ser ese el problema, o cambiar la lógica de **Semáforo**, que puede ser simulable pero presentar problemas al implementar.

Otro cambio que se podría realizar en trabajos futuros sería implementar un mayor flujo de datos mediante AXI. En este proyecto se generaron ráfagas para guardar 4 datos

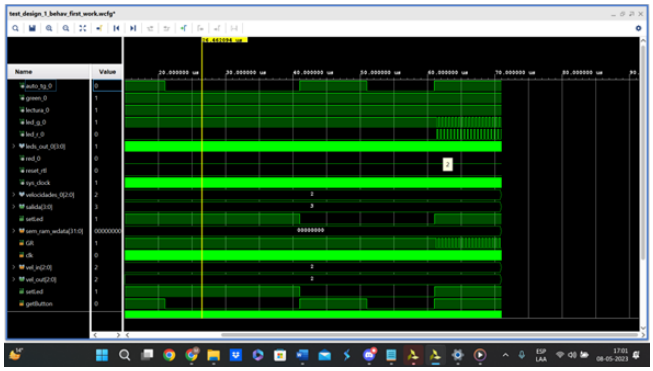

FIGURE 6: Funcionamiento proyecto en Simulation Behavioral

en la RAM periférica, sin embargo esto puede escalar a muchos mas datos, lo que indirectamente traerá un mayor número de velocidades aplicables al proyecto. Si bien la selección se hace con 3 switches, permitiendo 8 velocidades solamente (y en nuestro caso restringiendo a entradas que cambian la velocidad las "000", "001", "010", "100"), esto puede modificarse para que un mayor número de velocidades sea permitido.

También una mayor integración entre los bloques puede realizarse (agregando, por ejemplo, el trans, que es traductor de switch a dirección de la ram, como un componente).

Se debe tener cuidado con la implementación de la comunicación axi al querer agregar más velocidades ya que, en primer lugar, se debe cambiar el periférico aumentando su capacidad (que actualmente era de 16 datos de 32 bits o 4 bytes), por lo que el tema de las direcciones puede cambiar. También, hay que tener en cuenta que en los archivos .coe, al escribir datos de 32 bits, para mantener mensajes "alineados", la dirección dentro de la ram va aumentando de 4 en 4. Esto sucede ya que se deja siempre los dos ultimos bits con valor "00" y lo que aumenta es del tercero en adelante (valor "100", es decir, 4).

Otro aspecto que un trabajo futuro puede implementar es agregar un mayor set de luces al semaforo, con una amarilla por ejemplo, haciendo que esto modifique las velocidades de los autos (luces led). Se debe agregar entonces otro boton que pueda configurar la luz amarilla de forma manual, y en el modo automatico, agregarla a la secuencia, agregando tambien en el coe correspondiente cual va a ser la duración de esta luz amarilla.

REFERENCES

[1] Xilinx (2019) AXI Traffic Generator v3.0: LogiCORE IP Product Guide pp. 7-40. Recuperado de: https://docs.xilinx.com/v/u/en-US/pg125-axi-traffic-gen .

[2] Xilinx. (2023). 64983 - Vivado IP Integrator - How to generate a testbench for the Block Diagram (BD). Xilinx.com. Recuperado de: https://support.xilinx.com/s/article/64983?language=en$_U S$?.