Fecha de entrega 10 de mayo de 2024.

Conversor de hora en formato UTC

TOMÁS GUZMÁN[1], CARLOS STAPPUNG[1]
[1]Pontificia Universidad Católica de Chile (e-mail: tomas.guzman@uc.cl, carlos.stappung@uc.cl)

SÍ 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 Nuestro proyecto calcula la hora de un país en formato UTC, en base a una hora y minutos ingresados en UTC 0, y cierto índice que representa un país. Utiliza diversas máquinas de estado, dependientes de reloj o de botones, además de módulos con puertos AXI y ATGs en *Advance Mode* y *Test Mode*, para poder obtener las diferencias horarias de una RAM y calcular posteriormente la hora del país seleccionado. Se logró lo esperado, obteniéndose los resultados correctamente, aunque hizo falta evitar un problema lógico que aumenta las horas y minutos al introducir valores de tiempo iniciales, y un mayor control de los datos ingresados

INDEX TERMS VHDL, máquina de estados, reloj.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

PARA este proyecto decidimos implementar un conversor de hora en formato UTC (*Coordinated Universal Time*). En pocas palabras, este es capaz de recibir una hora y minutos específicos en UTC 0, ya sea de parte de los switches o de la VIO (dependiendo del modo de operación), para después seleccionar un índice el cual revisa un bloque de memoria RAM con valores configurados mediante un ATG en *Test Mode*. Los valores presentes en esta memoria indican la diferencia horaria, con la cual es posible calcular la hora y los minutos actuales en cierto país, y presentar unos de los dos valores en la tarjeta mediante los leds simples y el led RGB, dependiendo de la opción escogida.

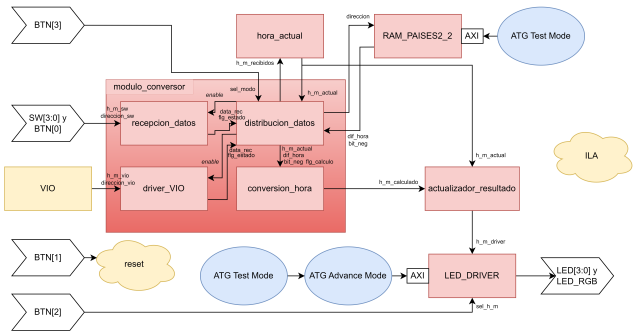

FIGURE 1: Diagrama de bloques simplificado del sistema.

En la figura 1 se presenta un esquema simplificado del sistema con los bloques principales que se utilizan. Como se podrá notar, la información de la hora y minutos iniciales y la selección del país mediante una dirección se puede realizar mediante switches y un botón o mediante el VIO. La selección del modo (modo switches o modo VIO) depende del botón 3, el cual ingresa a *distribucion_datos* para definir cuál información interpretar.

En el caso del modo switches, se tiene que realizar el ingreso de la información por pasos: se dejan los switches en el valor deseado, y al apretar el botón 0 el bloque *recepcion_datos* recibe un primer bus de datos de 4 bits. Al enviar 2 buses de datos se termina de ingresar la hora, después los siguientes 2 buses de datos representan los minutos iniciales, y por último los siguientes dos buses de datos representan la dirección de la memoria RAM (o el país). Una vez ingresado el primer índice, los siguientes 2 buses de datos seguirán representando el índice deseado (es decir, no se podrá ingresar una nueva hora y minutos iniciales).

En el modo VIO, *distribucion_datos* toma en cuenta la información proveniente de *driver_VIO*. Este bloque recibe 3 buses de datos de 8 bits provenientes del VIO, que representan la hora y minutos iniciales y el índice deseado. La máquina de estados presente en *driver_VIO* depende del reloj, por lo que en el modo VIO las operaciones son inmediatas. Cabe notar que, al igual que en el modo switches, la primera vez la hora y los minutos ingresados son considerados, pero en las siguientes iteraciones solo el índice es considerado, sin poder ser posible cambiar los valores iniciales de tiempo.

Después, la información ingresada entra al bloque *distribucion_datos*, la cual se encarga de interpretar la información y enviarla a los bloques necesarios, y una vez alcanzado

cierto punto acumular la información necesaria para realizar los cálculos. Una vez recibido el tiempo actual, envía la hora y los minutos a *hora_actual*, el cual se encarga de actualizar los minutos cada 60 segundos y las horas de ser necesario. La dirección la envía a *RAM_PAISES2_2*, la cual actualiza de esta forma los valores de diferencia de hora (8 bits) y el bit negativo (1 bit): si este último es 1, el país objetivo tiene una diferencia horaria menor a UTC 0, es decir, UTC -X con X la diferencia horaria.

Cuando los bloques receptores terminan de recibir un índice, envían una señal a *distribucion_datos* indicando esto. Al llegar la señal a este bloque, una máquina de estados secundaria inicializa un proceso, con el cual se juntan los datos necesarios para realizar la conversión de la hora en *conversion_hora*. Una vez actualizados estos datos, una señal para iniciar los cálculos se activa, y esto lleva a que en este último componente de *modulo_conversor* se calcule la hora, dependiendo de la hora actual, en el país indicado.

Posteriormente este resultado es recibido por *actualizador_resultado*, el cual almacena este valor y lo actualiza dependiendo de la hora indicada por *hora_actual*. Si es que los minutos o las horas cambian en este bloque, el bloque de actualización cambia el resultado recibido aumentando en uno el valor correspondiente.

Por último, la hora y los minutos son entregados a *LED_DRIVER*, el cual realiza la transformación correspondiente para poder visualizar el resultado en los leds. Cabe notar que, dado que no hay suficientes leds para representar las 23 horas máximas o los 59 minutos máximos, se utilizó el led RGB para poder representar "escalones" de tiempo para cada caso: al mostrarse las horas, la presencia de led RGB rojo implica que al valor mostrado por los leds simples se le debe sumar 16. Al mostrarse los minutos, ocurre lo mismo, con el led RGB rojo mostrando que se debe sumar 16 al valor de los leds, el led RGB amarillo que se debe sumar 32, y el led RGB celeste que se debe sumar 48. Para pasar de mostrar la hora a mostrar los minutos, se debe presionar el botón 2.

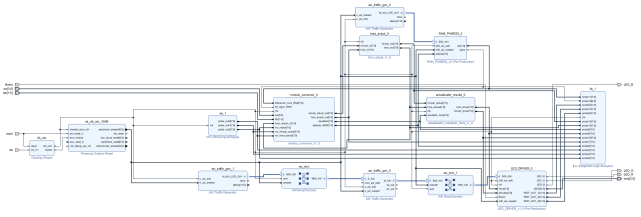

FIGURE 2: Diagrama de bloques completo del sistema.

En la figura 2 se presenta el diagrama de bloques completo, el cual incluye además bloques provenientes de Vivado como un *clk wizard*, bloques *AXI smart connect*, un bloque ILA y un bloque *clk wizard reset*. Sobre este último, sirve para poder reiniciar los bloques asociados a los AXIs. No es estrictamente necesario, pero se incluyó para evitar avisos de errores críticos por parte de Vivado. El botón 1 controla el reinicio del *clk wizard reset*.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

El avance realizado en cada uno de los puntos pedidos se presenta a continuación:

AO1 (100%): El bloque *modulo_conversor* se encuentra compuesto por 4 *components* distintos: *recepcion_datos*, *driver_VIO*, *distribucion_datos* y *conversion_hora*. Este bloque funciona correctamente, generando un resultado correcto dependiendo de los valores ingresados, por lo que se cumple con lo pedido.

AO2 (100%): El proyecto final se encuentra compuesto por diversos IPCores de elaboración propia. En *modulo_conversor* se incluyó un parámetro genérico booleano para poder elegir el modo en el cual inicializa el proyecto (por defecto inicializa en el modo switches); en *hora_actual* se incluyó un parámetro genérico entero para configurar el valor máximo del contador de los segundos (por defecto, el valor sirve para un reloj de 100 MHz); y en *LED_DRIVER* se incluyó un parámetro genérico el cual altera la intensidad de la luz. Todos funcionan como es debido.

AO3 (100%): Se utilizó tanto un ATG maestro en *Test Mode* como uno en *Advanced Mode* conectados a *IP cores* esclavos creados por el equipo: uno para guardar los datos de diferencia horaria de los países (en *RAM_PAISES2_2*), y otro para definir los colores del led RGB (en *LED_DRIVER*). Ambos funcionan correctamente, lo cual se comprueba dado que en la implementación del proyecto se obtuvieron los resultados esperados.

AC1 (100%): Se implementó una serie de máquinas de estados en diversas partes del proyecto (principalmente en los componentes del bloque *modulo_conversor*) las cuales funcionan correctamente. Se reconocen como máquinas de estados ya que interpretan los valores recibidos de forma distinta, dependiendo del estado actual.

AC2 (100%): En el diagrama de bloques se añadieron un módulo VIO y uno ILA. Es importante notar que el primero se puede utilizar tras presionar el botón 3 (el botón más a la izquierda).

AC3 (100%): Se utilizan dos operadores diferentes (+ y - en *conversion_hora*) y dos atributos diferentes (*'EVENT* en diversos bloques, como *driver_VIO*, y *'LENGTH* en *conversion_hora*).

AC4 (100%): En varios *process* se incluyeron variables para actualizar valores instantáneamente, cumpliéndose su función. Por ejemplo, *hora_actual* posee varias variables.

AC5 (100%): Las máquinas de estados y los *process* utilizados se tratan de código secuencial, el cual depende de un valor anterior. El código concurrente no requiere conocer el valor anterior, y también fue aplicado, por ejemplo, al final del bloque *distribucion_datos*, al actualizar una señal de salida con una señal interna, sin importar su valor previo.

AC6 (100%): Se aplicó una *function* en *distribucion_datos* y un *procedure* en *hora_actual*, y se reconoce la diferencia entre ambas: las funciones son bloques de código descriptivo que se pueden instanciar, para así ingresar datos y obtener un único dato de salida. Los procedimientos hacen lo mismo, pero pueden retornar múltiples datos de salida. La función

utilizada retorna un *std_logic* 1 si es que se ingresa un booleano *true*, y uno 0 si el booleano es *false*. El procedimiento creado se utilizó para reiniciar múltiples señales y variables a ciertos valores predeterminados.

AC7 (100%): Como función nueva que no se ha visto en el curso, se utilizó el IPCore AXI SmartConnect que viene incorporado en Vivado. Su función es conectar uno o más AXI Masters con uno o más AXI Slaves. Tal como dice la documentación, se pueden conectar hasta 16 Slaves y 16 Masters. Este bloque viene a ser el reemplazo del AXI interconect v2, dado que cumplen el mismo propósito, pero el SmartConnect se adapta de manera automática con la mínima intervención del usuario [1].

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

Se llevó a cabo una simulación de tiempo post-implementación (*post-implementation timing simulation*) para el bloque *modulo_conversor*, para así poder observar ciertas características no visibles en las simulaciones de comportamiento.

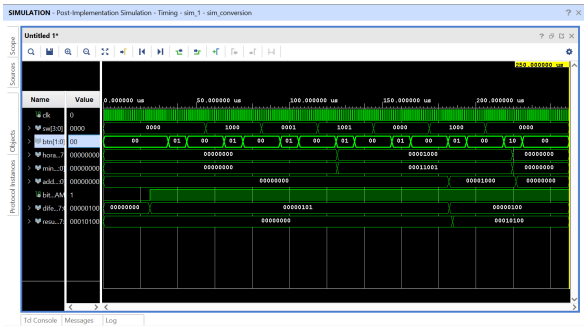

FIGURE 3: Simulación de tiempo post-implementación para *modulo_conversor*.

Como se puede observar en la figura 4, la señal *btn[0]* pasa de 0 a 1 y de 1 a 0 varias veces. Esto representa al usuario presionando el botón más a la derecha para ingresar los datos de los switches para lograr representar las horas y minutos iniciales, y la dirección (o *address* en este caso). El ingreso de un primer bus 0000 y un segundo bus 1000 sirve para representar la hora 00001000, o las 8, lo cual se ve reflejado en la señal de salida *hora_actual_out*. Lo mismo ocurre con los siguientes dos buses, que se ven reflejados en los minutos, y los siguientes dos, que se ven reflejados en la señal *address*.

Sin embargo, cabe notar que los cambios no son inmediatos. Por ejemplo, al presionar por cuarta vez el botón, las horas y los minutos no se actualizan instantáneamente, ya que requieren de un flanco de subida del reloj para cambiar sus valores. Por otro lado, al ocurrir el flanco de subida, pasa cierto tiempo, aunque mínimo, antes de que los valores de hora y minuto se actualicen. Estas no idealidades, aunque menores, pueden generar problemas en el sistema y deben ser tomadas en cuenta. Por ello, utilizar máquinas de estado que esperen cierto tiempo para que se estabilicen las señales es una buena idea.

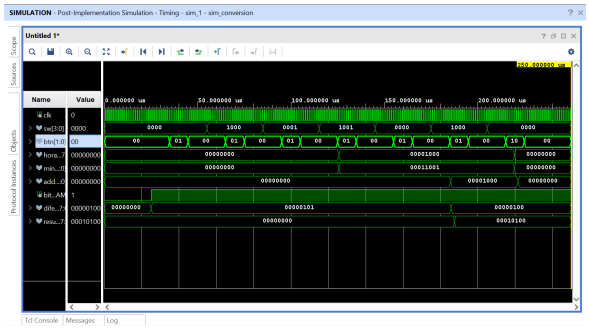

FIGURE 4: Simulación de tiempo post-implementación para *modulo_conversor*.

Otro aspecto que hubiera sido interesante mostrar es la diferencia en cuanto a la velocidad de actualización entre las señales y las variables. Las variables poseen la ventaja que, dentro de un *process*, cambian de inmediato, mientras que las señales esperan a que se termine el proceso para cambiar. Sin embargo, no es posible mostrar esto en una simulación dado que Vivado no permite mostrar variables en la ventana de simulación (a pesar de tener la opción disponible). Aun así, para ejemplificar un tanto la importancia de saber escoger entre una variable o una señal, me gustaría hablar acerca de un caso en específico que ocurrió durante nuestro trabajo. En la máquina de estados secundaria del bloque *distribucion_datos* inicialmente usamos una variable para representar los estados, pero esto llevaba a que los cálculos se llevaran a cabo con señales antiguas. Resulta que el cambio de la variable era tan instantáneo, que las señales no alcanzaban a actualizarse a tiempo. Esto nos llevo a tener que cambiar esta variable a señal, y así logramos resolver el problema.

Por otro lado, en la realización del proyecto se realizaron tres transacciones AXI. La primera, un AXI lite, con un ATG en *Test Mode*, que entrega información al bloque *RAM_PAISES2_2*. La segunda y tercera corresponden a la implementación del bloque ATG en *Advance Mode*, donde la primera transacción es en AXI lite desde un ATG en *Test Mode* para cargar los datos al segundo ATG en *Advance Mode* y la segunda es en AXI Full para entregar los datos a la RAM *LED_DRIVER*. A continuación, mostraremos la correcta ejecución de estas tres transacciones. Cabe destacar que las figuras a continuación vienen de un bloque ILA.

A. AXI LITE (ATG TEST MODE PARA CARGAR RAM_PAISES2_2)

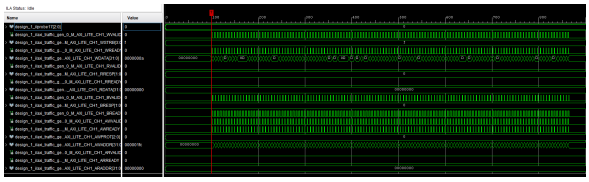

FIGURE 5: Transacción AXI LITE.

La figura 5 muestra la transacción de la comunicación

AXI LITE entre el bloque ATG en *Test Mode* con el bloque *RAM_PAISES2_2*. En esta se hace envío de 128 datos, por lo que se decidió hacer un zoom a la ventana para que se pueda ver más claramente la transacción.

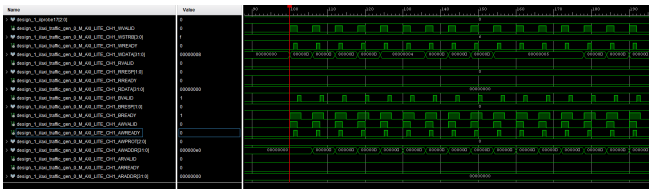

FIGURE 6: Transacción AXI LITE (zoom in).

La figura 5 muestra todas las señales de AXI LITE. M_AXI_LITE_WDATA es la señal que envía nuestra información que se escribe en la *RAM_PAISES2_2*, mientras que M_AXI_LITE_WDATA_AWADDR indica la dirección donde se escribe esta información. Por otro lado, las señales M_AXI_LITE_WVALID y M_AXI_LITE_WREADY son las encargadas de realizar el *handshake* entre ambas partes.

B. AXI LITE (ATG TEST MODE PARA CARGAR LA ATG ADVANCE MODE)

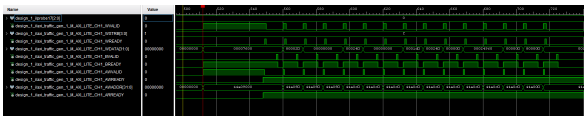

FIGURE 7: Transacción AXI LITE.

La figura 7 muestra todas las señales de AXI LITE. M_AXI_LITE_WDATA es la señal que envía nuestra información al bloque ATG y escribe en su RAM, mientras que M_AXI_LITE_WDATA_AWADDR indica la dirección del ATG donde se escribe esta información. Nuevamente las señales M_AXI_LITE_WVALID y M_AXI_LITE_WREADY son las encargadas de realizar el *handshake* entre ambas partes.

C. AXI FULL (ATG ADVANCE MODE PARA CARGAR LED_DRIVER)

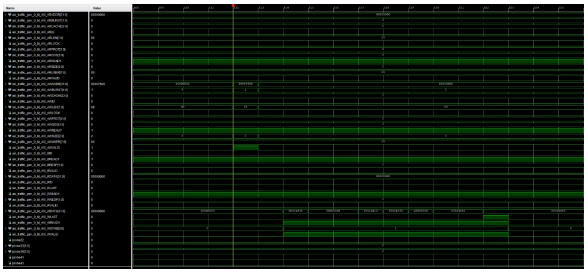

FIGURE 8: Transacción AXI FULL.

La figura 8 muestra todas las señales presentes en el protocolo AXI Full. Dado que tenemos una gran cantidad de señales, se extrajo las principales en la figura 9.

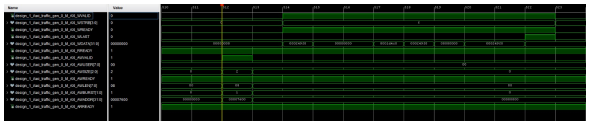

FIGURE 9: Transacción AXI FULL simplificada.

En la figura 9 podemos ver de manera mas sencilla el envío de información a través de la señal M_AXI_WDATA que envia los códigos RGB para los colores usados en *LED_DRIVER*. Además, somos capaces de ver la señal M_AXI_AWDADDR que envía el *address* de nuestra RAM. También tenemos M_AXI_AWBURST que vemos que está enviando el modo de BURST y la señal M_AXI_AWLEN, que indica la cantidad de datos que se envían en un BURST. Por último, tenemos la señal M_AXI_WLAST, la cual tiene la función de realizar el *handshake* de la transacción y esta se activa cuando se envía el último dato de información desde el Master, además de que también tenemos el handshake de las señales M_AXI_LITE_WVALID y M_AXI_LITE_WREADY.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

La implementación se pudo completar correctamente. Se obtuvieron los resultados esperados, pudiéndose observar en los leds las horas y minutos de los países indicados. Logramos resolver el problema de tener que presionar nuevamente el botón 0 de cambio de estado para poder observar los resultados de cálculo, al igual que ciertos problemas asociados a la implementación de reinicios en bloques dependientes de señales de reloj.

Lo que faltó fue evitar aumentar las horas y minutos tras introducir horas y minutos iniciales nuevos, e incluir un mayor control de los datos ingresados por usuario, para evitar problemas derivados de datos fuera de rango.

V. CONCLUSIONES(0.2)

- La obtención de los 6 buses de datos de 4 bits en el bloque *distribucion_datos*, en el modo VIO, se esperaba después de 6 ciclos de reloj, lo cual se cumplió, y se pudo demostrar que la transmisión de datos fue realizada efectivamente gracias a las pruebas físicas.
- Lo mismo ocurrió con el modo switches, con la diferencia de que debía ocurrir después de 6 presiones del botón 0.
- Se esperaba que el componente *distribucion_datos* iniciara la máquina es estados secundaria tras recibir un nuevo índice, lo que se cumplió. Sin embargo, cabe notar que fue necesario cambiar la variable *state_local* a una señal, ya que inicialmente el cambio de estados era más rápido que el cambio de las señales de salida, lo que se pudo reconocer físicamente dado que era necesario

volver a apretar el botón 0 para obtener la hora del país indicado.

- La señal de *enable* utilizada para *recepcion_datos* y *driver_VIO* cumplió su función, siendo capaz de reiniciar las señales *batch_1*, *batch_2*, *flg_out* y la variable *state* para cada caso, a valores 0.

VI. TRABAJOS FUTUROS (0.2)

En primer lugar, uno de los aspectos más relevantes a mejorar es el del aumento de las horas y los minutos en *actualizador_resultado* al ingresar los valores de tiempo inicial. Por cómo está armado el sistema lógico, al cambiar la hora y los minutos en *hora_actual*, a los valores de hora y minutos de salida del actualizador del resultado se le suman 1 respectivamente. Por ello, inicialmente o al cambiar de modo de operación (switches a VIO o viceversa), al ingresar el tiempo inicial la hora en los leds cambia. Una forma de arreglar esto sería enviar una señal desde *distribucion_datos* la cual active el bloque de actualización tras recibir los minutos iniciales.

Otra posible mejora sería añadir un control a los valores ingresados. Actualmente se asume que se ingresarán valores de horas, minutos e índices razonables (minutos que no superen los 59, por ejemplo), pero los usuarios se pueden equivocar, y actualmente no existe un control para los errores de ingreso de datos.

Por último, otra posible mejora sería alterar el bloque *modulo_conversor* para poder recibir horas y minutos iniciales los cuales no necesariamente sean UTC 0. Por ejemplo, poder ingresar la hora en Chile con UTC -4 (actualmente) y obtener la hora en otro país. Esto implicaría tener que recibir más datos y alterar la operación del bloque *conversion_hora*, pero aumentaría la utilidad del proyecto.

REFERENCES

[1] AMD Xiling, 'AXI SmartConnect," URL:https://www.xilinx.com/products/intellectual-property/smartconnect.html.