Fecha de entrega 07 de julio de 2025.

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

TEST COMPLETO "ZYBOOSTER"

SAMUEL RODRÍGUEZ VANIER[1], MARTÍN VALDÉS VERGARA[1]

[1]Pontificia Universidad Católica de Chile (e-mail: srodriguezva@uc.cl, mvaldv@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 presente consiste en un testeo completo de los periféricos, tanto de la tarjeta Zybo Z7-10 como del BoosterPack. Existen dos modos de testeo: un modo rápido, el cual aprovecha la pantalla LCD para mostrar el estado de varios periféricos a la vez, y un modo paso a paso, donde cada periférico se testea de manera secuencial y con instrucciones detalladas.

En cuanto a tecnologías utilizadas, se hace uso de los periféricos de las tarjetas antes mencionadas, los cuales transaccionan información con el procesador ARM-Cortex en el PS de la Zybo Z7-10 mediante protocolos como AXI, I2C y SPI, y a su vez el protocolo UART permite al usuario interactuar con el procesador mediante una terminal abierta en el PC. Se modifican registros de periféricos para poder obtener una mayor simplicidad al momento de comprobar su funcionamiento.

Se puede probar la efectividad del programa, al detectar una pequeña falla en el joystick, que tiene un valor mínimo en el eje X de 31, cuando el límite debiese ser 0, por lo que se detecta exitosamente ese problema.

INDEX TERMS TESTEO, PERIFÉRICOS, I2C, SPI, UART, AXI, INTERRUPCIONES, BUZZER, Zybo z7, Booster Pack

I. ARQUITECTURA DE HARDWARE Y SOFTWARE (1 PUNTO)

El proyecto consiste en un testeo completo de los periféricos que componen a la tarjeta Zybo Z-10 y al BoosterPack, incluyendo leds, botones, switches, acelerómetro, joystick, sensores de temperatura y luz, micrófono y bocina, y el estado de los distintos periféricos se va mostrando en la pantalla LCD dependiendo del modo de testeo. Si se selecciona el modo de testeo rápido, se prueban todos de manera simultánea, y cuando se selecciona el modo de testeo paso a paso, cada periférico se prueba de manera secuencial según las instrucciones mostradas en pantalla. También existe una opción en el testeo paso a paso que permite al usuario seleccionar la canción a probar en la bocina mediante una terminal en el PC.

En cuanto al hardware del proyecto, se encuentra dividido en secciones respecto a las tareas que se realizan:

En primer lugar, se leen las entradas de los botones de ambas tarjetas e ingresan a módulos debouncer para evitar lecturas incorrectas. Los botones del BoosterPack también ven invertida su salida luego del debouncer, puesto que al presionarse bajan el voltaje en vez de subirlo.

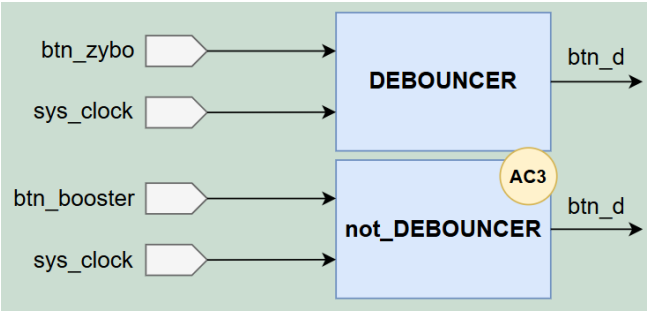

FIGURE 1: Periféricos botones

Posteriormente, los botones ingresan a un módulo que se encarga de concatenar todas la interrupciones a aplicar y enviarlas a un puerto de interrupción del Zynq7. Las interrupciones generadas por ambos timers son para diferencias la actualización de periféricos rápidos y lentos por separado, y la interrupción de luz permitirá que se active una señal una vez se supere el umbral de luz determinado en la configuración.

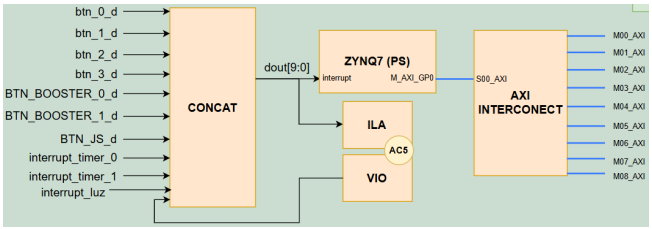

FIGURE 2: Interrupciones aplicadas

Por otra parte, existen módulos que se comunican por AXI al procesador y que permiten la interacción con periféricos asociados a los protocolos I2C (sensores de temperatura y luz) y SPI (pantalla LCD y ADC).

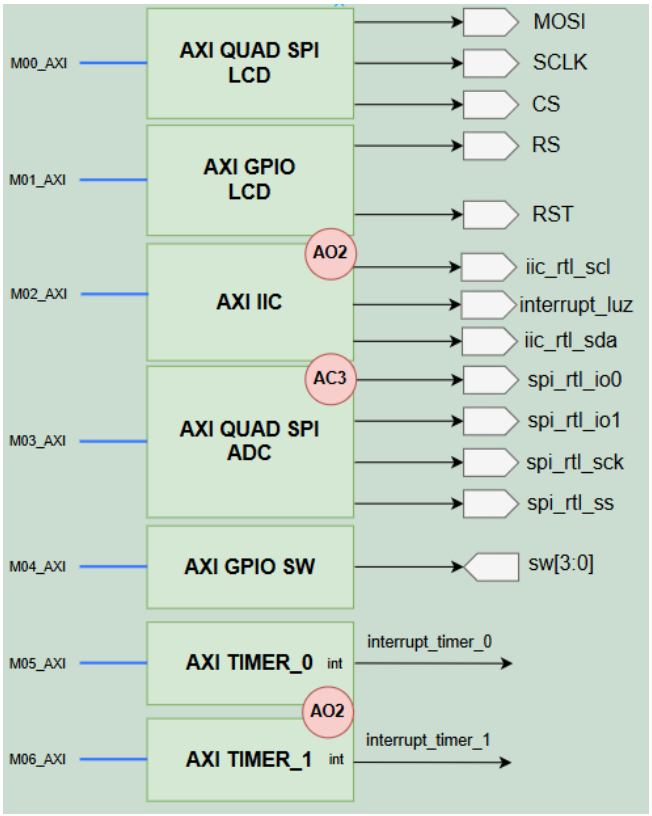

FIGURE 3: Protocolos I2C y SPI

Por último, para el testeo de la bocina en el BoosterPack y de los leds en la Zybo, se implementan 4 módulos adicionales.

- Buzzer_RTL: Maneja la señal PWM que luego es enviada a la bocina, contiendo dos opciones de canciones que el usuario puede seleccionar por UART.
- LED_TEST_DRIVER: Activa un parpadeo intermitente en los 4 leds de la Zybo, de manera que pueda verificarse si existen errores de funcionamiento.
- PWM_LED_RGB: Contiene la lógica interna de la señal PWM que es enviada al led RGB de la Zybo.
- RGB_IDLE_DRIVER: Activa el led RGB de manera que se realice un recorrido suave entre distintos colores del espectro.

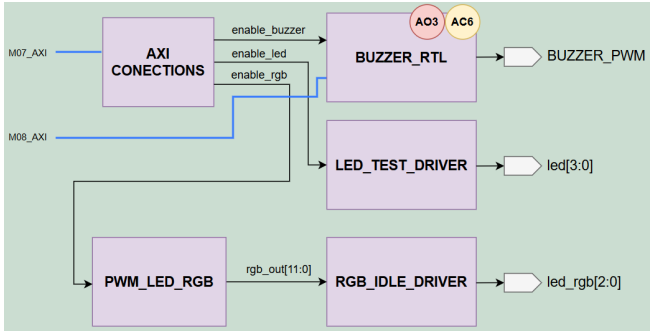

FIGURE 4: IP Core propios

En cuanto al software, el flujo de la máquina de estados para pasar por los distintos modos de operación se resume en el siguiente diagrama, que contiene ambos modos de operación y el flujo que se desea seguir.

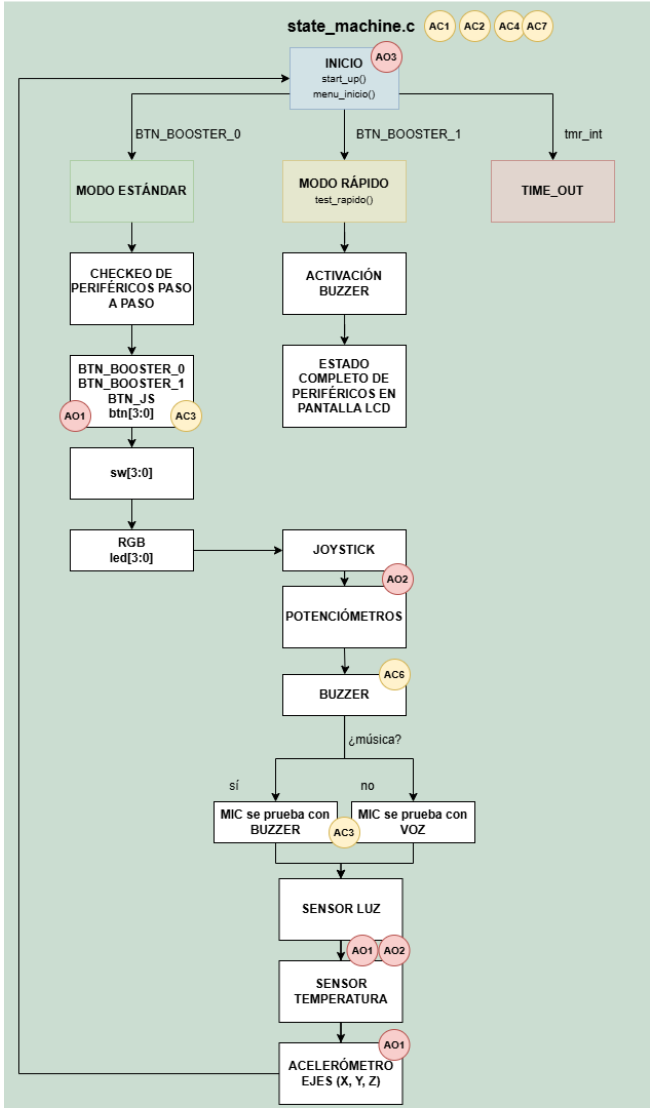

FIGURE 5: Diagrama software

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): *Descripción:* El código cumple con las funciones que habíamos planteado en nuestro proyecto, las cuales son mostrar secuencialmente en la pantalla LCD el estado de los periféricos que están siendo testeados, además de modificar los registros internos de configuración (bit de polaridad del pin de interrupción) y de límite máximo para el sensor de luz, con motivo que gatillar posteriormente una interrupción. De acuerdo a la arquitectura presentada en la sección I, esto se implementó después de cada testeo de los periféricos, mostrando primero las lecturas actuales y posteriormente los valores logrados, en consecuencia con el diagrama de software, asociado al bloque AXI IIC en los diagramas de hardware. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AO2 (100%): El código compila correctamente, pasa por los test de debug y se carga correctamente en la tarjeta Zybo Z7-10, cumpliendo con lo solicitado al leer los periféricos periódicamente con timers y generar una interrupción alcanzado cierto umbral de intensidad lumínica. De acuerdo a la arquitectura presentada en la sección I, dicha actividad se cumple en los bloques AXI IIC y los AXI TIMER_#, específicamente en la lectura y actualización de los valores de joystick, potenciómetros, sensores de temperatura y luz, así como en la interrupción asociada a este último. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AO3: Tanto la descripción de hardware aplicada al IP Core creado, como el código que permite la comunicación por UART con el PC, pasaron por los test de debug y se cargan correctamente a la tarjeta Zybo Z7-10. De acuerdo a la arquitectura presentada en la sección I, dicha actividad se cumple en el bloque BUZZER_RTL, cuya función es recibir una canción seleccionada por el usuario (via UART o via interrupción por botón) y generar la salida PWM que va hacia la bocina del BoosterPack. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC1: El código de la máquina de estados permite un correcto funcionamiento del flujo del proyecto, siendo correctamente cargado en la tarjeta Zybo Z7-10. De acuerdo a la arquitectura presentada en la sección I, dicha actividad no está asociada con un bloque del hardware en específico, puesto que requiere del funcionamiento de todos los periféricos involucrados. Está asociado al código del archivo main.c, y en el diagrama del software se puede identificar el flujo en que se cumplen las distintas actividades. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC2: Se implementaron estructuras (para guardar los valores asociados a potenciómetros y joystick), arreglos (para contener la información en pixeles de las imágenes cargadas), punteros (para la configuración de la bocina por UART), funciones (modularizando la lectura de muchos periféricos) y macros (al declarar las direcciones de algunos componentes y registros). De acuerdo a la arquitectura presentada en la sección I, dicha actividad no está asociada con un bloque del hardware en específico, puesto que requiere del funcionamiento de todos los periféricos involucrados. Está asociado al código del archivo main.c. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC3: Al realizar un testeo de todos los periféricos disponibles, tanto en la tarjeta Zybo Z7-10 como en el BoosterPack, nos aseguramos de implementar más de dos periféricos adicionales a los solicitados en la AO1. De acuerdo a la arquitectura presentada en la sección I, dicha actividad fue orientada al uso de los botones y del micrófono para su posterior testeo. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC4: Fue posible implementar el boot del código completo del proyecto em la tarjeta Zybo Z7-10, de manera que, si se apaga y se vuelve a encender mientras está en modo QSPI, el código se reinicia y se puede aplicar el testeo de inmediato. De acuerdo a la arquitectura presentada en la sección I, dicha actividad no está orientada a un bloque del hardware en específico, sino a lo implementado en Vitis. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC5: Fue posible implementar los módulos de VIO e ILA para testear y monitorear la lógica de las interrupciones que ingresan al PS. Se habilitó un puerto para que VIO pudiese activar una interrupción de reseteo en el código y se conectó ILA para leer la transacción entre el bloque que concatena las interrupciones y el ZYNQ7. De acuerdo a la arquitectura presentada en la sección I, dicha actividad se relaciona a la implementación de los IP Core VIO e ILA y su conexión con los IP Core antes descritos. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC6: Fue posible describir el hardware del módulo que envía la señal PWM a la bocina del BoosterPack, el cual contiene las frecuencias para dos canciones distintas que pueden ser seleccionadas tanto por UART como por interrupción por botones. De acuerdo a la arquitectura presentada en la sección I, dicha actividad se relaciona al bloque BUZZER_RTL. *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstram y cargarlo en la ZYBOZ7.

AC7: (**COMPLETAR**)

III. RESULTADOS (3 PUNTOS)

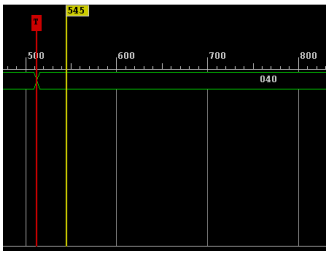

FIGURE 6: Prueba ILA NDEBOUNCER

De la imagen, se puede ver el momento en que se levanta la interrupción del botón. Con esto, probamos que el debouncer funciona correctamente, al visualizar solamente un cambio en el valor, y también, el valor 040 corresponde a la entrada 6 del GIC, que corresponde al botón del joystick del Booster Pack, que es un botón "Active Low", por lo que también prueba la negación que ocurre en el NDEBOUNCER.

```
void light_max() {//Interrupción de luz
    //Disable interrupts
    XScuGic_Disable(&GIC, INT_ID_LIGHT_IQR);
    light = 1;
}
```

FIGURE 7: Interrupción de luz captada en modo debug

En la imagen, la línea verde representa la línea que corre el procesador, que se llegó a esta mediante un breakpoint en la función light_max, la cual solamente está conectada a la interrupción que es generada por el sensor de luz.

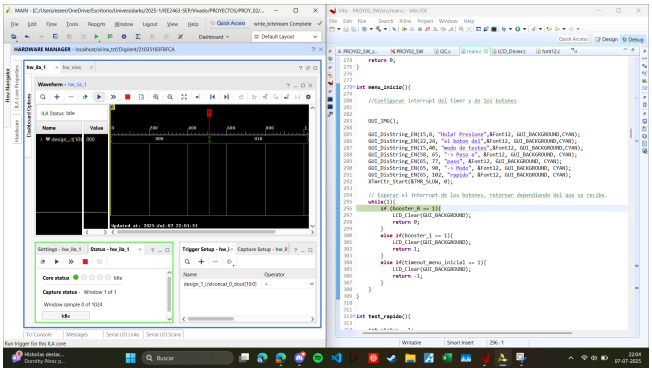

FIGURE 8: Vivado y Vitis simultáneo

Se puede ver como en simultáneo corren ILA y el debug del procesador, en el momento que se levanta la interrupción del botón.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Las principales dificultades en cuanto a la implementación fueron asociadas a las interrupciones de botones y del sensor de luz. Todo fue manejado con las direcciones de un registro interno en el ZYNQ7 usado por el GIC para acceder a los distintos flags que gatillan las interrupciones. Sin embargo, la actualización de los valores no era inmediata y, en el caso especial del sensor de luz, fue complejo implementar la interrupción al superar el límite superior configurado. Muchas veces dicha interrupción se activaba sin motivo o no respondía ante los estímulos presentados. La forma en que esto fue resuelto, es que se implementaron delays en el código principal, de manera que el reconocimiento de la interrupción ocurriese luego de haber estabilizado los valores de lectura de luz.

V. CONCLUSIONES(0.2)

- Con el testeo de periféricos aplicado, fue posible detectar una falla en el eje x del joystick del BoosterPack, debido a que el mínimo registrado fue de 31 unidades, siendo que el mínimo esperado era de 0 unidades, lo cual corresponde a un error del 3.02% (sobre los 1024 valores disponibles).
- Se logra conocer las limitaciones del procesador, al presentar problemas con las interrupciones cuando la tasa de refresco de la interrupción hecha por un AXI Timer bajan los 10 ms en tiempo. (A esta velocidad ya presentan problemas, como la falta de limpieza de la pantalla, pero aún así de puede utilizar).

VI. TRABAJOS FUTUROS (0.2)

Actualmente los valores de los periféricos son mostrados en pantalla tal como se reciben y se almacenan en Vitis, lo cual permite un testeo más cuantitativo y preciso. Sin embargo, requiere del conocimiento de las variables mínimas o máximas para cada uno, o de los valores ambiente que indican una lectura apropiada. Por este motivo, una oportunidad para mejorar este proyecto consiste en implementar un modo de testeo más amigable e interactivo con el usuario, haciendo uso de elementos gráficos que permitan un acercamiento incluso a usuarios inexpertos. Un ejemplo de esto podría ser normalizar la posición de los potenciómetros y mostrarlo en una barra de progreso, o también mostrar la acción del acelerómetro de manera que si se inclina la tarjeta se vea cómo la pantalla se va pintando en dicha dirección, entre otros. Otra implementación útil sería mejorar la interfaz del testeo rápido, dado que, si bien se muestran todos los valores y todos los periféricos, esto causa una sobrecarga en el procesador, por lo que se podría dividir tareas en más de un procesador.

REFERENCES

[1] Rojas, F. (2025). “ IEE2463 Sistemas Electrónicos Programables” in https://github.com/IEE2463-SEP.