Fecha de entrega 7 de Julio de 2025.

ZYBOMON 2

RODRIGO ELGUETA[1], JOSÉ QUIMI[1]
[1]Pontificia Universidad Católica de Chile (e-mail: rodrigo.elgueta@uc.cl, jquimi@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 Zybomon 2 es la continuación del proyecto *Zybomon*, que simula una clásica batalla Pokémon®. Para su desarrollo se utilizó la FPGA Zybo Z7 y el BoosterPack. Donde por medio software programado en la placa FPGA se genera el ciclo de juego, el boot del programa y carga e instanciación de los diferentes archivos que ejecutara en el programa, además, se agrega la interacción con el BoosterPack, el cual entrega la posibilidad de comunicarse con los diferentes periféricos que dispone, tales como la pantalla LCD, sensor de temperatura, luminosidad, acelerómetros y elementos como potenciómetros, botones, joystick, entre otros. Estos periféricos se comunican con el software por medio de distintas interrupciones que permiten al procesador manejar las diferentes rutinas que ejecutan las características del proyecto, ya sea el flujo de batalla y menús, como las diferentes interacciones con los periféricos. Se logró implementar correctamente la interacción Software-Hardware, así como los elementos individuales de cada uno, como el flujo de menús, asignación de memoria, flujo de juego, etc. Y, por el lado del hardware, la comunicación con los diferentes periféricos, interrupciones y respuesta a través de los mismos hacia el usuario.

INDEX TERMS VHDL, State machine, C, Texas BoosterPack, Zybomon.

I. ARQUITECTURA DE HARDWARE Y SOFTWARE (1 PUNTO)

ZYBOMON 2 es la continuación del proyecto anterior ZYBOMON, incorporando un sistema de audio y una pantalla LCD para mostrar la información del juego gráficamente. Este proyecto integra el diseño y manejo de hardware con la implementación de software, haciendo uso de un procesador programable de un núcleo y una tarjeta booster para la integración de interacciones con periféricos, por medio de interrupciones del procesador, como se muestra en la figura 1.

En primer lugar, el flujo del juego sigue siendo una batalla por turnos entre 2 jugadores que buscan dejar al oponente sin zybomones para ganar. Los zybomones se componen de 4 atributos de vida máxima, ataque, defensa y velocidad con valores aleatorios, además de 4 ataques cada uno con un poder y una cantidad de usos determinada. Además, tanto los zybomones como los ataques tienen tipos elementales que influyen en el cálculo de daño.

El juego comienza generando los datos de los entrenadores y pidiendo por comunicación serial UART el nombre del entrenador junto con el nombre que le asigna a cada uno de sus zybomones. Luego comienza el combate en el cual el jugador puede navegar por distintos menús para revisar sus estadísticas y elegir usar un ataque o cambiar el zybomon activo. Esto se realiza con una máquina de estados con un estado para cada etapa del juego, además en cada ciclo se escribe la pantalla LCD verificando los píxeles ca cambian para ahorrar recursos.

Una vez el jugador realiza su elección, le toca al segundo jugador. Este es controlado por un bot que tiene 50% de elegir el ataque óptimo o elegir uno aleatorio con usos restantes. Las acciones de ambos jugadores se colocan en una cola y se procesan en el orden adecuado dependiendo de la velocidad de sus zybomones.

Por último se comprueba si un jugador se quedó sin zybomones y se termina el juego indicando con una pantalla final y música si ganó o perdió.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): Se actualiza la pantalla de forma dinámica mediante una función que compara un mapa de bits de lo que se desea mostrar con el mapa de bits del cuadro anterior, de forma de solo escribir los píxeles que cambian de un cuadro a otro ahorrando recursos. Además, se utilizó el sensor de luz, el acelerómetro y el micrófono de la tarjeta booster. Con respecto al sensor de luz, se modificaron sus registros para gatillar una interrupción cada vez que alcance un nivel de luz superior o inferior a márgenes específicos, en este caso cuando está por debajo de 10 lux y sobre 100 lux.

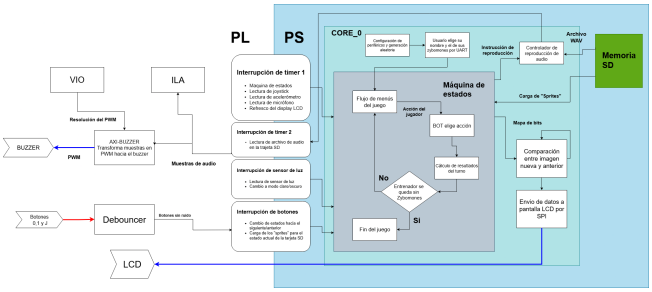

FIGURE 1: Diagrama de hardware y software de ZYBOMON 2

AO2 (100%): Se implementaron dos timers con interrupciones asociadas. El primero se encarga de reproducir la música de juego leyendo un archivo WAV. Luego envía las muestras de audio al periférico axi_buffer a una frecuencia múltiplo de la frecuencia de muestreo del archivo. El segundo timer se encarga de leer los periféricos, manejar la máquina de estados principal del juego y de actualizar la pantalla LCD. Además, se implementó una interrupción del sensor de luz, la cual se encarga de alternar entre el modo oscuro y modo claro de la pantalla de juego dependiendo del nivel de luminosidad que recibe el sensor.

AO3 (100%): Se creó el IP CORE axi_buzzer encargado de recibir 12 muestras de audio desde el procesador y enviarlas en orden y a la frecuencia de reproducción adecuada de 8 kHz buzzer de la tarjeta booster mediante la generación de una PWM. En el código se emplean punteros en funciones constantemente permitiendo inicializar distintas estructuras necesarias para el funcionamiento del software, por ejemplo los mapas de bits se inician empleando la función Paint_Bitmap la cual recibe el puntero de una variable de BITMAP. Además, los nombres de los entrenadores y de los zybomones se configuran con el envío de datos por UART mediante la consola serial.

AC1 (100%): El flujo del juego se realiza mediante una máquina de estados que muestra los datos en la pantalla y establece las dimensiones de los menús conforme al estado actual en el que se encuentra mediante la función StateHandler. Además, la función advance_state se encarga de determinar el estado siguiente. La máquina de estados se implementó usando una estructura de stacks basada en [1].

AC2: (100%): Se definieron estructuras personalizadas en el código para manejar la información de los mapas bits, zybomones y ataques. Para cada uno de estos tipos se definieron funciones que, gracias al uso de punteros, permiten modificar las variables accediendo a la ubicación de su registro correspondiente en la memoria. Además, se emplearon macros para definir distintos parámetros de los periféricos, como por ejemplo el periodo de los timers.

AC3: (100%): Se utilizaron los botones y el joystick de la tarjeta booster como periféricos adicionales navegar por los menús y sus opciones.

AC4: (100%): Se logró programar una imagen de booteo en la memoria flash de la Zybo de manera de que es capaz de cargar la configuración de hardware y software con solo encenderla, sin la necesidad de un computador.

AC5: (100%): Se implementó un módulo VIO capaz de variar la resolución del PWM generado por axi_buzzer. Además, se agregaron dos módulos ILA que se encargan de revisar la transacción que recibe el axi_buzzer desde el PS y de monitorear los botones y las interrupciones de hardware.

AC6: (100%): El IP CORE axi_buzzer implementado permite comunicar el procesador con el buzzer en la tarjeta booster. De esta forma, el procesador puede enviar el audio que corresponda según el estado del juego presente en el software.

AC7: (100%): Se logró leer desde la tarjeta SD archivos WAV de audio para reproducir música compleja sin ocupar la memoria del procesador. También se logró implementar la lectura de archivos de imágenes BMP para aprovechar el espacio que ofrece la tarjeta SD y guardar "sprites" del juego, los cuales se cargan solo cuando se necesitan.

III. RESULTADOS (3 PUNTOS)

Se utilizó un módulo ILA para ver la transacción entre el procesador y el hardware axi_buzzer. Se puede apreciar de la figura 2 que se escriben 3 registros de 32 bits del IPCORE: 43C00004, 43C00008 y 43C0000C. En cada uno de estos se guardan 4 muestras de la canción codificadas en un byte PCM (Pulse Code Modulation) sin signo.

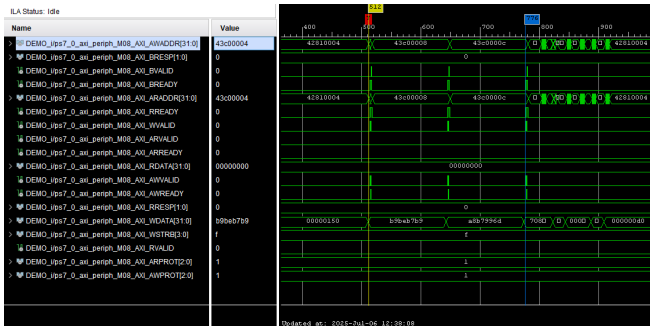

FIGURE 2: Transacción AXI entre el PS y el IPCORE axi_buzzer

Para comprobar el funcionamiento tanto de las interrupciones como del flujo de la máquina de estados del juego, se revisaron los valores de la estructura que guarda el historial de estados utilizando el debug de Vitis.

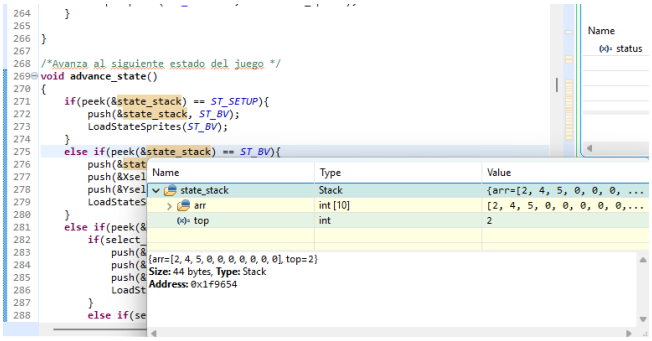

FIGURE 3: Stack de estados previo a la interrupción

En la figura 3 se muestra el "stack" que almacena los estados del juego cuando este se encuentra en el menú principal. Utilizar este tipo de estructura permite avanzar y retroceder en los menús de manera sencilla, utilizando una función que coloca un nuevo estado o retira el último en el "stack".

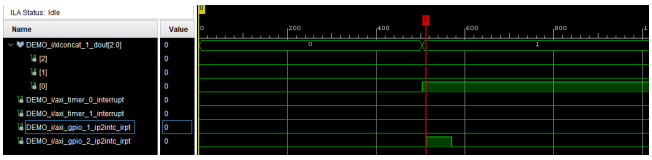

FIGURE 4: Captura de la interrupción del botón

Luego, el manejador de la interrupción del botón se encarga de ejecutar la función que avanza de estado, colocando un nuevo valor en el "stack" de estados según el estado actual. En la figura 4 se captura la interrupción del botón y se puede apreciar cuál de los 3 botones fue apretado.

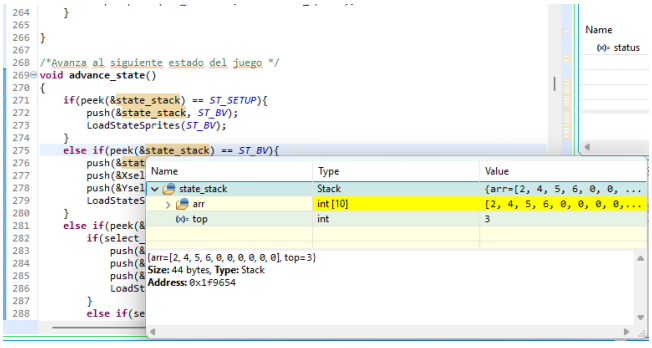

FIGURE 5: Stack de estados después de la interrupción

Por último, el estado final del "stack" de estados se puede ver en la figura 5. El debugger de Vitis destaca que la estructura ha sido modificada recientemente y se puede apreciar que el estado 6 ha sido añadido al arreglo y que el indicador de tope del "stack" se ha aumentado en 1.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Se pudo implementar el flujo del juego completo, de principio a fin y completando todas las actividades. Se utilizó un uso intensivo de la memoria SD tanto para la música como para la pantalla, esto introdujo distintos problemas que pudieron ser solucionados. Por ejemplo gracias al debuggeo se descubrió que el procesador es incapaz de leer archivos con un nombre de más de 8 caracteres, además como la lectura de música y archivos de imágenes se realizó simultáneamente se tuvo que implementar un sistema para que se detuviera la reproducción de música por un instante mientras se cargan las imágenes de la tarjeta SD a la memoria del procesador y así evitar colisiones de lectura. Sin embargo, un problema que no pudo ser solucionado completamente tuvo que ver con la generación del archivo de booteo. Originalmente, se programaron ambos núcleos del procesador, lo que permitía delegar la escritura de la pantalla al procesador secundario, lo que resultaba en una reproducción continua de la música del juego cuando la pantalla estaba siendo escrita. Más no se pudo crear una imagen de booteo capaz de cargar los dos procesadores, por lo que se adaptó el programa para funcionar en un solo núcleo.

V. CONCLUSIONES(0.2)

- Se pudo crear un IPCORE que 12 bytes de muestras de audio por medio de AXI-LITE y que posteriormente las lee a 8 kHz y las envía como PWM al buzzer de la tarjeta booster, reproduciendo un total de 12 archivos de audio distintos.
- Se implementó una máquina de estados que administra el flujo del juego a través de 17 estados distintos en total
- Los datos del juego se generaron empleando inputs del usuario por comunicación serial y generación aleatoria para determinar los nombres y estadísticas de los zybomones respectivamente. Estos fueron guardados en la memoria del procesador mediante estructuras personalizadas
- Se pudo ahorrar un total de 8.18 MB de la memoria del procesador utilizando la memoria SD para almacenar archivos de audio e imagen.

VI. TRABAJOS FUTUROS (0.2)

Se espera que de seguir trabajando en el proyecto a futuro se logre:

- Investigar en profundidad el mecanismo de BOOT de la Zybo Z7 y generar una imagen de booteo capaz de cargar correctamente el código en ambos núcleos del procesador.
- Agregar un sistema de reproducción de animaciones para los ataques, lo que involucraría modificar la máquina de estados principal, añadiendo una forma de monitorizar el tiempo transcurrido en cada cuadro de la animación para generar fluidez.
- Aumentar la cantidad de tipos, música y "sprites".
- Añadir acciones adicionales cada turno, además de atacar, como defender o aumentar estadísticas. Para eso habría que añadir los estados correspondientes al flujo del juego junto con funciones que manejen internamente lo que ocurre con los datos del juego.

REFERENCES

[1] Implement a stack in C programming. (2024, mayo 5). GeeksforGeeks. https://www.geeksforgeeks.org/c/implement-stack-in-c/