Diseño y simulación de un videojuego de batalla naval en la Zybo Z7 utilizando Vivado

RODRIGO POZO[1], CRISTÓBAL VASQUEZ[1]
[1]Pontificia Universidad Católica de Chile (e-mail: rfpozo@uc.cl, cristobal.vasquez@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 Este estudio se enfocó en la implementación del juego de batalla naval para dos jugadores mediante el uso de la tarjeta de desarrollo Zybo Z7 y el software Vivado. Para lograr este objetivo, se utilizó el lenguaje de programación VHDL y se diseñó un hardware personalizado para controlar los elementos del juego.

Entre las funcionalidades implementadas, se destaca una fase de ubicación de los barcos en el tablero por parte de cada jugador, una fase de ataques con su comprobación respectiva y una fase final donde se indica el ganador. Adicionalmente, se desarrolló una interfaz de usuario intuitiva que permitió a los jugadores interactuar con el juego y realizar sus movimientos mediante la utilización de los 4 *switches* y botones de la tarjeta Zybo Z7.

Los resultados obtenidos demostraron la efectividad de la implementación realizada, evidenciando la capacidad de la tarjeta Zybo Z7 y del software Vivado para la creación de aplicaciones de hardware personalizadas. En este sentido, se realizaron pruebas exhaustivas para garantizar la estabilidad del juego y su correcto funcionamiento en distintos escenarios.

En conclusión, todos los bloques implementados permitieron obtener los resultados esperados en tu implementación. A modo general, se pudo guardar la posición de los barcos elegidos de cada jugador, se pudo escoger y atacar posiciones elegidas por el usuario y se detecta correctamente al ganador en cada partida.

INDEX TERMS Zybo Z7, Vivado, VHDL, batalla naval, juego, tarjeta de desarrollo, diseño de circuitos digitales, sistemas embebidos, descripción de hardware, simulación de circuitos, interfaz de usuario, programación, electrónica, tecnología.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

La figura 1 muestra los bloques funcionales principales del proyecto. Dichos bloques se detallan a continuación:

1) El bloque `FASE_DE_UBICACION` tiene la finalidad de controlar el flujo de la fase de ubicación de los barcos. Para esto, se le pide a cada uno de los jugadores en su turno utilizar los *switches* para elegir una posición del mapa en la que se desea colocar un barco. Una vez elegida la posición, utilizando el botón `CONFIRMAR_BTN` se confirma la colocación del barco.
   A medida que se van posicionando los barcos, el valor de `POS_BARCOi_Jk` pasa de 0 a 1 para cada uno de los i barcos del jugador k.
   `LED_ELECCION`, `TURNO_ELECCION` y `TERMINO` son *flags* que entregan información a los módulos que controlan los LEDs para que se enciendan las luces correspondientes a cada caso. En particular, `TERMINO` da paso a que el módulo `FASE_DE_ATAQUE` comience su lógica.
2) `FASE_DE_ATAQUE` es un módulo que permite controlar la lógica de los ataques de cada jugador. Para esto recibe los bits `POS_BARCOi_Jk` para conocer las posición de los barcos del jugador k para compararlos con las posiciones atacadas por el jugador contrario. Para atacar, se utilizan los *switches* de la tarjeta de

prototipado (`SW[3:0]`) para seleccionar la posición y el botón `ATACAR_BTN` para confirmar el ataque. Por otro lado, se da la posibilidad de pasar turno al presionar el botón `PASAR_TURNO_BTN`.
Con respecto a los *outputs* del bloque, se encuentran los bits `BARCOi_Jk_DESTRUIDO` que cambian su valor de 0 a 1 si es que el ataque al barco i del jugador k es efectivo. Posteriormente estos bits son usados para comprobar las condiciones de victoria en el bloque `VERIFICAR_CONDICIONES_DE_VICTORIA`.
Finalmente, al igual que el bloque anterior `TURNO_ATAQUE`, `ATAQUE_LED_RGB` y `ATAQUE_LED` son *flags* que determinan el comportamiento de los LEDs.

3) `VERIFICAR_CONDICIONES_DE_VICTORIA` recibe los bits `BARCOi_Jk_DESTRUIDO` para y comprueba si todos los i barcos de algún jugador k están destruidos. Si esto es así, el bit `VICTORIA` cambia de 0 a 1 para indicar esto. Por otro lado, `LED_VICTORIA` aporta información a los LEDs para que se comporten en función de la ocasión.
4) Los módulos `DRIVER_LED_RGB` y `DRIVER_LED` reciben las *flags* comentadas anteriormente para variar en tiempo real el LED RGB y los LEDs, respectivamente, según las decisiones y etapas de la partida.

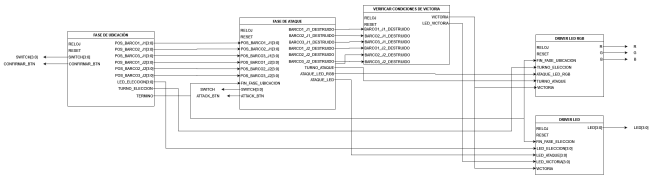

FIGURE 1: Diagrama de bloques de la lógica simplificado de la lógica programada. Se obvio la conexión de y RESET

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

1) AO1
   a) Descripción: Se utilizaron *components* tanto para el *debouncer* como para *AXI Lite* y *AXI Full*, cumpliendo con la cantidad mínima requerida.
   b) Nivel de logro: 100%
2) AO2
   a) Descripción: Se utilizaron más de tres diferentes *packages*, los cuales utilizan parámetros genéricos.
   b) Nivel de logro: 100%
3) AO3
   a) Descripción: Se utilizaron *IP cores* esclavos con un ATG maestro tanto en *Test mode* como en *Advance mode*.
   b) Nivel de logro: 100%
4) AC1
   a) Descripción: Se utilizaron máquinas de estado para distintos módulos del proyecto. Dentro de estos se encuentran `FASE_DE_UBICACION` y `FASE_DE_ATAQUE`
   b) Nivel de logro: 100%
5) AC2
   a) Descripción: Se utilizaron todos los botones, *switches* y LEDs de la tarjeta de prototipado.
   b) Nivel de logro: 100%
6) AC3
   a) Descripción: Cumplimos con los requisitos mínimos de operadores, pero no de atributos.
   b) Nivel de logro: 50%
7) AC4
   a) Descripción: Se utilizaron variables para la lógica del contador del `blinking_led`
   b) Nivel de logro: 100%
8) AC5
   a) Descripción: Se utilizó código secuencial en los módulos que se implementó alguna máquina de estado (por ejemplo `FASE_DE_UBICACION` y `FASE_DE_ATAQUE`). Por otro lado, se utilizó código concurrente en `VERIFICAR_CONDICIONES_DE`
   b) Nivel de logro: 100%
9) AC6
   a) Descripción: No se utilizó ni *functions* ni *procedures*.
   b) Nivel de logro: 0%
10) AC7
   a) Descripción: Algunas de las funciones adicionales implementadas son la utilización del LED RGB, `UTIL VECTOR LOGIC` y `AXI SMART CONNECT`.
   b) Nivel de logro: 100%

III. RESULTADOS DE SIMULACIÓN (3 PUNTOS)

En las simulaciones de comportamiento no se ve reflejado el *delay* en la transmisión de señales. Es por esto que se utilizará este tipo de simulación como punto de comparación para determinar la diferencia en la actualización de señales de la simulación de implementación. Las figuras 3 y 5 muestran la simulación de comportamiento y de implementación del módulo `FASE_DE_UBICACION`, respectivamente. Si se realiza un *zoom* al instante en el que se presiona el `btn_elegir` se obtienen las imágenes 4 y 6. Como es posible notar, en la simulación de implementación se introduce un retardo de aproximadamente $7nS$ en la señal `POS_BARCO1_J1[3:0]` con respecto a la simulación de comportamiento. Algo similar ocurre con la señal `led[3:0]` ya que se retrasa por $6nS$ aproximadamente.
Por otro lado, respecto a las simulaciones de AXI, en la figura 2 se puede observar los resultados de simulación de comportamiento de la transacción de AXI Lite. En ella se observa que la señal

`s00_axi_wdata[31:0]` muestra los datos enviados en la transacción. Para este caso, se hizo el envío de 4 datos de 32 bits cada uno. La señal `s00_axi_awvalid` se levanta cada vez que se comienza el envío de un dato. La señal `s00_axi_awadrr[3:0]` representa la dirección en donde se escribe el respectivo mensaje. El resto de señales que se muestran representan el resto de los handshake respectivos que ocurren durante la transacción de AXI LITE.

Por otra parte, si bien la transacción de de AXI FULL fue realizada con sus respectivos ATGs y Ram con puerto esclavo AXI FULL, no fue posible realizar la simulación respectiva debido problemas en la planificación utilizada.

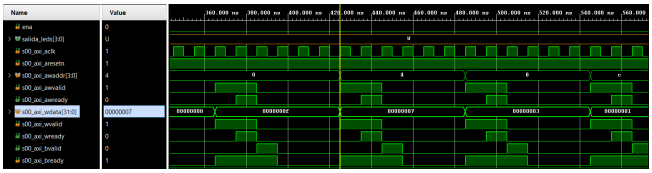

FIGURE 2: Simulación de comportamiento transacción AXI LITE

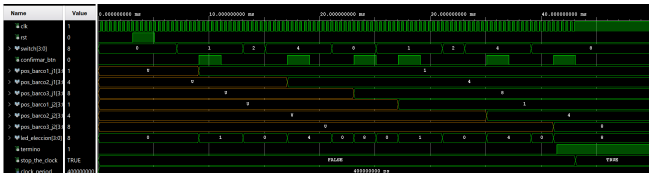

FIGURE 3: Simulación de comportamiento del módulo FASE DE UBICACION

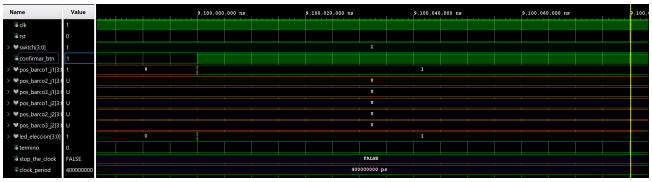

FIGURE 4: Simulación de comportamiento del módulo FASE DE UBICACION con un *zoom* en el primer flanco de btn elegir

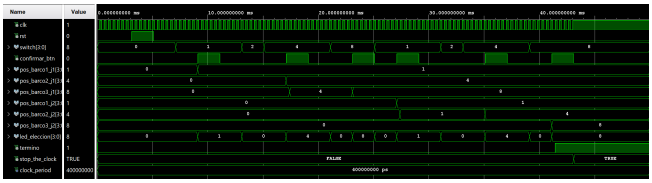

FIGURE 5: Simulación de implementación del módulo FASE DE UBICACION

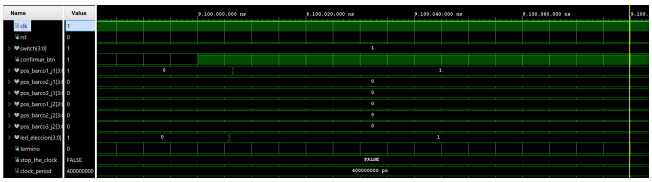

FIGURE 6: Simulación de implementación del módulo FASE DE UBICACION con un *zoom* en el primer flanco de btn elegir

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

En el marco del presente informe, se hace mención de dos problemas que surgieron en el contexto del modelación del proyecto. En un principio, se tenía contemplado utilizar una memoria RAM para el almacenamiento y consulta de la ubicación de los barcos correspondientes a cada jugador. Sin embargo, durante el proceso de implementación se pudo constatar la existencia de una alternativa más sencilla, consistente en la utilización de bits para cada uno de los barcos en cuestión. Esta solución permitió reducir considerablemente la complejidad del proyecto, lo que a su vez mejoró su mantenibilidad y legibilidad.

Otro problema que se presentó fue el relacionado con la desincronización de señales. Este inconveniente se debió a la utilización inicial de módulos no sincronizados con la señal de *clock*, lo que generaba un desfase en la llegada de las señales a su destino y, por ende, el incorrecto funcionamiento de la lógica. Una vez implementada la lógica, se constató que, en la simulación, las señales llegaban desfasadas debido a ciertas imperfecciones de los conductores. Para resolver este problema fue necesario segmentar la lógica en diferentes procesos y modificar el orden de ejecución de cada pieza de código.

V. CONCLUSIONES(0.2)

Enumere las conclusiones más importante de su proyecto. Recuerde que:

- Durante la `FASE_DE_UBICACION`, se esperaba que cada jugador pudiera elegir 3 barcos, separando todas las elecciones con una entrada de botón. Lo obtenido fue una correcta separación de estados, mediante presión de boton. El programa logra guardar en las señales de salida la posición de cada barco definida por la respectiva combinación de switch definida por el usuario.
- Durante la `FASE_DE_ATAQUE`, se esperaba que el programa recordara la posición escogida por un jugador para atacar y revisara en los registros de barcos de cada jugador si es que la posición coincidía con la de un objetivo. En la practica se obtuvo que cuando un jugador seleccionaba una posición para atacar, el programa mostraba una señal

(LED) que indicaba si el ataque había impactado un blanco. La cantidad de blancos impactados se guardó para cada jugador correctamente y se fue sumando mediante este aumentaba.

- Para la fase de victoria (constituida por los módulos `DRIVER LED RGB` y `DRIVER LED` y `VERIFICAR CONDICION DE VICTORIA`) se esperaba que a partir de las señales del módulo `BARCOi_Jk_DESTRUIDO` esta sea capaz de gatillar una *flag* `VICTORIA` que introduzca la lógica a un estado el cual se anunciara el fin de la partida a través de la luz RGB. Por otro lado, en esta fase también se esperaba que a través de la botonera de la tarjeta de prototipado se pudiera conducir una "celebracion" consistente en hacer parpadear los 4 LEDs de la Zybo Z7.
  En términos prácticos, todos los resultados cumplieron con todos los puntos esperados comentados anteriormente.
- Para la transacción AXI LITE se esperaba que los datos ubicados en el archivo `data.coe` en las direcciones especificadas en `addres.coe` de la RAM con puerto AXI LITE slave creada. En la práctica resulto que se guardaron todos los datos esperados en los registros de la RAM.

VI. TRABAJOS FUTUROS (0.2)

En el presente artículo se reporta el desarrollo de un proyecto funcional con respecto a las funcionalidades de juego definidas anteriormente. No obstante, durante el proceso de desarrollo se han identificado ciertas funcionalidades que podrían mejorar la experiencia del usuario y que se planean añadir en futuras versiones del juego.

Entre las principales funcionalidades que se han contemplado se encuentran: la posibilidad de añadir barcos con dimensiones distintas a la de 1x1, la orientación de los barcos, la definición personalizada de la dimensión del mapa, la definición personalizada de la cantidad de barcos y la implementación de patrones de ataque que cubran más de una casilla. Cabe señalar que, si bien estas funcionalidades son deseables, se debe tener en cuenta que su implementación requiere de la entrada de señales desde un periférico externo que cuente con un mayor número de botones. Un teclado de computadora podría ser una alternativa adecuada.

Por otro lado, también se debe considerar la posibilidad de implementar todas las funcionalidades utilizando únicamente el módulo de FPGA de la placa de prototipado. Es importante mencionar que la complejidad de las funcionalidades a implementar puede ser un factor determinante para considerar la utilización del microprocesador de la Zybo Z7. En este sentido, se recomienda evaluar cuidadosaFmente el uso de ambos elementos para maximizar la eficiencia y minimizar los costos de producción.

REFERENCES

[1] FPGA4Student. (2023, 09 de mayo). VHDL code for debouncing buttons on FPGA. Recuperado el 3 de mayo de 2023, de https://www.fpga4student.com/2017/08/vhdl-code-for-debouncing-buttons-on-fpga.html.

[2] Profesor Félix Rojas. (s.f.). LAB04 [Carpeta con archivos]. Pontificia Universidad Católica de Chile.