Fecha de entrega 11 de diciembre de 2023.

Aquí SEP Baila

MAURICIO ÁLVAREZ MONTIEL [1], FELIPE MACHUCA GONZÁLEZ[1]
[1]Pontificia Universidad Católica de Chile (e-mail: mfalvarez1@uc., fcmachuca@uc.cl)

Este proyecto ha sido desarrollado bajo el curso IEE2463: Sistemas Electrónicos Programables.

ABSTRACT Aquí SEP Baila es un juego desarrollado en la tarjeta Zybo $z7\_10$ de Digilnet y la tarjeta educacional Boosterpack MKII de Texas Instrument. Inspirado en las máquinas de baile arcade, el juego desafia la rapidez y agilidad del usuario. La aplicación utiliza una pantalla LCD para mostrar flechas direccionales, un Joystick para que los usuarios indiquen la dirección correcta, y periféricos adicionales para la interacción del juego. Los aciertos se confirman con un sonido y el incremento del puntaje.
Para la implementación del juego las interrupciones de Timer son fundamentales para el manejo eficiente de eventos en tiempo real, el sensor de luz permite la emisión de un mensaje dependiendo de la exposición de luz del entorno y el uso de potenciómetros permite ajustar la velocidad y dificultad del juego. Además, la comunicación UART, I2C, SPI permiten una interacción fluida con el usuario.
EL resultado es un juego entretenido y desafiante que demuestra la aplicación efectiva de los conceptos de sistemas embebidos y programación en C y VHDL. El juego demuestra ser una herramienta efectiva para entender la programación de microprocesadores, la gestión de entradas/salidas y la manipulación de dispositivos de interfaz de usuario.

INDEX TERMS Zybo Z7-10, BoosterPack MKII, Sistemas Embebidos, Interrupciones de Timer, Sensor de Luz, Comunicación UART, Protocolo I2C, Comunicación SPI, Desarrollo de Juegos Electrónicos.

I. ARQUITECTURA DE HARDWARE Y SOFTWARE (1 PUNTO)

EL proyecto consiste en el diseño e implementación de un juego de agilidad llamada "Aquí SEP baila" en la FPGA de una placa Zybo $z7\_10$ y una tarjeta educacional Boosterpack MK II. El objetivo del juego es indicar con el Joystick la flecha mostrada en la tarjeta LCD, si se indica correctamente, se emite un sonido a través del parlante y aumenta el puntaje del usuario. La velocidad del juego y la duración se configuran a través del potenciómetro 1 y 2. Además, a través del sensor de luz es posible emitir un mensaje en la pantalla LCD indicando si hay mucha o poca luz.

Al iniciar el juego, la pantalla LCD de la tarjeta Booster muestra la imagen de inicio 'Aquí SEP Baila' y se ilumina el LED RGB en la tarjeta Zybo Z7-10. Luego, el jugador debe configurar la duración y velocidad del juego a través de los potenciómetros. Posteriormente, comienza el juego, en donde se muestran secuencialmente flechas aleatorias en la pantalla. Durante el desarrollo del juego, se observa el valor del sensor de luz, el puntaje actual y la cantidad de flechas que faltantes. Si el jugador no acierta la dirección correcta, aparece en pantalla el mensaje 'Te equivocaste'. En cambio, si acierta, se emite un sonido, cuya frecuencia y amplitud son controladas mediante los botones de la Booster, y el puntaje aumenta. Una vez se completa la duración del juego, el juego finaliza mostrando el mensaje 'FELICIDADES Tu puntaje es: (valor del puntaje) ' en pantalla.

A. HARDWARE

La arquitectura de Hardware se resume en la figura I.

EL Hardware se desarrollo en la plataforma Vivado de Xilinx. Para desarrollar el Hardware se utilizó el procesador ZYNQ7 Proccesing System, además se agregaron los siguientes IP-Cores.

- Timer 1: Me permite generar interrupciones de forma periódica cada 1 segundo para leer y escribir continuamente el valor del sensor de luz, además me permite actualizar la pantalla de juego.
- Timer 2: Me permite generar interrupciones de forma periódica cada 0.33 segundos para leer y escribir continuamente el valor del eje x e y del Joystick.
- Parlante: IP-Core propio creado a partir del buzzer del proyecto base. Este modulo permite que el par-

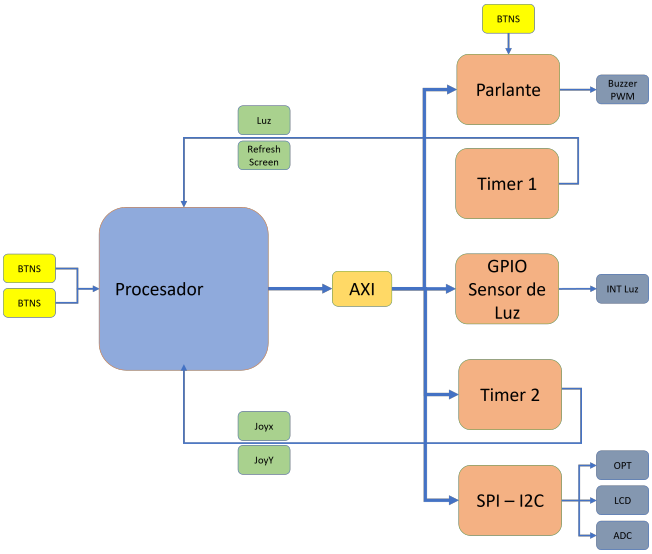

FIGURE 1: Diagrama de flujo. Hardware.

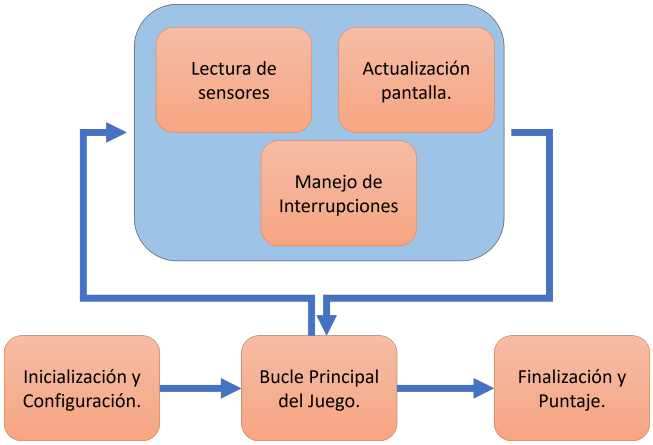

FIGURE 2: Diagrama de flujo. Software.

lante suene en caso de seguir el patrón correcto que se muestra en la pantalla actual. Tiene como entradas los botones de la booster para controlar la amplitud y frecuencia de la PWM siguiendo las frecuencias correspondientes a la escala de Do. Se comunica con el procesador para enviar 2 datos del $slv\_register$ uno que es $encendido$ (para indicar que el juego inicia y esto se debe escribir desde el computador, por lo tanto, es la parte que se comunica por UART) y otro que es $correcto$ (para indicar que la flecha actual es la correcta). Para indicar que el juego inicia se tiene una salida conectada al led RGB de la zybo que se prende con un color cyan.

- GPIO sensor de luz: Puerto de salida que permite leer la interrupción creada por el sensor de luz al sobrepasar o no alcanzar cierto umbral de luz.
- GPIO SPI, I2C: IP-Cores que permiten llevar a cabo el protocolo I2C y la comunicación SPI para conectar los periféricos de la tarjeta Booster con el procesador.

B. SOFTWARE

La arquitectura de Software se resumen en la figura II.

El software se programó en la plataforma Vitis de Xilinx.

- Inicialización y configuración: En este apartado se incluyen las librerías a utilizar, se definen los Macros y las variables. Además se inicializa, configura y establecen los punteros de los Timer, GPIO e interrupt controller.
- Bucle principal: Es un ciclo continuo que gestiona la lógica central del juego.
  - Lectura de interrupciones: Dentro del bucle, el programa está constantemente leyendo y respondiendo a las interrupciones generadas por los timers. Las interrupciones permiten leer los movimientos del Joystick y los cambios en el sensor de luz.
  - Manejo de interrupciones: Cuando se genera una interrupción, el programa ejecuta una serie de acciones. Si la interrupción proviene del Timer 1, se actualiza el estado de la luz, y si es necesario se cambia la dirección de la flecha mostrada en la pantalla. Este cambio ocurre en intervalos definidos en la configuración del juego. El Timer 2 permite determinar si el jugador ha respondido correctamente a la flecha mostrada. Además, se verifica si se ha alcanzado el término del juego basándose en la cantidad de flechas mostradas.
  - Actualización de pantalla: La pantalla LCD se actualiza continuamente para refleja el estado actual del jeugo. Esto incluye mostrar la flecha actual que el jugador debe seguir, el puntaje actual, el número de flechas restantes, y el estado de la luz (alta o baja). Si el jugador acierta la dirección de la flecha se incremente el puntaje y se muestra una confirmación auditiva a través del parlante y visual a través del LED RGB. En caso contrario, si el jugador falla, se muestra un mensaje de error.
- Finalización y Puntaje: Al finalizar la secuencia de flechas, se muestra un mensaje de felicitaciones y una pantalla con el recuento final del puntaje.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

AO1 (100%): *Descripción:* El código cumple con al utilización de la pantalla de la tarjeta booster para mostrar información del juego, imagen de inició, figuras de ejecución y término del juego. Además, se utilizaron 5 periféricos adicionales, el joystick, el sensor de luz, los potenciómetros y el parlante. La comunicación con la pantalla se realiza en el Hardware a través del GPIO 0. La interacción con los periféricos se ejecuta a través de comunicación I2C y SPI. *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%): *Descripción:* El código cumple con la uti-

lización de dos Timers, los cuales permiten leer continuamente el valor del sensor de Luz cada 1 segundo y cada 0.3 segundos el valor de la posición del Joystick.

```
 Luz CADA 1 SEGUNDO
10425
 Eje X e Y CADA 0.3 SEGUNDOS
775
1
 Eje X e Y CADA 0.3 SEGUNDOS
1023
463
 Eje X e Y CADA 0.3 SEGUNDOS
900
90
 Luz CADA 1 SEGUNDO
10409
 Eje X e Y CADA 0.3 SEGUNDOS
767
```

FIGURE 3: Ejecución de Timers.

Se utiliza la interrupción generada por el sensor de luz. Cuando el nivel de luz detectado por el sensor supera el umbral de 27076, o bien, cuando esta por debajo de 10692, el juego responde mostrando un mensaje específico en la pantalla LCD. Esta funcionalidad se logra a través de una modificación en la función $'read\_opt()'$ del archivo I2C.c.

```
int read_opt(){
    u8 config[3] = {0x01, 0xc4, 0x08};   /*  c4 es 11000100 y 0x08 es 00001000  */
    u8 high[3] = {0x03, 0x69, 0xc4};   /*  0110_ 1001 y 1100_0100 = 27076 */
    u8 low[3] = {0x02, 0x29, 0xc4};      /* 0010_1001 y 1100_0100 = 10692 */
    if (Start == 1){
        XIic_Send(iic.BaseAddress,OPT_ADDR,(u8 *)&config, 3, XIIC_STOP);
        XIic_Send(iic.BaseAddress,OPT_ADDR,(u8 *)&high, 3, XIIC_STOP);
        XIic_Send(iic.BaseAddress,OPT_ADDR,(u8 *)&low, 3, XIIC_STOP);
    }

    SendBuffer[0] = 0x00;
    XIic_Send(iic.BaseAddress,OPT_ADDR,(u8 *)&SendBuffer, 1, XIIC_REPEATED_START);
    XIic_Recv(iic.BaseAddress,OPT_ADDR,(u8 *)&RecvBuffer, 2, XIIC_STOP);

    Lux = (int)((RecvBuffer[0])*256 + (RecvBuffer[1]));

    return Lux;
}
```

FIGURE 4: Función $read\_opt()$.

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 (100%): *Descripción:* En esta actividad se genera un nuevo IP core llamado "Sonido Correcto" que interactúa con los siguientes periféricos: Botón0, Botón1, botón del Joystick y el parlante. Tiene como entradas los 3 primeros, aparte del clock de la zybo y como salidas la PWM que controla el parlante y los leds RGB de la zybo. Interactúa con el procesador a través del puerto axi. El bloque se presenta en la siguiente figura:

Dentro del ip se puede observar la lógica dentro del archivo $Sonidocorrecto\_v1\_0\_S00\_AXI$. Para indicar que se inicia el juego se manda la señal "encendido" por AXI a través

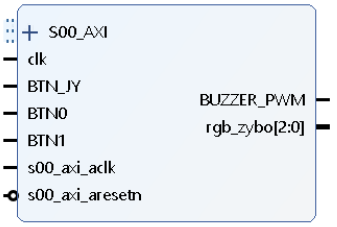

FIGURE 5: IPcore Sonido Correcto

del slv_reg0. Lo anterior se manda cuando se pide al usuario que ingrese el número 1 a través de la consola del PC, es decir, este dato es enviado a través de UART. El extracto que representa el funcionamiento de esto se encuentra acá:

```
xil_printf("Ingrese un 1 si quiere iniciar el juego: \n");
scanf("%d",&numero);
int volatile comparador1 = numero;
Xil_Out32(XPAR_SONIDOCORRECTO_0_S00_AXI_BASEADDR,comparador1);
```

Para indicar que lo anterior funciona correctamente se decidió prender el led RGB de la Zybo.

De forma similar se envía por AXI la señal "correcto" a través del slv_reg1. Es importante destacar que el largo de estos datos es de 32 bits y se pasan a unsigned para tener sentido dentro de la lógica.

El resto del código es similar al buzzer original pero en este caso se dejan frecuencias que indican las notas DO, RE,MI, FA, SOL, LA, SI , DO según la cantidad de veces que se presiona el botón 1 y la amplitud se controla con las veces que se ha presionado el botón 0. Finalmente, el contador que controla la PWM del parlante se controla por la señal correcto, en caso de ser mayor a 0 se reinicia la cuenta.

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: No se creó una máquina de estados asociada al funcionamiento del proyecto. (0%):

AC2: En el código se utiliza la estuctura a través de 'typedef struct ConfigRango', la cual me permite establecer los valores del potenciómetro para configurar la duración y dificultad del juego. Se utiliza el arreglo 'FuncionDibujo' con las opciones de flecha a dibujar en la pantalla. Se definen funcionen como 'my_interrupt_Timer'. Se definen Macros para facilitar el acceso a la dirección de registros. Dentro del código se utilizan punteros para acceder y manipular registros de hardware, además XTmrCtr, XScuGic, XGpio, y XSpi son estructuras que contienen información sobre los controladores de temporizador, controlador de interrupción, GPIO y SPI, respectivamente. (100%)

AC3: En total el código utiliza 6 periféricos, la pantalla LED de la booster, el senosr de luz, Joystick, potenciometro 1, potenciometro 2 y parlante. (100%).

AC4: Se realiza el Booteo de la Zybo, evidencia se muestra en el video asociado al proyecto. (100%).

AC5: Para realizar un seguimiento eficiente a las señales se agregaron los IP Core de ILA y Vivado. (100%).

AC6: El Buzzer interactúa directamente con la ejecución del código, cuando el usuario mueve el Joystick en la dirección correcta se emite un sonido en el Buzzer.(100%).

AC7: No se utiliza la memoria SD. (0%)

III. RESULTADOS (3 PUNTOS)

Con el objetivo de capturar una interrupción se realizó un debug en Vitis. Se seleccionó como base la interrupción generada por el timer1 que ocurre cada 1 segundo. Bajo este método lo primero es seleccionar en $SW\_base\_boster\_system$ la opción Debug as y luego Launch Hardware. Se decidió poner algunos breakpoints en el handler de ese timer y además seguir la expresión opt que es la cual indica el valor de luz medido cada vez que el timer llega a la cuenta fijada. En los siguientes pantallazos se observa el funcionamiento del debug:

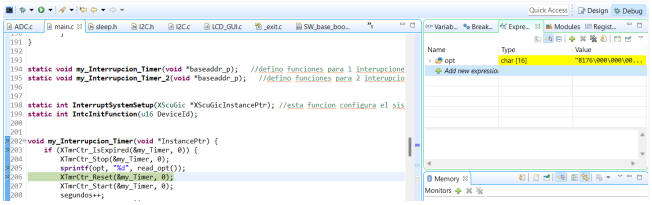

FIGURE 6: Debug 1

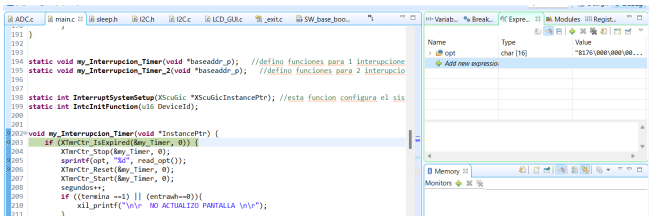

FIGURE 7: Debug 2

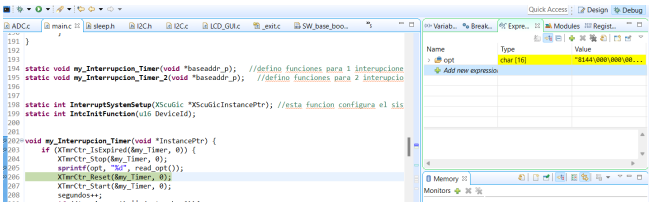

FIGURE 8: Debug 3

A partir de lo anterior, se puede concluir que el handler del timer efectivamente está funcionando ya que mide el valor del sensor de luz con $read\_opt$ y da distintos valores en las diferentes lecturas, ya que en la primera se obtiene 8176 y en la segunda 8144. También en la Figura 3 se observa el comportamiento de los 2 timers en conjunto al hacer un xilprintf cada vez que se ejecutan, es decir, cuando llegan a sus cuentas y se resetan.

Para el debugeo de hardware de forma simultánea a la ejecución de códigos en el procesador se utilizó LightIRQ la cual se encuentra conectada a un GPIO como input. Esta señal en caso de encontrarse fuera de los límites establecidos al configurar el sensor de luz toma el valor de 1 si sobrepasa por arriba y 0 si sobrepasa por abajo.

Para esto en Vivado se conectó el bloque ILA a la misma salida del GPIO que a su vez va conectado al constrain correspondiente U15. Luego como estamos utilizando el procesador ZYNQ se agregó también un bloque BRAM Controller. Posteriormente en Vitis se corre el hardware-software y se obtiene un archivo de extensión elf que se carga en Vivado. Después de agregar el elf a vivado, se corre el bitstream nuevamente y al abrir el hardware manager podemos debuggear en tiempo real. Los resultados obtenidos se resumen en las siguientes imágenes:

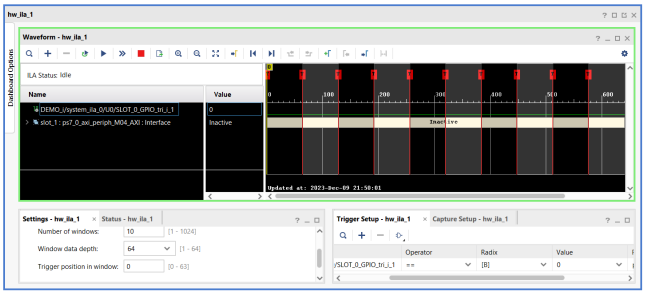

FIGURE 9: Debugg sensor de luz 0

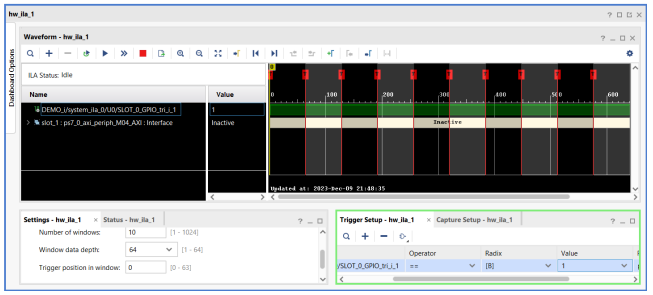

FIGURE 10: Debugg sensor de luz 1

Se logró identificar que efectivamente los valores de LightIRQ cambiaban entre 1 y 0 dependiendo de las condiciones de luminosidad aplicadas al aplicar como trigger .

Para probar el IP-core como este recibía datos desde la booster también se llevó a cabo un procedimiento similar al anterior. En este caso se colocó un ILA a la salida de los leds rgb de la zybo y se cargó el elf. Se incluyó como trigger cuando esta señal es distinta de 0 y efectivamente funcionaba al correr el programa ya que pasaba de 000 a 110 que forma el color cyan visto en la tarjeta. Un pantallazo del momento se muestra en la siguiente imagen:

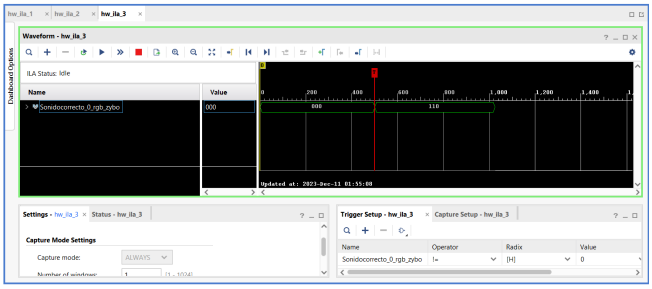

FIGURE 11: Debug IP-core con led RGB

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

La implementación de la lógica general del juego es correcta ya que el tiempo en que cambian las flechas es adecuada según el valor de velocidad dado por el potenciómetro y suena cuando se mueve en la dirección correcta además de aumentar el score. Se intentó poner el potenciómetro que controla el largo de la secuencia dentro del mismo struct pero no se lograban los valores deseados por lo que se decidió dejar este potenciómetro con lógica de IF.

Una dificultad que no se pudo resolver fue que cuando la velocidad de cambio era de 1 segundos no siempre toma el valor correcto del joystick porque efectivamente se mueve pero el score no alcanza a aumentar. Debido a esto la velocidad más rápida se decidió dejar en 2 segundos.

V. CONCLUSIONES(0.2)

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

- Funcionamiento del Handler del Timer: El handler del timer 1, configurado para generar interrupciones cada 1 segundo, está operando según lo esperado. Durante el proceso de debuggeo, se pudo observar que el valor del sensor de luz se mide correctamente utilizando la función $'read\_opt()'$ en cada interrupción del timer. Además, se pudo verificar mediante el uso de breakpoints en el handler y el seguimiento de la variable 'opt' los cambios de luminosidad en el sensor.
- Interrupciones por cambios de luz: Utilizando el GPIO configurado como entrada para la señal Light IRQ se confirma que la señal varia entre 1 y 0 de acuerdo con las condiciones de luz. Se estableció que la interrupción toma el valor de 1 cuando la luminosidad excede el umbral superior establecido y 0 cuando cae por debajo del umbral inferior. Lo cual se logró validadar mediante el uso del bloque integrated logic analizer (ILA) en Vivado, conectado a la salida del GPIO y configurando la señal como trigger.
- Depuración en Vivado y Vitis: El proceso de debuggeo implicó la conexión del bloque ILA y una BRAM controller, luego de ejecutar el programa en Vitis y la carga del archivo .elf en Vivado, al correr el bitstream y abrir el hardware manager se logró depurar en tiempo real, lo que permitió observar el comportamiento dinámico de las señales y validad el funcionamiento del sistema.

VI. TRABAJOS FUTUROS (0.2)

Para futuras versiones del proyecto, hay diferenetes oportunidades interesantes de desarrollo y mejora que podrían enriquecer la experiencia de juego.

- Selección de música personalizada: Integrar una función que permita a los jugadores elegir su porpia música para bailar. Esto podría incluir una biblioteca de canciones predefinida
- Mejora en la interfaz de usuario: Desarrollar una interfaz de usuario más avanzada, que incluya la progresividad de la canción, animaciones, una pantalla de menú, entre otros.
- Modos de juego Múltiples: Implementar diferentes modos de juego, como niveles con aumentos progresivos de dificultad, o modos de desafío con patrones de flechas más complicados.
- Modo Multijugador: Desarrollar una funcionalidad para que varios jugadores puedan competir o cooperar en el mismo juego, lo que podría implicar la comunicación entre varias tarjetas Zybo Z7-10 o jugar en linea.

REFERENCES

[1] Cristobal Sebastian Vasquez Rosel. [Felix Rojas] (12/10). AYUD 06 Xilin y XilOut [Video]. Youtube. https://www.youtube.com/watch?v=iRbIIK3E60Y&list=PLACEUah7BCQvPCO8X1JKmvREXUQkfqjQA&index=6&t=3s.

[2] Catalina Pía Sierra Simon. [Felix Rojas] (18/10). AYUD 07 AXI Timer Interrupts [Video]. Youtube. https://www.youtube.com/watch?v=eUUw3ANzMpg&list=PLACEUah7BCQvPCO8X1JKmvREXUQkfqjQA&index=5.

[3] Catalina Pía Sierra Simon. [Felix Rojas] (25/10). AYUD 08 Boot y Debug [Video]. Youtube. https://www.youtube.com/watch?v=HEcVxrbAIDQ&list=PLACEUah7BCQvPCO8X1JKmvREXUQkfqjQA&index=4&t=1036s.

[4] Agustín Pablo Kamke Mardones. [Felix Rojas] (3/11). AYUD 09 UART [Video]. Youtube. https://www.youtube.com/watch?v=3dngzqBcg&list=PLACEUah7BCQvPCO8X1JKmvREXUQkfqjQA&index=3.

[5] Raimundo Alcalde Cornejo. [Felix Rojas] (9/11). AYUD 10 I2C Buzzer [Video]. Youtube. https://www.youtube.com/watch?v=Ua9ko6uNEl4&list=PLACEUah7BCQvPCO8X1JKmvREXUQkfqjQA&index=2.