

Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias

Unidad de Formación:
TE2002B **Diseño con Lógica Programable**

Etapa #3
Entrega Final

Profesores : Enrique González Guerrero y Raúl Peña Ortega
Grupo: 402

Equipo y porcentaje de participación:
Alondra Caspeta Juárez **A01571416** 100%
Ximena Lizeth Trejo Lavín **A01198557** 100%
Angelica Nahomi Velazquez Gaytán **A00838198** 100%
Valeria Meneses Parra **A01735686** 100%

Confirmamos que somos los autores de este trabajo y que es la versión final. Se citaron debidamente las palabras o ideas de otras personas, expresando estas de forma escrita, oral o visual.

Monterrey, Nuevo León, México, **03 de mayo de 2024**

Índice

Sección	Página(s)
I. 1. Introducción	5
II. Justificación del problema	6
III. Metodología	7
- Figura 1. Implementación del prototipo	
IV. Juego	8 - 10
A. Objetivo	
B. Explicación de mundos/escenas	
- Figura 2. Printscreen de nivel uno (verano).	
- Figura 3. Printscreen de nivel dos (otoño).	
- Figura 4. Printscreen de nivel tres (invierno).	
C. Personajes	
- Figura 5. Personaje principal (tractor).	
D. Reglas del juego	
E. Dinámica	
V. Máquina de Estados (HLSM)	11 - 13
- Figura 6. Representación de estados para la secuencia del videojuego.	
A. Indicación del estado inicial	
B. Entradas, salidas y variables en formato bit y multi-bit	
- Figura 7. Inputs, Outputs y variables de máquina de estados.	
C. Transiciones entre estados	
- Figura 8. Pantalla de subir nivel y transición entre niveles.	
- Figura 9. Pantalla de juego perdido y transición a menú.	
D. Expresiones aritméticas en la asignación de variables y salidas.	
VI. Programación de placa	14 - 19
A. Explicación de código de VHDL	
a. de10_lite	
- Figura 10. Declaración de la entidad “de10_lite”.	
- Figura 11. Componente de la entidad “uart”.	
- Figura 12. Componentes de las entidades “debounce” y “bcd7seg_sec”.	
- Figura 13. Señales internas.	
- Figura 14. Instancias de los componentes.	
- Figura 15. Cláusula “if” dentro del flanco de subida del reloj.	
- Figura 16. Cláusulas para mandar información dependiendo de switches y botones.	
b. bcd7seg_sec	
- Figura 17. Declaración de la entidad “bcd7seg_sec”.	

- Figura 18. Casos para desplegar dígitos en el display de 7 segmentos.
- B. Pruebas de funcionamiento
- Figura 19. Entidad para separar unidades y decenas para displays.
 - Figura 20. Arquitectura de separador de unidades y decenas utilizando bcd7seg_sec
 - Figura 21. Declaración de componente en entidad de10.vhd
 - Figura 22. Instanciación de componente para desplegar el número recibido.
 - Figura 23. Printscreen al inicio del juego donde hay 0 maíces recolectados.
 - Figura 24. Printscreen durante el juego donde hay 19 maíces recolectados.

VII. *Conexión serial* 20 - 26

- Tabla 1. Conexiones entre la FTDI y el FGPA.
 - Figura 25. Instanciación de SerialPort.
 - Figura 26. Método Start().
 - Figura 27. Método Update()
- A. Demostración de funcionamiento con ejemplos
- a. Prueba 1
 - Figura 28. Código VHDL para enviar un 5 al activar switch 0.
 - Figura 29. Transmisión de datos en Termite con el switch 0.
 - Figura 30. Código del movimiento hacia adelante en Unity.
 - Figura 31. Debug Log “5” al activar SW0.
 - b. Prueba 2
 - Figura 32. Código VHDL para enviar un 1 al activar KEY1.
 - Figura 33. Transmisión de datos en Termite con KEY1.
 - Figura 34. Código del aumento de velocidad en Unity.
 - Figura 35. Debug Log “1” al presionar KEY1.
 - c. Prueba 3
 - Figura 36. Código para actualizar el texto en la interfaz de usuario en Unity.
 - Figura 37. Método para enviar la cantidad de maíces desde Unity.
 - Figura 38. Debug Log en la esquina inferior izquierda.

VIII. *Procesador* 27 - 36

- A. *Explicación de código en VHDL*
- a. *de10_lite*
 - Figura 39. Entidad de componente de10_lite

- Figura 40. Componente de la UART.
- Figura 41. Componente de Gummnut.
- Figura 42. Señales declaradas en la arquitectura de de10_lite
- Figura 43. Instanciaciones de componentes en la arquitectura de de10_lite
- Figura 44. Procesos para transmisión de datos.
- b. *Display de 7 segmentos (contador de plantas)*
 - Figura 45. Proceso del DISPLAY
 - Figura 46. Función para mantener el contador actualizado
- c. *Push buttons (freno y acelerador)*
 - Figura 47. Componente debounce.
 - Figura 48. Instanciaciones para hacer debounce de botones
 - Figura 49. Asignación de datos de botones a port_dat_i dependiendo de la dirección.
 - Figura 50. Asignación de direcciones y datos de los botones.
 - Figura 51. Función para KEY0 en ensamblador.
 - Figura 52. Función para KEY1 en ensamblador.
- d. *Switches (adelante y atrás)*
 - Figura 53. Asignación de datos de switches a port_dat_i dependiendo de la dirección.
 - Figura 54. Asignación de direcciones y datos de los switches.
- e. *Acelerómetro*
 - Figura 55. Funciones de los switches en ensamblador.
 - Figura 56. Componente del acelerómetro.
 - Figura 57. Instanciación de acelerómetro, asignación de bits significativos a una segunda señal y asignación de datos del acelerómetro a los LEDs.
 - Figura 58. Asignación de datos del acelerómetro a port_dat_i de acuerdo a la dirección.
 - Figura 59. Asignación de direcciones y datos del switch habilitador y el acelerómetro.
 - Figura 60. Función de acelerómetro de lado derecho en ensamblador.
 - Figura 61. Función de acelerómetro de lado izquierdo en ensamblador.
 - i. Prueba de funcionamiento de acelerómetro
 - Figura 62. Led 09 encendido por inclinación a la izquierda.
 - Figura 63. Led 00 encendido por inclinación a la derecha.

- Figura 65. Entidades “debounce”..
- B. gumnut_with_mem.vhd*
 - Figura 66. entidad “gumnut_with_mem”.
- C. uart.vhd*
 - Figura 67. Entidad “uart”.
- D. acel.vhd*
 - Figura 68. Entidad “acel”.

<i>X. Conclusiones Individuales</i>	39 - 40
<i>XI. Videos demostrativos</i>	41
<i>XII. Referencias</i>	42
<i>XIII. Anexos</i>	43 - 57
<i>A. Etapa 2</i>	
- Anexo 1. Código de de10_lite.vhd	
- Anexo 2. Código de bcd7seg_sec.vhd	
- Anexo 3. Código de uart.vhd	
- Anexo 4. Código de debounce.vhd	
- Anexo 5. Código de de10.cs	
- Anexo 6. Código de Recio.cs	
- Anexo 7. Código de carController.cs	
- Anexo 8. Código de InventoryUI.	
- Anexo 9. Código de PlayerInventory.cs	
<i>B. Etapa 3</i>	
- Anexo 10. Código de ensamblador.	
- Anexo 11. Código de de10_lite.vhd	

I. Introducción

Una de las grandes innovaciones dentro de la industria del entretenimiento han sido los videojuegos. De acuerdo a Saravia Numa (2024), una periodista del Diario El Transmisor, “[...] han evolucionado rápidamente y se han convertido en una forma de entretenimiento popular y lucrativa en todo el mundo”. Prueba de ello es el cómo los videojuegos han creado comunidades enteras que comparten la pasión por los videojuegos, estando dispuestos a gastar dinero en estos mismos, así como accesorios.

Antes de que apareciera la realidad virtual, los videojuegos se desarrollaban en 2D y 3D. Ambos formatos han tenido éxito, con títulos remarcables como Mario Bros, el cual tuvo muchas de sus entregas en un formato 2D, antes de pasar a ofrecer secuelas en 3D. Aunque existen múltiples plataformas para poder desarrollar videojuegos profesionalmente, durante el desarrollo de este proyecto se utilizará Unity 3D.

El uso más común que se le puede dar a los videojuegos es el de comunicar historias con tonos de ciencia ficción hacia el público consumidor. Aun así, existen casos como es en el ejército de los Estados Unidos, donde se están utilizando videojuegos con detalles para entrenar a los soldados (Xataka, 2017). Por ello, en el contexto de este Reto, se estará desarrollando un videojuego con el propósito de entrenar a los trabajadores de John Deere, esto de acuerdo a los recursos que da el juego y a las condiciones que se establecen.

II. Justificación del problema

La educación a través de los videojuegos no es un tema nuevo. En un estudio dirigido por Matthew Barr, un catedrático de la Universidad de Glasgow, se utilizó Minecraft Education (una versión especial del aclamado videojuego de construcción enfocada en materias específicas, como Matemáticas y Química) para saber cómo es que el usar este videojuego (entre otros más como Team Fortress 2) ayudaba en el proceso educativo de los estudiantes (García-Bullé, S., 2019).

Tras un periodo de 8 semanas, se demostró que el grupo que jugó durante ese tiempo tuvo niveles más avanzados en rubros como la comunicación, inventiva y adaptabilidad. Esto se puede traducir al contexto de esta problemática, donde la empresa John Deere necesita de prototipos de videojuegos para poder educar a sus trabajadores sobre las distintas maneras que hay para poder garantizar un trabajo eficiente y productivo. Todo esto es con el propósito de que los trabajadores empiecen a encontrar una relación entre el cómo operan un tractor (el medio por el cual el usuario interactúa con la interfaz principal del videojuego) y los beneficios que esto trae (mayor cosecha recolectada, trabajo hecho en un menor tiempo, etc).

Al aplicar un videojuego a esta Situación Problema, se está teniendo un ambiente seguro y práctico para poder aprender y cometer errores, esto para identificar áreas de oportunidad y mejora entre la misma comunidad trabajadora. El integrar un elemento lúdico también puede llevar a la colaboración y el trabajo en equipo entre los empleados, propiciando un ambiente sano y enriquecedor.

III. Metodología

La plataforma por la cual se realizará el trabajo es Unity 3D, esto de acuerdo a los requerimientos manifestados por John Deere. Unity 3D ayuda a los usuarios a poder desarrollar videojuegos con una plataforma abierta, pudiendo adaptarlos a consolas de videojuegos, a la par de al ambiente de las computadoras.

Al mismo tiempo que se colabora con los docentes de la materia y la empresa John Deere, a lo largo de las recientes semanas se han estado estableciendo actividades a realizar dentro del Reto, mismas que se planea ejecutar dentro de plazos claros y medibles en relación con el tiempo que se dispone en la materia. Todo esto se debe alinear a las competencias y habilidades necesarias para el desempeño efectivo de la Unidad de Formación.

Las distintas etapas del desarrollo del videojuego han sido estratificadas de tal manera que todos los integrantes del equipo participen, además que mediante un consenso sobre lo que se apega más a los requerimientos manifestados por el socio formador se están decidiendo continuamente aquellas alternativas que darán a un videojuego más amigable para el usuario, y que al mismo tiempo cumple con el propósito final.

A 2 de abril de 2024, se ha avanzado en la presentación digital del videojuego sin una conexión con el FPGA (Field Programmable Gate Array), que de acuerdo a John Deere se tienen que conectar ciertos componentes que este incluye con el comportamiento del juego (véase Dinámica, en el apartado Juego). Por ello, el prototipo presentado en este Reporte es la ejecución del videojuego en la plataforma Unity 3D.

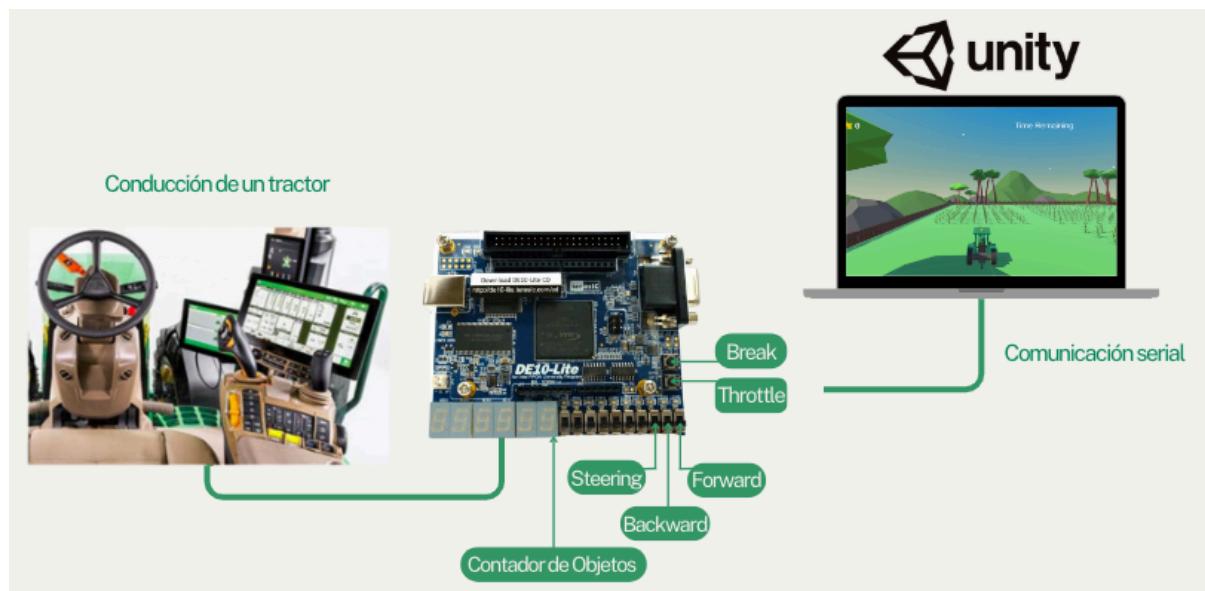


Figura 1. Implementación del prototipo

IV. Juego

IV. A. Objetivo

Los videojuegos han sido un concepto clave para la demostración del avance de la tecnología, siendo aplicados no solamente para entretenimiento, sino que también para simulaciones dentro de distintos ámbitos.

A través de la realización de este juego, se espera emular la forma de trabajo de un tractor de John Deere, esto con los requerimientos que manifestó el socio formador. Por ello, se planea proporcionar una experiencia práctica y enriquecedora, esto para que los trabajadores puedan familiarizarse con el manejo de la maquinaria agrícola. Por propósitos de identificación del videojuego, este mismo se llamará *Crop Crusade: Tractor Tales*.

El aplicar este videojuego también significa que se tienen que integrar los conocimientos teóricos dentro de John Deere, asegurando una experiencia fiel a la que se daría en el mundo real.

IV. B. Explicación de mundos/escenas

El factor de diferencia entre cada mundo es el tiempo, esto debido a que se tiene la misma cantidad de plantas, pero menor tiempo para recogerlas todas. Esto implica que el usuario tendrá que hacer uso de elementos como el acelerador, tarea que tiene que ser realizada de una forma consciente. A continuación, se detallan los mundos a desarrollar:

- *Mundo 1: Escena 1.* El jugador recoge 512 plantas por medio de un tractor, con un tiempo máximo de 40 segundos. El escenario es en verano.



Figura 2. Printscreen de nivel uno (verano).

- *Mundo 2: Escena 1.* El jugador recoge 512 plantas por medio de un tractor, con un tiempo máximo de 25 segundos. El escenario es en otoño.



Figura 3. Printscreen de nivel dos (otoño).

- *Mundo 3: Escena 1.* El jugador recoge 512 plantas por medio de un tractor, con un tiempo máximo de 15 segundos. El escenario es en invierno.



Figura 4. Printscreen de nivel tres (invierno).

IV. C. Personajes

El medio por el cual el usuario puede interactuar con la dinámica del juego es el tractor, dado que los mundos giran alrededor de este mismo. No habrá un personaje humano en el desarrollo de este videojuego debido a que no se considera algo necesario para que el jugador sepa lo que se tiene que hacer.



Figura 5. Personaje principal (tractor).

IV. D. Reglas del juego

1. El juego consta de un solo jugador (el tractor).
2. El jugador debe recolectar las plantas que se encuentran dentro del área para poder pasar de nivel. Esto es, el juego es secuencial.
3. El juego se termina si el usuario no recolecta cierto número de plantas en el tiempo límite establecido (esto depende de cada mundo).
4. Cuando el jugador pierde, siempre regresará al menú.

IV. E. Dinámica

A través del Pin Planner en Quartus Prime, se planea relacionar la información que envía cada elemento externo a la placa para afectar el comportamiento del juego. El FPGA utilizado es el DE-10 Lite. A continuación se menciona cada punto a tomar en cuenta en el apartado de Dinámica:

- El *acelerómetro*, que está enlazado con el FPGA, manifiesta la tasa de cambio en la velocidad al momento en que el tractor es conducido.
- Los switches para la selección de marcha serán los que incluye el FPGA.
- Los botones que se requieren (acelerador y freno) serán los que provee el FPGA.
- El contador de plantas recolectadas se desplegará en el display de 7 segmentos que incluye el FPGA.
- En una pantalla externa se desplegará información como el desplazamiento del tractor y la colocación de objetos.

V. Máquina de Estados (HLSM)

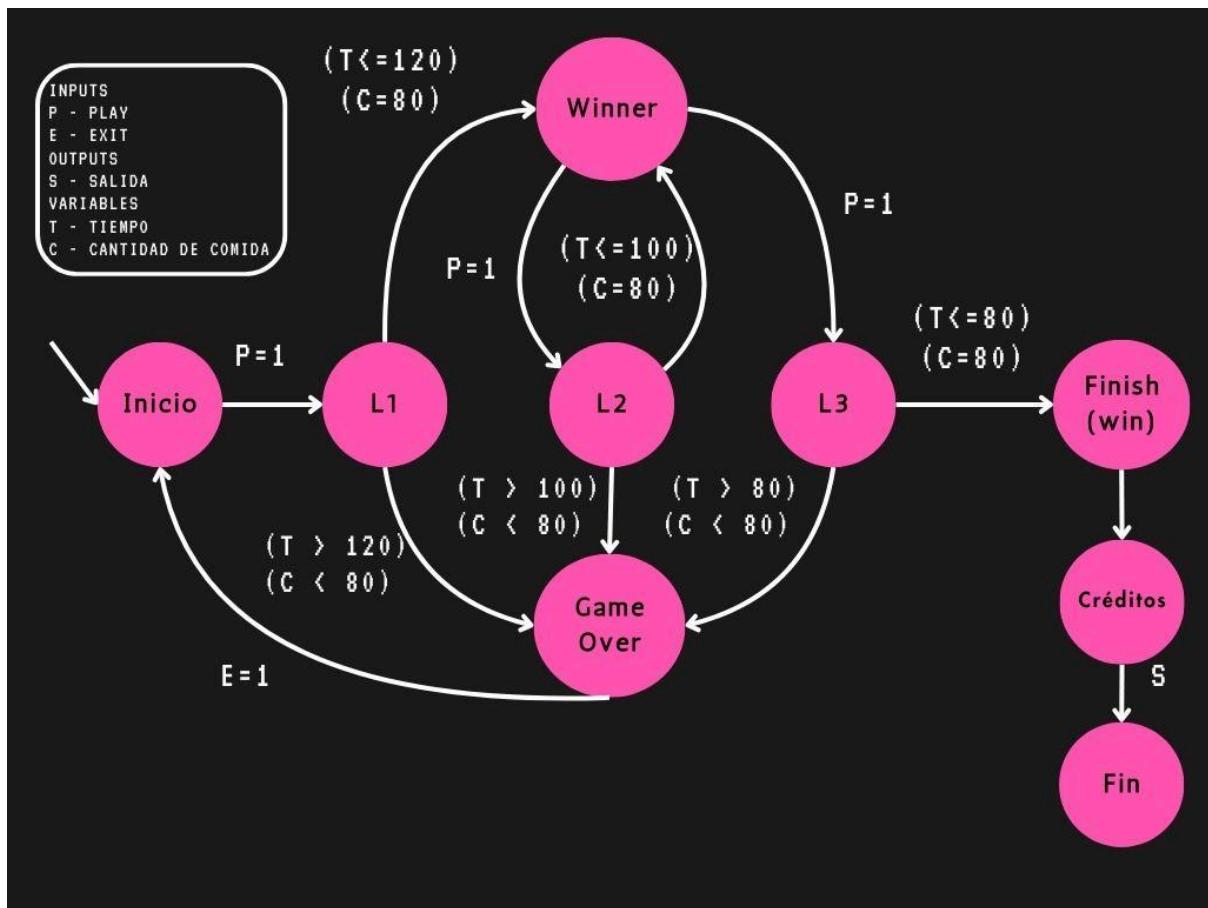


Figura 6. Representación de estados para la secuencia del videojuego.

V. A. Indicación del estado inicial

El juego empieza con el estado “**Inicio**”, y cuando el jugador desea empezar con el primer nivel la primera transición es con el input “**P**”, que significa play. Si el jugador no selecciona la opción de play, entonces este se mantendrá en el estado inicial. De ahí en adelante el jugador va saltando de nivel al completar las diferentes condiciones y reglas especificadas del juego.

V. B. Entradas, salidas y variables en formato bit y multi-bit



Figura 7. Inputs, Outputs y variables de máquina de estados.

Inputs

- P: significa play, e indica cuando el jugador pasa del estado “Inicio” al nivel uno (“L1”).
- E: exit es el input que el jugador selecciona para salir del estado “Game Over”, en caso de perder o no cumplir con alguna de las condiciones, e ir al estado “Inicio” para volver a empezar en el L1.

Outputs

- S: indica la salida y marca el fin del videojuego cuando el jugador logró completar los tres niveles del videojuego. Este output lo lleva al estado final “Fin”.

Variables

- T: es el tiempo que transcurre en cada nivel, y este va disminuyendo conforme avanza de nivel (grado de dificultad).
- C: es la comida que el jugador debe de recolectar para ir completando cada uno de los tres niveles. Esta variable está representada por un contador dentro del videojuego.

V. C. Transiciones entre estados

Para poder subir de niveles en el videojuego, el jugador debe de completar las condiciones representadas con operaciones de igualdad y operadores relacionales. Por ejemplo, la primera transición del nivel uno “**L1**” al nivel dos “**L2**” está marcada por las condiciones ($T \leq 120$) y ($C = 80$), donde para avanzar el tractor debe de recolectar una cantidad igual a 80 plantas en un límite de tiempo menor o igual a 120 segundos. Al cumplir esta condición, el juego despliega una pantalla con la frase “Winner”, y el jugador presiona el botón de “next level”. indicando el cambio de nivel.

Así, para los tres niveles la diferencia es que la T va disminuyendo, esto para agregar un grado de dificultad en cada estación.

El jugador cae en el estado “**Game Over**” si alguna de las condiciones anteriores no se cumple. Por ejemplo, ($T > 80$) y ($C < 80$) donde el tractor se tardó más de 80 segundos en recolectar 80 plantas o recolectó menos de 80. Una vez que nos encontramos en el estado Game Over, se da la opción de seleccionar Exit ($E = 1$), para que nos lleve de regreso al estado inicial y de ahí volver a comenzar (se reinicia y el progreso no se guarda, porque se vuelve a iniciar desde cero). Si $E = 0$, entonces el jugador permanece en el estado Game Over.

Por otro lado, una vez que se completa el nivel tres “**L3**”, porque se cumplieron todas las condiciones, pasamos al estado “**Finish**”, y automáticamente se manda al jugador a la pantalla de “**Créditos**”. También se despliegan los nombres de todos los que participaron en el desarrollo del videojuego. Después, con el output “**S**” pasamos directamente al último estado llamado “**Fin**”, para marcar el final del juego.

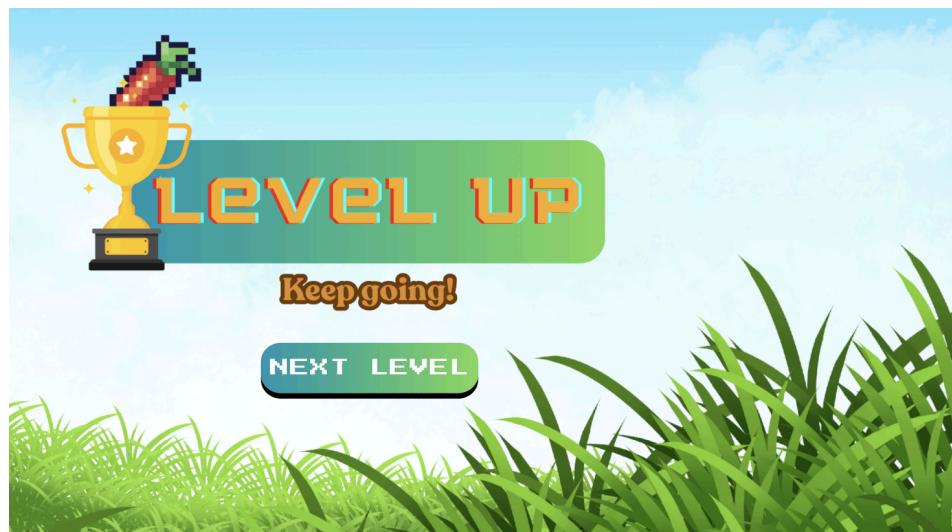


Figura 8. Pantalla de subir nivel y transición entre niveles.

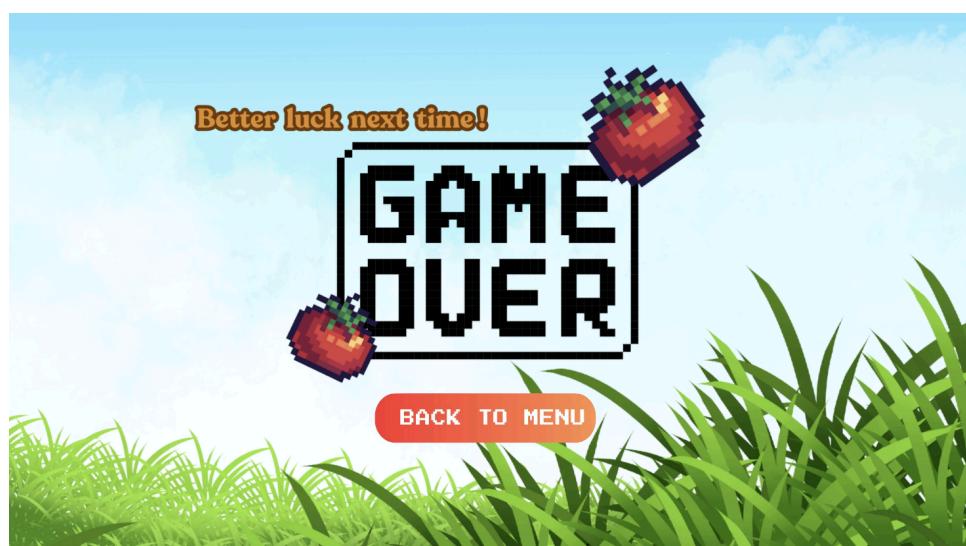


Figura 9. Pantalla de juego perdido y transición a menú.

V. D. Expresiones aritméticas en la asignación de variables y salidas

Al realizar el diseño y la máquina de estados del videojuego, concluimos que no aplica ninguna expresión aritmética en variables y salidas.

VI. Programación de placa

Para el control del juego en Unity, fue indispensable el uso del FPGA (DE10-Lite Board de Altera). Este último se programa utilizando VHDL. Debido a los requerimientos que expresó el socio formador, fue menester implementar los push buttons, los switches y el display de 7 segmentos de la placa en relación con el juego. En próximas entregas, se estará adecuando en mejor medida el funcionamiento del display de 7 segmentos y el acelerómetro.

La implementación en VHDL ayuda a crear una comunicación entre los periféricos del FPGA y el juego, debido a que (como se mencionó en apartados anteriores) existen ciertos comportamientos que deben haber dependiendo de lo que ocurra en el juego (por ejemplo, al ir obteniendo plantas el display de 7 segmentos tiene que desplegar este mismo contador).

VI. A. Explicación de código de VHDL

VI. A. a. de10_lite

En la parte inicial se están declarando como puertos de entrada y salida los elementos como el reloj interno de la placa, los switches, los GPIOs para RX y TX (con la ayuda además de un FTDI), los LEDs y el display de 7 segmentos. Después, se citan distintos componentes que se necesitan en el código: la UART, el debounce del push button y la implementación de la funcionalidad del display para mostrar el contador de las plantas que se van recogiendo (`bcd7seg_sec`).

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY de10_lite IS
  PORT(
    CLOCK_50 : IN std_logic;
    KEY      : IN std_logic_vector(1 DOWNTO 0);
    SW       : IN std_logic_vector(9 DOWNTO 0);
    GPIO_24  : IN std_logic; --RX
    GPIO_25  : OUT std_logic; --TX
    LEDR    : OUT std_logic_vector(9 DOWNTO 0);
    HEX0    : OUT std_logic_vector(7 DOWNTO 0)
  );
END;
```

Figura 10. Declaración de la entidad “`de10_lite`”.

```
ARCHITECTURE Structural OF de10_lite IS
COMPONENT uart IS
  GENERIC(
    clk_freq   : integer := 50_000_000; --frequency of system clock in Hertz
    baud_rate : integer := 115_200; --data link baud rate in bits/second
    os_rate   : integer := 16; --oversampling rate to find center of receive bits
    --(in samples per baud period)
    d_width   : integer := 8; --data bus width
    parity    : integer := 0; --0 for no parity, 1 for parity
    parity_eo : std_logic := '0'); --'0' for even, '1' for odd parity
  PORT(
    clk      : IN std_logic; --system clock
    reset_n : IN std_logic; --asynchronous reset
    tx_ena  : IN std_logic; --initiate transmission
    tx_data : IN std_logic_vector(d_width-1 DOWNTO 0); --data to transmit
    rx      : IN std_logic; --receive pin
    rx_busy : OUT std_logic; --data reception in progress, LEDR(9)
    rx_error: OUT std_logic; --start, parity, or stop bit error detected
    rx_data : OUT std_logic_vector(d_width-1 DOWNTO 0); --data received
    tx_busy : OUT std_logic; --transmission in progress, LEDR(8)
    tx      : OUT std_logic); --transmit pin
END COMPONENT;
```

Figura 11. Componente de la entidad “`uart`”.

```

COMPONENT debounce IS
    PORT ( Clock      : IN      STD_LOGIC;
            button     : IN      STD_LOGIC;
            debounced  : BUFFER  STD_LOGIC);
END COMPONENT;

COMPONENT bcd7seg_sec IS
    PORT ( bcd : in std_logic_vector(3 downto 0);
            display : out std_logic_vector(7 downto 0)
        );
end COMPONENT;

```

Figura 12. Componentes de las entidades “debounce” y “bcd7seg_sec”.

Como siguiente paso se crean señales internas para usarse en las instancias de cada componente. Existen señales internas que se utilizan para el proceso de la UART, así como para el debounce y lo que necesita el display.

```

SIGNAL tx_ena_de10:      std_logic := '1';
SIGNAL tx_busy_de10:     std_logic;
SIGNAL tx_data_de10:     std_logic_vector(7 DOWNTO 0);

SIGNAL rx_busy_de10:     std_logic;
SIGNAL rx_error_de10:    std_logic;
SIGNAL rx_data_de10:     std_logic_vector(7 DOWNTO 0);

--Botones
SIGNAL key0_db:          std_logic;
SIGNAL key0_db_past:     std_logic := '0';
SIGNAL key1_db:          std_logic;
SIGNAL key1_db_past:     std_logic := '0';

--Display
SIGNAL display :         std_logic_vector(7 DOWNTO 0);
SIGNAL bcd   :           std_logic_vector(3 DOWNTO 0);

```

Figura 13. Señales internas.

Tras hacer esto, se hacen las instancias con estas señales internas, haciéndose entonces asignaciones de las señales creadas anteriormente con los puertos de las entidades que se pusieron como componentes. Esto es importante para tener una forma de relacionar los elementos desarrollados en el proyecto de VHDL.

```

uart_0      : uart    PORT MAP( CLOCK_50, Sw(9), tx_ena_de10, tx_data_de10, GPIO_24, rx_busy_de10, rx_error_de10, rx_data_de10, tx_busy_de10, GPIO_25 );
button_0    : debounce PORT MAP( CLOCK_50, KEY(0), key0_db );
button_1    : debounce PORT MAP( CLOCK_50, KEY(1), key1_db );
display7seg : bcd7seg_sec PORT MAP( rx_data_de10(3 DOWNTO 0), HEX0 );

```

Figura 14. Instancias de los componentes.

Empezando por el proceso que tiene como lista de sensitividad el reloj interno del FPGA, por cada que hay un flanco de subida del reloj, se le asignan los datos recibidos por la UART a la señal interna del display de 7 segmentos. Aunado a esto, dependiendo de cada switch que se levante (debido a que se está usando los switches del 0 al 3) dependen los datos que se envían.

```

--Process to receive
PROCESS(CLOCK_50)
BEGIN
    IF(rising_edge (CLOCK_50)) THEN
        bcd <= rx_data_de10(3 downto 0);
    END IF;
END PROCESS;

```

Figura 15. Cláusula “if” dentro del flanco de subida del reloj.

Los switches están distribuidos de la siguiente manera:

- Switch 0: movimiento hacia delante. Se está enviando un 5 en binario, que es un valor predeterminado en Unity.

- Switch 1: movimiento hacia atrás. Se está enviando un 6 en binario.
- Switch 2: movimiento hacia la derecha. Se está enviando un 2 en binario.
- Switch 3: movimiento hacia la izquierda. Se está enviando un 3 en binario.

En el caso de los push buttons, estos están programados de esta manera:

- KEY0: frenado. Se está enviando un 4 en binario.
- KEY1: acelerador. Se está enviando un 1 en binario.

De otra manera, si estos casos no ocurren, se da un “1” lógico a “tx_enade10”, no transmitiendo información. También se asigna un 0 en binario a “tx_data_de10”. De igual manera, al final se van asignando los valores de los push buttons a las variables internas.

```
--Process to transmit data according to the interface inputs
PROCESS( CLOCK_50 )
BEGIN
    IF rising_edge( CLOCK_50 ) THEN
        IF SW(0) = '1' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000101";    -- Send value 5 when switch 1 is on FORWARD

        ELSIF SW(1) = '1' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000110";    -- Send value 6 when switch 1 is on BACKWARD

        ELSIF SW(2) = '1' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000010";    -- Send value 2 when switch 1 is on RIGHT

        ELSIF SW(3) = '1' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000011";    -- Send value 3 when switch 1 is on LEFT

        ELSIF key0_db = '1' and key0_db_past = '0' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000100";    -- Send value 4 when button is pressed

        ELSIF key1_db = '1' and key1_db_past = '0' THEN
            tx_enade10 <= '0';
            tx_data_de10 <= "00000001";    -- Send value 1 when button is pressed

        ELSE
            tx_enade10 <= '1';
            tx_data_de10 <= "00000000";    -- Send value 0 when neither switch 1 is on nor button is pressed
        END IF;
        key0_db_past <= key0_db;
        key1_db_past <= key1_db;
    END IF;
END PROCESS;

END Structural;
```

Figura 16. Cláusulas para mandar información dependiendo de switches y botones.

VI. A. b. *bcd7seg_sec*

En la entidad que ayuda a obtener la cantidad de plantas que se están recibiendo, se tuvieron que declarar dos puertos: “bcd” y “display”. El primero es una entrada que recibe el dato mencionado anteriormente, mientras que el segundo puerto es una salida que ilumina al display de 7 segmentos.

```

Library ieee;
use ieee.std_logic_1164.all;

entity bcd7seg_sec is
  port(
    bcd: in std_logic_vector(3 downto 0);
    display: out std_logic_vector(7 downto 0)
  );
end entity;

```

Figura 17. Declaración de la entidad “bcd7seg_sec”.

Dentro de la arquitectura, se está haciendo un switch case dependiendo del valor que se está recibiendo de las plantas, donde con base en esto mismo se va iluminando o no iluminando cada segmento del display.

```

architecture behavior of bcd7seg_sec is
begin
  process (bcd)
  begin
    case bcd is
      when "0000" =>
        display <= "11000000";
      when "0001" =>
        display <= "11111001";
      when "0010" =>
        display <= "10100100";
      when "0011" =>
        display <= "10110000";
      when "0100" =>
        display <= "10011001";
      when "0101" =>
        display <= "10010010";
      when "0110" =>
        display <= "10000010";
      when "0111" =>
        display <= "11111000";
      when "1000" =>
        display <= "10000000";
      when "1001" =>
        display <= "10011000";
      when "1010" =>
        display <= "10001000";
      when "1011" =>
        display <= "10000011";
      when "1100" =>
        display <= "10100111";
      when "1101" =>
        display <= "10100001";
      when "1110" =>
        display <= "10000110";
      when others =>
        display <= "10001110";
    end case;
  end process;
end architecture;

```

Figura 18. Casos para desplegar dígitos en el display de 7 segmentos.

VI. B. Prueba de funcionamiento

Para el funcionamiento de dos displays se creó una entidad nueva, la cual se nombró *displaycont*. En ella se agrega como componente la entidad *bcd7seg_sec* que se explicó con anterioridad.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity displaycont is
    Port (
        clk      : in STD_LOGIC;
        counter  : in STD_LOGIC_VECTOR(7 downto 0);
        unidades : out STD_LOGIC_VECTOR(7 downto 0);
        decenas  : out STD_LOGIC_VECTOR(7 downto 0)
    );
end displaycont;

architecture Behavioral of displaycont is
    signal unids, decs, cents : STD_LOGIC_VECTOR(3 downto 0);
    signal num_int : integer range 0 to 999;

    component bcd7seg_sec is
    port(
        bcd: in std_logic_vector(3 downto 0);
        display: out std_logic_vector(7 downto 0)
    );
    end component;

```

Figura 19. Entidad para separar unidades y decenas para displays

Dentro de la arquitectura de esta entidad se realizan las operaciones aritméticas haciendo uso de módulos para separar las unidades y las decenas del vector de 8 bits que se reciben desde Unity. En esta arquitectura se instancian dos decoders utilizando como componente la entidad *bcd7seg_sec* para finalmente desplegar los dígitos en su correspondiente display.

```

begin
    num_int <= to_integer(unsigned(counter));

    -- Convertir unidades, decenas y centenas
    decs <= std_logic_vector(to_unsigned(((num_int mod 100) / 10), 4));
    unids <= std_logic_vector(to_unsigned((num_int mod 10), 4));

    decoder_un: bcd7seg_sec
        port map (
            unids,
            unidades
        );

    decoder_tens: bcd7seg_sec
        port map (
            decs,
            decenas
        );
end Behavioral;

```

Figura 20. Arquitectura de separador de unidades y decenas utilizando *bcd7seg_sec*

Integrándose en la entidad *de10*, se declara igualmente como componente la entidad *displaycount*.

```

component displaycont is
    Port (
        clk      : in STD_LOGIC;
        counter  : in STD_LOGIC_VECTOR(7 downto 0);
        unidades : out STD_LOGIC_VECTOR(7 downto 0);
        decenas  : out STD_LOGIC_VECTOR(7 downto 0)
    );
end component;

```

Figura 21. Declaración de componente en entidad *de10.vhd*

Por último en la arquitectura se instancia y se le brindan los argumentos CLOCK_50 (clock del FPGA), rx_data_de10 (datos recibidos), y los dos displays que serán utilizados.

```
display7seg : displaycont PORT_MAP(CLOCK_50, rx_data_de10, HEX0, HEX1);
```

Figura 22. Instanciación de componente para desplegar el número recibido.

Como evidencia del funcionamiento se muestra el despliegue de maíces obtenidos al recolectarlos dentro del juego.



Figura 23. Printscreen al inicio del juego donde hay 0 maíces recolectados.

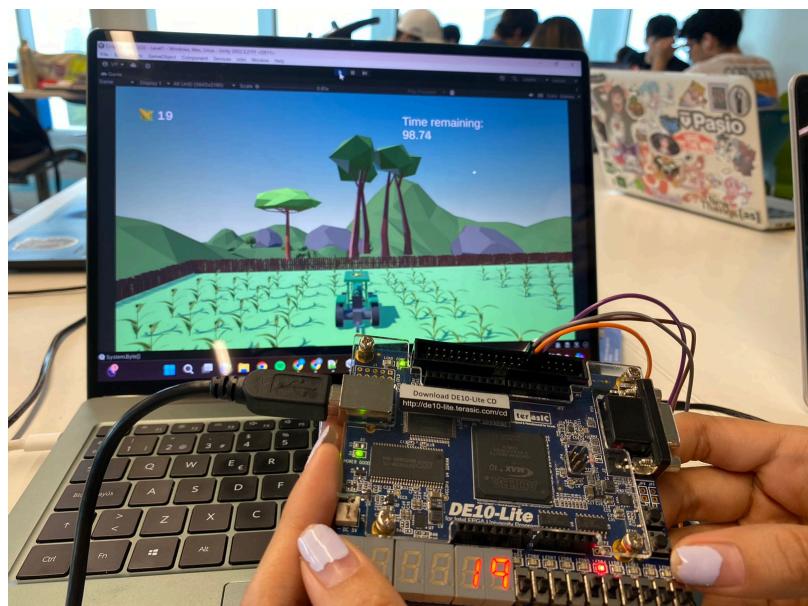


Figura 24. Printscreen durante el juego donde hay 19 maíces recolectados.

VII. Conexión Serial

En el caso de la conexión serial entre el videojuego y el FPGA, se necesitó de un espacio de nombres proporcionado por el framework .NET. Un espacio de nombres en programación se refiere a una forma de organizar y agrupar nombres de clases, interfaces, funciones y otros elementos dentro de un contexto específico.

Siendo en Unity *System.IO.Ports* el espacio de nombres empleado, esto brindó la comunicación con puertos seriales por medio de funciones integradas como *write()* y *read()*.

Cabe aclarar que se creó un código únicamente con la intención de enviar y recibir información, al cual se le llamó *de10l*. Para facilitar el entendimiento de este se destacan las siguientes líneas:

Se crea el objeto *de10l* del tipo *SerialPort*, donde se indica qué puerto de comunicación se utilizará y la velocidad en baudios (número de veces por segundo que una señal de comunicaciones serie cambia de estado).

FTDI	FPGA
TX	GPIO 27
RX	GPIO 28
VCC	3.3 V
GND	GND

Tabla 1. Conexiones entre la FTDI y el FGPA.

```
public SerialPort de10l = new SerialPort("COM4", 115200);
```

Figura 25. Instanciación de SerialPort.

Dentro del método *Start()* se asegura que el puerto por el cual se requiere recibir o enviar datos esté abierto. En caso de que no lo esté, lo abre y adicionalmente transmite un byte para hacer un *clear* a los LEDs de la DE10-Lite.

```
void Start()
{
    CornText = GetComponent<TextMeshProUGUI>();
    if (!de10l.IsOpen)
    {
        de10l.Open();
        de10l.ReadTimeout = 1;
    }

    if (de10l.IsOpen)
    {
        var dataByte = new byte[] { 0x00 };
        de10l.Write(dataByte, 0, 1);
    }
}
```

Figura 26. Método Start().

En el método *Update()*, que se ejecuta automáticamente cada frame del juego, se reciben datos enviados desde el FPGA. Dependiendo del número recibido, se ejecuta una acción dentro del videojuego; ya sea conducción hacia adelante, hacia atrás, *boost* de velocidad o cambio de dirección del tractor. También en el método se incluye la transmisión de datos hecha por el juego (número de plantas recolectadas).

```

void Update()
{
    if (!de10l.IsOpen)
    {
        de10l.Open();
        de10l.ReadTimeout = 1;
    }

    if (de10l.IsOpen)
    {
        try
        {
            int value;
            if (de10l.BytesToRead > 0)
            {
                value = de10l.ReadByte();
                de10l.DiscardInBuffer();
            }
            else
            {
                value = 0;
            }

            if (value == 0x01)
            {
                Recio = 1;
            }
            else if (value == 0x02)
            {
                Right = 1;
            }
            else if (value == 0x03)
            {
                Left = 1;
            }
            else if (value == 0x04)
            {
                Break = 1;
            }
            else if (value == 0x05)
            {
                Forward = 1;
            }
            else if (value == 0x06)
            {
                Backward = 1;
            }
            else
            {

                Recio = 0;
                Break = 0;
                Right = 0;
                Left = 0;
                Forward = 0;
                Backward = 0;
            }

            de10l.Write(playerInventory.NumberOfCorns.ToString());
        }
        catch { }
    }
}

```

Figura 27. Método Update()

VII. A. Demostración de funcionamiento con ejemplos

VII. A. a. Prueba 1.

- **Acción:** Movimiento hacia adelante.
- **Descripción:** Para lograr representar el movimiento hacia adelante, dentro del código en Quartus Prime se tiene establecida la conexión serial, donde dentro de un proceso se verifica que el switch 0 esté activado (“1” lógico) para ejecutar las acciones.

```
--Process to transmit data according to the interface inputs
PROCESS( CLOCK_50 )
BEGIN
  IF rising_edge( CLOCK_50 ) THEN
    IF SW(0) = '1' THEN
      tx_ena_de10 <= '0';
      tx_data_de10 <= "00000101"; -- Send value 5 when switch 0 is on FORWARD
```

Figura 28. Código VHDL para enviar un 5 al activar switch 0.

Mediante el software *Termite*, se puede ver cómo al poner en “1” lógico el switch 0 se está transmitiendo el dato “5” en binario, donde la tasa de datos que se están recibiendo depende del *baud rate* establecido. Así comprobando la conexión serial.



Figura 29. Transmisión de datos en *Termite* con el switch 0.

Para la parte de Unity, dentro de los códigos en C, se encuentra el llamado “de10”, en este se establece que al recibir un valor de “5”, *Forward* será igual a “1”. Ahora el método que se encuentra en *CarController* llamado “*FixedUpdate*” se encarga de realizar el movimiento del tractor dentro del videojuego. Establecemos una condición para verificar que la acción está instanciada dentro de la clase de10l. Si *Forward* es igual a “1”, entonces se calcula la aceleración actual, que es el valor que se propone para moverse hacia adelante, multiplicando por 1.

```

if (de10l.Forward == 1)
{
    aAceleracion = aceleracion * de10l.Forward;
}

```

Figura 30. Código del movimiento hacia adelante en Unity.

Comprobando el recibimiento del dato transmitido por el código VHDL se demuestra una impresión de la terminal de Unity.

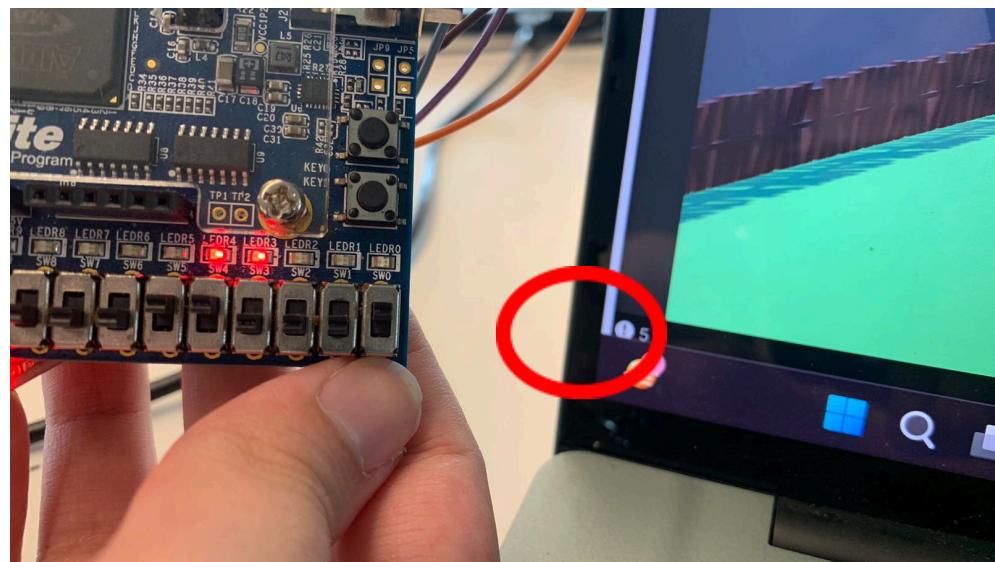


Figura 31. Debug Log “5” al activar SW0.

VII. A. b. Prueba 2.

- **Acción:** Aumento de velocidad.
- **Descripción:** Para lograr representar el aumento de velocidad, dentro del código en Quartus Prime se tiene establecida la conexión serial, donde dentro de un proceso se verifica que el push button 1 (KEY1) esté activado (“1” lógico) para ejecutar las acciones. También se necesita que el switch 0, que se utiliza para avanzar, esté en “1”, dejar que el tractor avance y después bajar el switch para poder mandar la señal del botón.

```

ELSIF key1_db = '1' and key1_db_past = '0' THEN
    tx_ena_de10 <= '0';
    tx_data_de10 <= "00000001"; -- Send value 1 when button is pressed

```

Figura 32. Código VHDL para enviar un 1 al activar KEY1.

Después, con ayuda del software Termite se asegura que la placa esté recibiendo la señal con valor binario de 5 primero (forward), y después el valor 1 (acelerador), comprobando la conexión serial. Como con anterioridad se mostró la transmisión del valor “5”, la siguiente figura demostrará la transmisión de “1”.

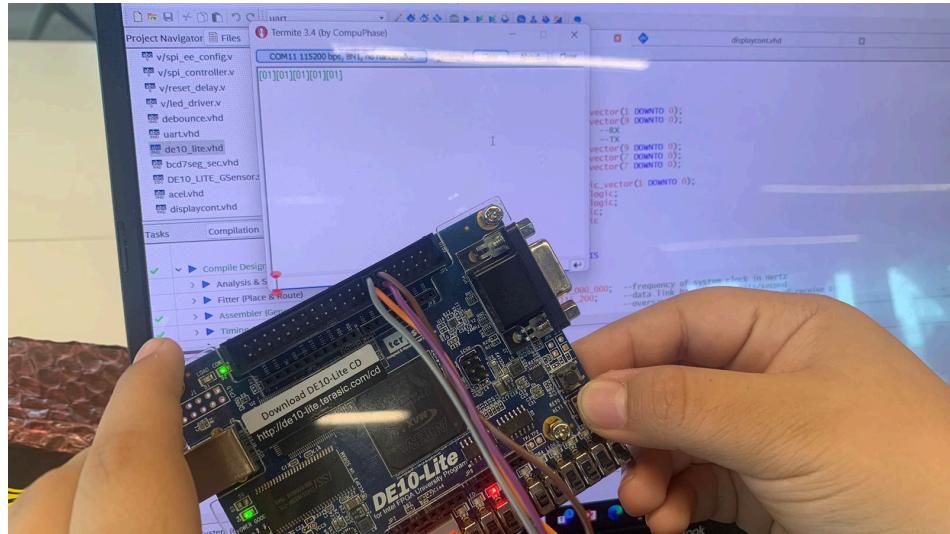


Figura 33. Transmisión de datos en Termite con KEYI.

En Unity, dentro del script “de10” se verifica si hay datos disponibles en el puerto serial. Si los hay, entonces se lee un byte, se comprueba si ese byte que se leyó es igual a “1”. En ese caso, se establece la variable “recio” para el aumento de velocidad en 1. Si no existen datos disponibles, la variable “value” se establece en 0. La línea correspondiente a de10l.DiscardInBuffer() sirve para que el movimiento del tractor sea de manera constante.

```

int value;
if (de10l.BytesToRead > 0)
{
    value = de10l.ReadByte();
    de10l.DiscardInBuffer();
}
else
{
    value = 0;
}

if (value == 0x01)
{
    Recio = 1;
}

```

Figura 34. Código del aumento de velocidad en Unity.

Comprobando el recibimiento del dato transmitido por el código VHDL se demuestra una impresión de la terminal de Unity.

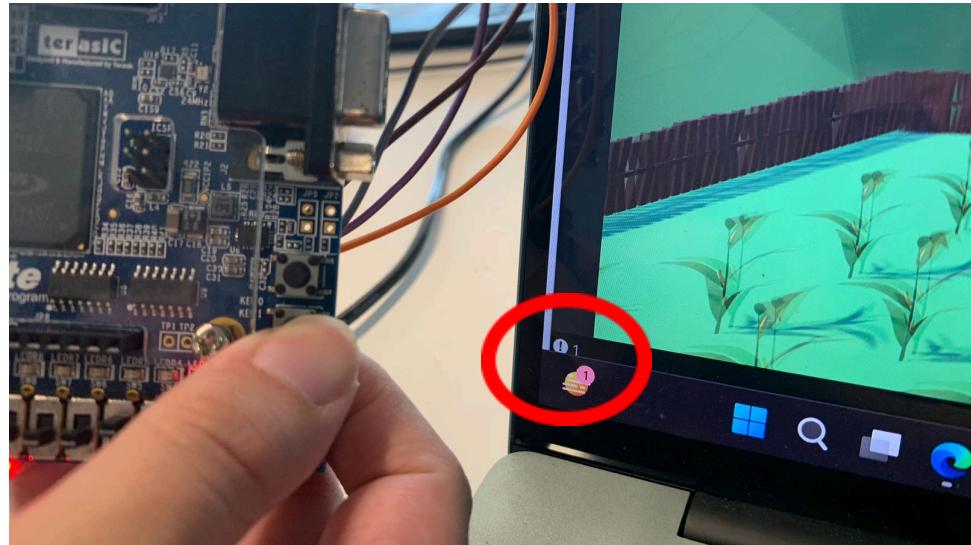


Figura 35. Debug Log “1” al presionar KEY1.

VII. A. c. Prueba 3.

- **Acción:** Contador de plantas.
- **Descripción:** Para el caso del contador, se envían datos desde Unity hasta el FPGA. Para esto se configura la interfaz de usuario para mostrar la cantidad de plantas en el inventario del jugador. Dentro del código en C para la clase de10, se tiene declarada la variable “de10l” de tipo *serialPort*, que representa la conexión serial con el FPGA en el puerto COM4, siendo el baudrate de 115200 (velocidad de transmisión).

```
Script de Unity (4 referencias de recurso) | 0 referencias
public class InventoryUI : MonoBehaviour
{
    public SerialPort de10l = new SerialPort("COM4", 115200);

    private TextMeshProUGUI CornText;

    // Start is called before the first frame update
    void Start()
    {
        CornText = GetComponent<TextMeshProUGUI>();
    }

    public void UpdateCornText(PlayerInventory playerInventory)
    {
        // Actualiza el texto en Unity
        CornText.text = playerInventory.NumberOfCorns.ToString();
    }
}
```

Figura 36. Código para actualizar el texto en la interfaz de usuario en Unity.

En la siguiente línea se muestra que el método “write” envía los datos, y *playerInventory.NumberOfCorns* es la cantidad de plantas en el inventario, se

convierte un número entero en un conjunto de bytes utilizando la clase BitConverter, posteriormente estos bytes se envían a través de un flujo de salida. Así, el número de plantas que el jugador llegue a recolectar se verá reflejado en la placa con el display de 7 segmentos, que irá aumentando cada vez que el tractor recolecta una planta.

```
96
97     // Contador de maíces
98     int MyInt = playerInventory.NumberOfCorns;
99     byte[] b = BitConverter.GetBytes(MyInt);
100    de10l.Write(b, 0, 1);
101
```

Figura 37. Método para enviar la cantidad de maíces desde Unity

Comprobando el recibimiento del dato por el código VHDL, se demuestra una impresión de la terminal de Unity con el número de maíces recolectado.



Figura 38. Debug Log en la esquina inferior izquierda

VIII. Procesador

En la última etapa del proyecto, se utilizó una herramienta de software para implementar el diseño en un lenguaje de bajo nivel, específicamente lenguaje ensamblador. Esto se hizo con la clara comprensión de que al emplear Gumnut, se hacen referencias a direcciones que están vinculadas al ensamblador. Esta elección permite que el programa sea compatible con cualquier microcontrolador que comparta las mismas características.

Dentro de la programación de este procesador se incluye la funcionalidad de los switches, botones, display y acelerómetro.

VIII. A. Explicación de código en VHDL

VIII. A. a. de10_lite

En la parte inicial se agregaron los puertos de la entidad principal. Están el reloj de la placa, los GPIOs de la UART, los displays, los LEDs, el switch habilitador del envío de datos del acelerómetro, y los puertos necesarios para que este último funcione.

```

5  ENTITY de10_lite IS
6    PORT( CLOCK_50      : IN  std_logic;
7          KEY           : IN  std_logic_vector(1 DOWNTO 0);
8          SW             : IN  std_logic_vector(9 DOWNTO 0);
9          GPIO_24        : IN  std_logic;  --RX
10         GPIO_25        : OUT std_logic;  --TX
11         LEDR           : OUT std_logic_vector(9 DOWNTO 0);
12         HEX0           : OUT std_logic_vector(7 DOWNTO 0);
13         HEX1           : OUT std_logic_vector(7 DOWNTO 0);
14         SW_Int          : IN  std_logic;
15         GSENSOR_INT_I  : IN  std_logic_vector(1 DOWNTO 0);
16         GSENSOR_SDI_I  : INOUT std_logic;
17         GSENSOR_SDO_I  : INOUT std_logic;
18         GSENSOR_CS_N_I : OUT std_logic;
19         GSENSOR_SCLK_I : OUT std_logic
20       );
21   END;
22 
```

Figura 39. Entidad de componente de10_lite.

El componente de la UART ayuda a la recepción y transmisión de datos desde Unity a la placa y viceversa. Sin esto, no podría haber una comunicación exitosa y un tráfico de datos. Los puntos a resaltar en esta entidad son *RX* y *TX*, que serán asignados después con los pines GPIO mencionados arriba.

```

COMPONENT uart IS
  GENERIC(
    clk_freq : integer    := 50_000_000;  --frequency of system clock in Hertz
    baud_rate : integer   := 115_200;    --data link baud rate in bits/second
    os_rate : integer    := 16;          --oversampling rate to find center of receive bits
    --(in samples per baud period)
    d_width : integer    := 8;           --data bus width
    parity : integer     := 0;           --0 for no parity, 1 for parity
    parity_eo : std_logic := '0');      --'0' for even, '1' for odd parity
  PORT(
    clk      : IN  std_logic;          --system clock
    reset_n : IN  std_logic;          --asynchronous reset
    tx_ena  : IN  std_logic;          --initiate transmission
    tx_data : IN  std_logic_vector(d_width-1 DOWNTO 0); --data to transmit
    rx      : IN  std_logic;          --receive pin
    rx_busy : OUT std_logic;         --data reception in progress, LEDR(9)
    rx_error : OUT std_logic;        --start, parity, or stop bit error detected
    rx_data : OUT std_logic_vector(d_width-1 DOWNTO 0); --data received
    tx_busy : OUT std_logic;         --transmission in progress, LEDR(8)
    tx      : OUT std_logic);        --transmit pin
  END COMPONENT;

```

Figura 40. Componente de la UART.

Se referencia también a la entidad *gumnut_with_mem*, que contiene los puertos del procesador Gumnut. Se incluye un reloj, un reset, los puertos de I/O (como para reconocer que existen ciclos de I/O, el *write enable* de los outputs, las direcciones, los datos de entrada y salida, etc). Esto se manejará en secciones posteriores.

```

47 component gumnut_with_mem IS
48   generic(
49     IMem_file_name : string      := "gasm_text.dat";
50     DMem_file_name : string      := "gasm_data.dat";
51     debug          : boolean     := false
52   );
53   port(
54     clk_i          : in std_logic;
55     rst_i          : in std_logic;
56     -- I/O port bus
57     port_cyc_o    : out std_logic;
58     port_stb_o    : out std_logic;
59     port_we_o     : out std_logic;
60     port_ack_i    : in std_logic;
61     port_adr_o    : out unsigned(7 downto 0);
62     port_dat_o    : out std_logic_vector(7 downto 0);
63     port_dat_i    : in std_logic_vector(7 downto 0);
64     -- Interrupts
65     int_req        : in std_logic;
66     int_ack        : out std_logic
67   );
68   ...
69 
```

Figura 41. Componente de Gumnut

Al hacer señales internas, se puede instanciar correctamente cada componente. Esto ayuda a que se pueda implementar cada entidad que se está incluyendo en la entidad principal.

```

89 SIGNAL clk_i, rst_i          : std_logic;
90 SIGNAL port_cyc_o, port_stb_o : std_logic;
91 SIGNAL port_we_o, port_ack_i : std_logic;
92 SIGNAL int_req, int_ack     : std_logic;
93 SIGNAL port_adr_o           : unsigned(7 DOWNTO 0);
94 SIGNAL port_dat_o, port_dat_i : std_logic_vector(7 DOWNTO 0);
95
96 SIGNAL tx_ena_de10:         std_logic := '1';
97 SIGNAL tx_busy_de10:        std_logic;
98 SIGNAL tx_data_de10:        std_logic_vector(7 DOWNTO 0);
99
100 SIGNAL rx_busy_de10:       std_logic;
101 SIGNAL rx_error_de10:      std_logic;
102 SIGNAL rx_data_de10:       std_logic_vector(7 DOWNTO 0);
103
104 --Botones
105 SIGNAL key0_db:            std_logic;
106 SIGNAL key0_db_past:       std_logic := '0';
107 SIGNAL key1_db:            std_logic;
108 SIGNAL key1_db_past:       std_logic := '0';
109
110 --Display
111 SIGNAL display :           std_logic_vector(7 DOWNTO 0);
112 SIGNAL bcd :                std_logic_vector(3 DOWNTO 0);
113 SIGNAL int_mov :            STD_LOGIC_VECTOR(9 DOWNTO 0);
114 SIGNAL int_mov_modified:   std_logic_vector(7 downto 0);
115 SIGNAL unids, decs :        STD_LOGIC_VECTOR(3 downto 0);
116 SIGNAL num_int :            integer range 0 to 999;
117 
```

Figura 42. Señales declaradas en la arquitectura de de10_lite

Dado que en el proceso que le sigue a las instanciaciones se especifica que si *rst_i* es igual a ‘1’ no se están enviando datos, se tiene que establecer esta señal como ‘0’. En la siguiente imagen se ve cómo se relacionan las señales internas con los componentes, además de los puertos de la entidad principal. Es importante mencionar que el ‘1’ asignado al puerto *reset_n* de la UART es para que siempre se esté recibiendo y transmitiendo información.

```

120 BEGIN
121   rst_i <= '0';
122
123   gumnut : gumnut_with_mem PORT MAP( CLOCK_50, rst_i, port_cyc_o, port_stb_o, port_we_o, '1', port_adr_o, port_dat_o, port_dat_i, int_req,
124                                         int_ack );
125   uart_0 : uart PORT MAP( CLOCK_50, '1', tx_ena_de10, tx_data_de10, GPIO_24, rx_busy_de10, rx_error_de10, rx_data_de10, tx_busy_de10,
126                           GPIO_25 );

```

Figura 43. Instanciaciones de componentes en la arquitectura de de10_lite.

Se empieza con los primeros dos procesos que están relacionados con Gumnut y la UART. Es decir, con el envío de datos desde el dispositivo *TX_DATA* y el control de inicio de la transmisión *TX_START*. Para *tx_data* se le asigna la dirección “00000000” (que es 0 en binario), y se habilitan las señales de control *port_cyc_o* y *port_stb_o* para indicar que se está llevando a cabo un proceso y *port_we_o* para indicar que se está realizando una operación de escritura, después se carga el dato presente en *port_dat_o* en *tx_data_de10*.

Para *tx_start* se le asigna la dirección “00000001” (1 en binario), y de igual manera se habilitan las mismas señales para que sea una operación de escritura. Al final se carga el bit menos significativo de *port_dat_o*(0) a *tx_ena_de10* para permitir la transmisión de datos.

```

134 -- Output => TX_DATA -> data memory address: 0x00
135 PROCESS( CLOCK_50, rst_i )
136   BEGIN
137     IF rst_i = '1' THEN
138       tx_data_de10 <= ( OTHERS => '0' );
139     ELSIF rising_edge( CLOCK_50 ) THEN
140       IF port_adr_o = "00000000" and -- address port
141         port_cyc_o = '1' and -- control signals for I/O
142         port_stb_o = '1' and
143         port_we_o = '1' -- 'write' operation
144       THEN
145         tx_data_de10 <= port_dat_o;
146       END IF;
147     END IF;
148   END PROCESS;
149
150 -- Output => TX_START -> data memory address: 0x01
151 PROCESS( CLOCK_50, rst_i )
152   BEGIN
153     IF rst_i = '1' THEN
154       tx_ena_de10 <= '1';
155     ELSIF rising_edge( CLOCK_50 ) THEN
156       IF port_adr_o = "00000001" and -- address port
157         port_cyc_o = '1' and -- control signals for I/O
158         port_stb_o = '1' and
159         port_we_o = '1' -- 'write' operation
160       THEN
161         tx_ena_de10 <= port_dat_o(0);
162       END IF;
163     END IF;
164   END PROCESS;

```

Figura 44. Procesos para transmisión de datos.

VIII. A. b. Display de 7 segmentos (contador de plantas)

El display de 7 segmentos cuenta como una salida de información para el procesador. Por ello, sus cláusulas son diferentes a las de las entradas, donde el *write enable output* está habilitado. Se le asignó una dirección de “00000010”, que es un 2 en binario. El contador que está saliendo tiene una dirección en ensamblador, pero esto es para que pueda funcionar como *output*. Se está entrando a la operación al momento de coincidir con la dirección de puerto de la variable *disp*.

Al caer en el llamado por la dirección de puerto, se asignan los datos recibidos por la UART a un entero, mismo que mediante operaciones se asigna a variables de decenas y unidades. Esto es debido a que se está operando hasta decenas en el juego. Se hace un *case* para cada variable, donde de coincidir con algún número de los desplegados en las cláusulas, se encienden los respectivos segmentos de cada display. HEX0 es el display de las unidades, mientras que HEX1 el de las decenas.

```

166 |      -- Output => DISPLAY -> data memory address: 0x02
167 |      PROCESS( CLOCK_50, rst_i )
168 |      BEGIN
169 |          IF rising_edge( CLOCK_50 ) THEN
170 |              IF port_adr_o = "00000010" and -- address port
171 |                  port_cyc_o = '1' and -- control signals for I/O
172 |                  port_stb_o = '1' and
173 |                  port_we_o = '1'        -- 'write' operation
174 |              THEN
175 |                  num_int <= to_integer(unsigned(rx_data_de10));
176 |                  -- Convertir unidades, decenas y centenas
177 |                  decs <= std_logic_vector(to_unsigned(((num_int mod 100) / 10), 4));
178 |                  unids <= std_logic_vector(to_unsigned((num_int mod 10), 4));
179 |                  case unids is
180 |                      when "0000" =>
181 |                          HEX0 <= "11000000";
182 |                      when "0001" =>
183 |                          HEX0 <= "11111001";
184 |                      when "0010" =>
185 |                          HEX0 <= "10100100";
186 |                      when "0011" =>
187 |                          HEX0 <= "10110000";
188 |                      when "0100" =>
189 |                          HEX0 <= "10011001";
190 |                      when "0101" =>
191 |                          HEX0 <= "10010010";
192 |                      when "0110" =>
193 |                          HEX0 <= "10000010";
194 |                      when "0111" =>
195 |                          HEX0 <= "11111000";
196 |                      when "1000" =>
197 |                          HEX0 <= "10000000";
198 |                      when "1001" =>
199 |                          HEX0 <= "10011000";
200 |                      when "1010" =>
201 |                          HEX0 <= "10001000";
202 |                      when "1011" =>
203 |                          HEX0 <= "10000011";
204 |                      when "1100" =>
205 |                          HEX0 <= "10100111";
206 |                      when "1101" =>
207 |                          HEX0 <= "10100001";
208 |                      when "1110" =>
209 |                          HEX0 <= "10000110";
210 |                      when others =>
211 |                          HEX0 <= "10001110";
212 |                  end case;

```

```

        case decs is
            when "0000" =>
                HEX1 <= "11000000";
            when "0001" =>
                HEX1 <= "11111001";
            when "0010" =>
                HEX1 <= "10100100";
            when "0011" =>
                HEX1 <= "10110000";
            when "0100" =>
                HEX1 <= "10011001";
            when "0101" =>
                HEX1 <= "10010010";
            when "0110" =>
                HEX1 <= "10000010";
            when "0111" =>
                HEX1 <= "11111000";
            when "1000" =>
                HEX1 <= "10000000";
            when "1001" =>
                HEX1 <= "10011000";
            when "1010" =>
                HEX1 <= "10001000";
            when "1011" =>
                HEX1 <= "10000011";
            when "1100" =>
                HEX1 <= "10100111";
            when "1101" =>
                HEX1 <= "10100001";
            when "1110" =>
                HEX1 <= "10000110";
            when others =>
                HEX1 <= "10001110";
        end case;
    
```

Figura 45. Proceso del DISPLAY

En ensamblador, tras asignar las direcciones que coinciden con VHDL al display y al contador, se reciben los datos del contador, y se despliega este mismo en el display. Se regresa al *main* después de esto. Esta parte es de suma importancia, ya que el contador siempre se tiene que estar actualizando.

```

disp:           bss 1 ;2
cont:          bss 1 ;9
counter_func:  inp r6, cont
               out r6, disp
               ret

```

Figura 46. Función para mantener el contador actualizado

VIII. A. c. Push buttons (freno y acelerador)

Debido a que los *push buttons* de la DE10-Lite tienen por fábrica una resistencia *pull-up*, siempre que no se esté presionando alguno de sus botones se está mandando el estado lógico ‘1’. En el caso de que se presionen, se envía un ‘0’. Por ello, con el componente *debounce* se está corrigiendo esto guardando el estado del botón para evitar también un error en el envío del estado debido a oscilaciones. A la par, se implementan cláusulas para enviar un ‘0’ cuando no se presiona un botón, y en caso contrario se envía un ‘1’.

```

COMPONENT debounce IS
    PORT ( Clock      : IN      STD_LOGIC;
           button     : IN      STD_LOGIC;
           debounced : BUFFER STD_LOGIC);
END COMPONENT;

```

Figura 47. Componente debounce

Con *PORT MAP* se están asignando los botones, y ahora las señales que ya están corregidas están en el último puerto.

```
button_0 : debounce PORT MAP(CLOCK_50, KEY(0), key0_db);
button_1 : debounce PORT MAP(CLOCK_50, KEY(1), key1_db);
```

Figura 48. Instanciaciones para hacer debounce de botones.

Dentro del *when else* en *port_dat_i* se está asignando el dato de cada botón dependiendo de su dirección. Por ejemplo, en el caso de KEY0 es la dirección “00000011”, que es un 3 en binario. Por otro lado, KEY1 tiene una dirección “00000100”, que es un 4 en binario. Es importante recalcar que los estados que envía cada botón son de 1 bit, por lo que por esta misma razón se están concatenando 7 bits con ‘0’, para coincidir con los 8 bits que tiene la señal *port_dat_i*.

```
port_dat_i <= "0000000" & key0_db WHEN (port_adr_o = "00000011") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & key1_db WHEN (port_adr_o = "00000100") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
```

Figura 49. Asignación de datos de botones a port_dat_i dependiendo de la dirección.

En ensamblador, se le asignan las direcciones a KEY0 y KEY1. Se puede ver cómo coinciden con el código de VHDL. En cada función para cada botón, esencialmente se está leyendo el dato que envía cada botón. Si se recibe un estado lógico ‘1’, se envía el dato relacionado a su función en Unity. En este último, el dato “4” (KEY0) es el freno, mientras que el dato “1” (KEY1) es el acelerador. En caso de que KEY0 no esté presionado, se revisa el estado de KEY1. Si KEY1 no está presionado, se pasa a comprobar el estado del acelerómetro.

```
KEY0:      bss 1 ;3
KEY1:      bss 1 ;4

KEY0_data: byte 4
KEY1_data: byte 1
```

Figura 50. Asignación de direcciones y datos de los botones.

```
next_KEY0: inp r2, KEY0
          and r2, r2, 1
          bz next_KEY1
          ldm r3, KEY0_data
          out r3, TX_DATA
          out r0, TX_START
          ldm r3, tx_disable
          out r3, TX_START
```

Figura 51. Función para KEY0 en ensamblador.

```
next_KEY1: inp r3, KEY1
          and r3, r3, 1
          bz next_acel
          ldm r4, KEY1_data
          out r4, TX_DATA
          out r0, TX_START
          ldm r4, tx_disable
          out r4, TX_START
```

Figura 52. Función para KEY1 en ensamblador.

VIII. A. d. Switches (adelante y atrás)

Los switches utilizados en el juego son SW0 y SW1. En el caso de SW0, este ayuda a ir hacia adelante, mientras que SW1 ayuda a ir hacia atrás. Cada uno tiene una dirección diferente en ensamblador. En el caso de SW0, este tiene una dirección “00000101”, que es un 5 en binario. En cuanto a SW1, este tiene una dirección “00000110”, que es un 6 en binario. Por cada que se esté coincidiendo con esta dirección en el llamado, se estará enviando el estado de cada switch en un arreglo de 8 bits a *port_dat_i*. Los switches, al estar levantados, envían un estado lógico de ‘1’. En el caso de no estar levantados, envían un ‘0’ lógico.

```
port_dat_i <= "0000000" & key0_db WHEN (port_adr_o = "00000011") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & key1_db WHEN (port_adr_o = "00000100") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW(0) WHEN (port_adr_o = "00000101") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW(1) WHEN (port_adr_o = "00000110") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW_int WHEN (port_adr_o = "00000111") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
int_mov_modified WHEN (port_adr_o = "00001000") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
UNAFFECTED;
```

Figura 53. Asignación de datos de switches a port_dat_i dependiendo de la dirección.

En ensamblador se aprecia cómo estos switches tienen las direcciones que se mencionaron anteriormente. Dado que en Unity el valor “5” significa ir hacia adelante y el valor “6” es ir hacia atrás, esto es precisamente lo que se está asignando. Las operaciones para los switches son muy parecidas, por lo que generalmente se está recibiendo el dato de cada switch dependiendo de la función. Después de eso, se comprueba si se está enviando un ‘1’ lógico en cada switch. Si es así, se manda el dato correspondiente al switch. En caso contrario, en la función de SW0 se pasa a revisar SW1. En la función de este último, se pasa a comprobar KEY0.

```
SW0:      bss 1 ;5
SW1:      bss 1 ;6

SW0_data: byte 5
SW1_data: byte 6
```

Figura 54. Asignación de direcciones y datos de los switches.

```
next_SW0: inp r4, SW0
and r4, r4, 1
bz next_SW1
ldm r5, SW0_data
out r5, TX_DATA
out r0, TX_START
ldm r5, tx_disable
out r5, TX_START
jmp main

next_SW1: inp r5, SW1
and r5, r5, 1
bz next_KEY0
ldm r6, SW1_data
out r6, TX_DATA
out r0, TX_START
ldm r6, tx_disable
out r6, TX_START
```

Figura 55. Funciones de los switches en ensamblador.

VIII. A. e. Acelerómetro

El acelerómetro es el componente que determina si el tractor va hacia la izquierda o la derecha, esto dependiendo de la posición de la placa. Por esto mismo, en VHDL fue

necesario instanciar el componente *acel*. Este último obtiene los datos que el acelerómetro está mandando por el puerto LEDR.

Con la instrucción *PORT MAP*, se está asignando este puerto a una variable interna llamada *int_mov*. Sin embargo, debido a que *port_dat_i* (en donde se están escribiendo las entradas al procesador Gumnut) tiene un tamaño de 8 bits e *int_mov* es de 10 bits, se le asignaron los bits que necesita el acelerómetro a una nueva señal llamada *int_mov_modified*. Los bits que no fueron necesarios fueron los bits 4 y 5, debido a que estos funcionan como un habilitador para el acelerómetro en el puerto *KEY*. También se muestra lo que el acelerómetro está recibiendo en los LEDs de la placa.

Tras asignarse los bits necesarios para operar con el acelerómetro, la nueva señal *int_mov_modified* es ahora de 8 bits, que es el mismo tamaño que *port_dat_i*.

```
COMPONENT acel IS
  PORT( CLOCK_50 : IN std_logic;
        KEY      : IN std_logic_vector(1 DOWNTO 0);
        GSENSOR_INT : IN std_logic_vector(1 DOWNTO 0);
        GSENSOR_SDI : INOUT std_logic;
        GSENSOR_SDO : INOUT std_logic;
        GSENSOR_CS_N : OUT std_logic;
        GSENSOR_SCLK : OUT std_logic;
        LEDR      : OUT std_logic_vector(9 DOWNTO 0)
      );
END COMPONENT;
```

Figura 56. Componente del acelerómetro.

```
acel_0: acel PORT MAP(CLOCK_50, `SW(5 downto 4), `GSENSOR_INT_I(1 downto 0), GSENSOR_SDI_I, GSENSOR_SDO_I,
                       GSENSOR_CS_N_I, GSENSOR_SCLK_I, int_mov(9 downto 0));
int_mov_modified <= int_mov(9 downto 6) & int_mov(3 downto 0);
LEDR(9 downto 0) <= int_mov(9 downto 0);
```

Figura 57. Instanciación de acelerómetro, asignación de bits significativos a una segunda señal y asignación de datos del acelerómetro a los LEDs.

Después de esto, dentro de *when else* se está esperando hasta que la dirección en el puerto del procesador sea “00001000”, que es un 8 en binario (dirección del acelerómetro en ensamblador). En ese momento, los datos del acelerómetro se envían a Unity para poder afectar el comportamiento del tractor. En cuanto a *SW_Int*, se está esperando una dirección de “00000111”, que es un 7 en binario. Su función con el acelerómetro se explica a detalle en el código en ensamblador.

```
port_dat_i <= "00000000" & key0_db WHEN (port_adr_o = "00000011") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"00000000" & key1_db WHEN (port_adr_o = "00000100") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"00000000" & SW(0) WHEN (port_adr_o = "00000101") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"00000000" & SW(1) WHEN (port_adr_o = "00000110") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"00000000" & SW_int WHEN (port_adr_o = "00000111") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
int_mov_modified WHEN (port_adr_o = "00001000") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
UNAFFECTED;
```

Figura 58. Asignación de datos del acelerómetro a *port_dat_i* de acuerdo a la dirección.

Regresando a donde se mencionó la dirección dentro del procesador para el acelerómetro, este mismo tiene esa precisa dirección en ensamblador. Tras comprobar el estado de los switches en Gumnut, se obtiene la información del switch habilitador de los datos del acelerómetro (*SW_Int*), que ayuda a enviar datos de este último solamente cuando sea necesario. Si este switch está mandando un estado lógico ‘1’, entonces se lee lo que está mandando el acelerómetro. Se hace una máscara para los bits en caso de que el acelerómetro esté posicionado hacia la derecha. Si la operación da 0, significa que no está hacia la derecha, y se salta a *send_left*. De otra manera, se manda el valor correspondiente a la acción de ir a la derecha en Unity.

```

SW_Int:      bss 1 ;7
int_mov:     bss 1 ;8

right_data: byte 2
left_data:  byte 3

```

Figura 59. Asignación de direcciones y datos del switch habilitador y el acelerómetro.

```

next_acel: inp r1, SW_Int
           inp r7, int_mov
           and r1, r1, 1
           bz main
           and r7, r7, 0x0C
           and r3, r7, 0x03
           or r4, r7, r3
           bz send_left
           ldm r2, right_data
           out r2, TX_DATA
           out r0, TX_START
           ldm r2, tx_disable
           out r2, TX_START

```

Figura 60. Función del acelerómetro de lado derecho en ensamblador.

En cuanto a *send_left*, se lee de nuevo el estado del switch habilitador. Si da un estado lógico ‘1’, se leen los datos del acelerómetro. Al hacer una máscara con estos datos, si existen grupos de 2 bits con ‘1’, entonces se manda el valor en Unity para moverse hacia la izquierda. De otra manera, se regresa al ciclo *main*.

```

send_left:  inp r1, SW_Int
            inp r7, int_mov
            and r1, r1, 1
            bz main

            and r2, r7, 0xC0
            and r5, r7, 0x30
            or r6, r2, r5
            bz main
            ldm r3, left_data
            out r3, TX_DATA
            out r0, TX_START
            ldm r3, tx_disable
            out r3, TX_START
            jmp main

```

Figura 61. Función del acelerómetro de lado izquierdo en ensamblador.

VIII. A. e. i. Prueba de funcionamiento de *acelerómetro*

Para verificar el funcionamiento del acelerómetro se implementó el uso de los LEDS del FPGA. Estos se encenderán dependiendo de la dirección que se registre gracias al sensor.

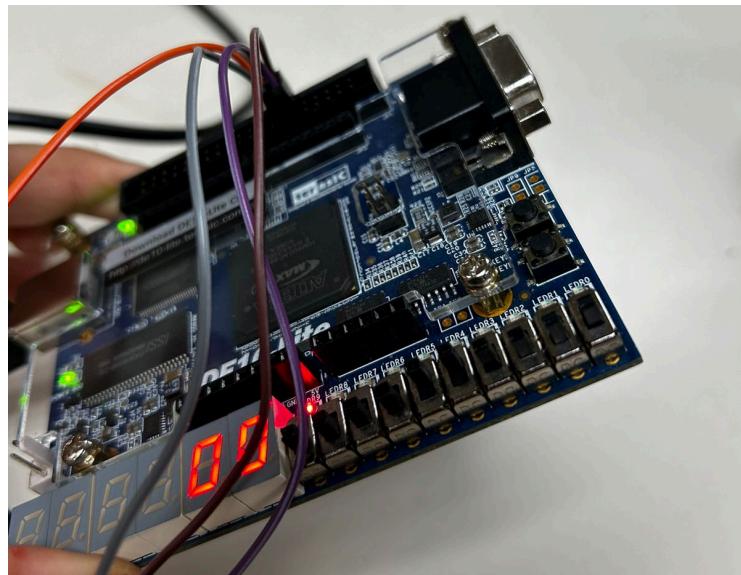


Figura 62. LED 09 encendido por inclinación a la izquierda.

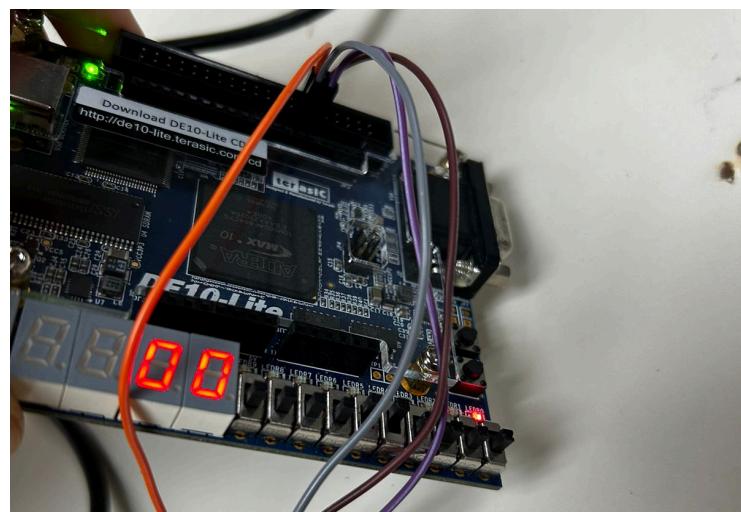


Figura 63. LED 00 encendido por inclinación a la derecha.

IX. RTL del prototipo

El diagrama RTL del prototipo ayuda a poder tener una mejor idea de cómo las distintas entidades están interactuando con los puertos y variables dentro del programa. En breve, se explican las partes esenciales de este mismo.

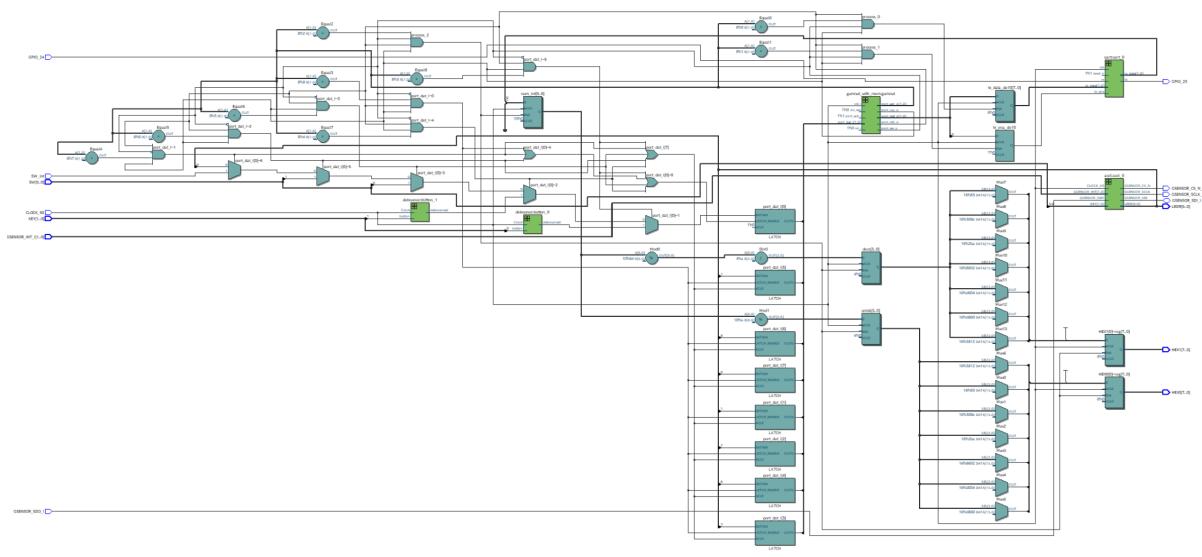


Figura 64. Diagrama RTL del proyecto.

IX. A. debounce.vhd

A la entidad *debounce* le están entrando dos datos: el reloj que se está usando durante el proceso y el estado del botón sin modificación alguna. La salida de esta entidad es un botón que ha sido modificado para evitar oscilaciones erróneas dentro de la lectura del estado del botón, así como una interpretación más lógica del estado de cada botón.

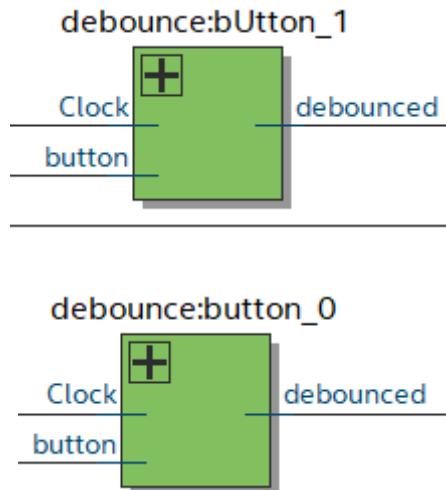


Figura 65. Entidades “debounce”.

IX. B. gumnut_with_mem.vhd

La entidad *gumnut_with_mem* tiene distintas entradas: *clk_i*, *rst_i*, *port_ack_i*, *port_dat_i* e *int_req*. Todo esto es para tener una mejor administración sobre lo que está ocurriendo al momento de tener una operación de I/O. Las salidas son *port_adr_o*, *port_cyc_o*, *port_dat_o*, *port_stb_o* y *port_we_o*, que tienen las direcciones de cada componente a conectarse con

ensamblador, los datos en forma de salidas, el habilitador de operaciones de escritura, entre otras funciones.

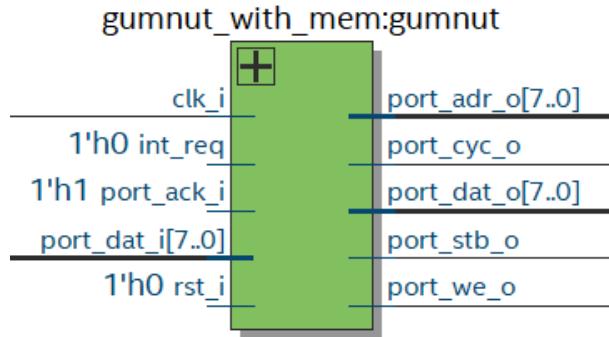


Figura 66. Entidad “gumnut_with_mem”.

IX. C. *uart.vhd*

La UART tiene cinco entradas: *clk*, *reset_n*, *rx*, *tx_data* y *tx_ena*. De aquí es donde se establece el reloj sobre el que trabaja la UART, el iniciador de transmisión, el GPIO de recepción, entre otros apartados que se han mencionado a lo largo del reporte. Como salidas están el GPIO de transmisión y los datos de recepción.

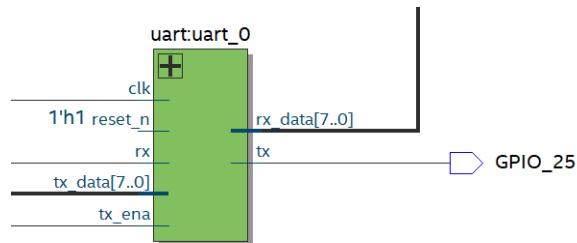


Figura 67. Entidad “uart”.

IX. D. *acel.vhd*

El acelerómetro necesita de ciertas entradas para poder funcionar, como son switches habilitadores y el reloj en funcionamiento con el proceso. Las salidas son los GPIOs que se asignan a la placa, así como los LEDs, que muestran los datos que está obteniendo el acelerómetro dependiendo de la orientación de la DE10-Lite.

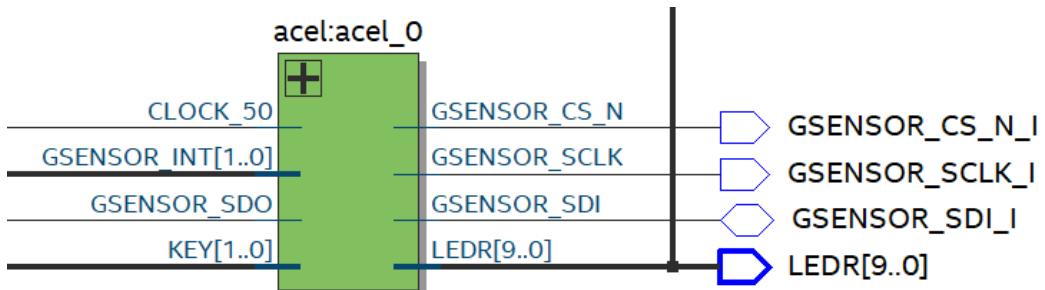


Figura 68. Entidad “acel”.

X. Conclusiones individuales

Ximena Trejo

La simulación en videojuegos ha sido una prueba definitiva sobre la posibilidad de incorporar distintos productos que históricamente han pertenecido a la industria del entretenimiento con implementaciones de situaciones de la vida real. En este caso, fue con el uso de un tractor de la empresa John Deere.

El uso de un FPGA ayudó a poder combinar conocimientos de lógica combinacional con programación de VHDL, pudiendo incluso desarrollar una interfaz para intercambiar datos. Además, con esto se llegó a tener un mejor conocimiento sobre esto a través de la práctica. Las competencias adquiridas ayudan a tener una mejor base para los siguientes semestres. Al final, gracias a esto se llegó al producto final esperado, y a la adquisición de habilidades.

Valeria Meneses

Durante este proyecto, hemos integrado exitosamente videojuegos en un entorno educativo, utilizando hardware (FPGA) y software (Gumnut) para desarrollar un videojuego educativo. Este enfoque nos ha permitido mejorar el aprendizaje y la adquisición de habilidades, en particular en la implementación de lógica de programas en alto y bajo nivel, así como en la creación de videojuegos en 3D. Además, hemos aprendido lecciones valiosas sobre la importancia de la colaboración y la comunicación efectiva dentro de un equipo.

Estoy muy satisfecha con los resultados de este proyecto. No sólo hemos desarrollado un videojuego educativo innovador, sino que también hemos adquirido habilidades técnicas y creativas valiosas. Estas habilidades y lecciones aprendidas serán de gran valor en nuestros futuros proyectos en el campo de la informática y el desarrollo de juegos. Estoy emocionada por aplicar lo que he aprendido en este curso a mis futuros proyectos.

Alondra Caspeta

Durante el desarrollo de este proyecto tuve la oportunidad de aprender sobre nuevos componentes de hardware (el uso de la FPGA y el FTDI) y software, esto con el propósito de crear un videojuego para nuestro socio formador John Deere. La clase me ayudó a reforzar conocimientos previos, como lo es la programación en ensamblador y el establecer una comunicación serial para el envío y recibimiento de datos en nuestra placa. También, está el hecho de aprender nuevos lenguajes de programación como lo es VHDL, que al final es otra herramienta de desarrollo que en un futuro puede llegar a ser útil para otros proyectos.

Considero que esta fue una experiencia muy enriquecedora y retadora, sobre todo la parte del uso de Gumnut para que todos nuestros componentes pudieran interactuar mediante el procesador. Me gustó mucho el resultado que obtuvimos y que logramos alcanzar el objetivo de nuestro proyecto.

Nahomi Velazquez

Al concluir este reto, se ha adquirido entendimiento de los sistemas ciber-físicos, donde hay una unión entre el software y el hardware, particularmente en el caso del FPGA, ha permitido fusionar lenguajes de programación de alto y bajo nivel. Esta integración ha sido fundamental para la configuración y programación del procesador, desempeñando un papel relevante en su implementación dentro del contexto del juego.

La experiencia ha sido enriquecedora, ya que se ha presenciado cómo diversos conceptos y aprendizajes se entrelazan y complementan entre sí para lograr un resultado final satisfactorio. Además se reforzó el concepto de la comunicación serial para el recibimiento y la transmisión de datos. Cada uno de los aspectos han contribuido de manera significativa al éxito del proyecto.

XI. Videos demostrativos

Video demostrativo - Etapa 1

https://drive.google.com/file/d/1QITHrL9UQg9j_3kSgtiV3Lb-AYcM20_q/view?usp=sharing

Video demostrativo - Etapa 2

https://drive.google.com/file/d/1gE2scJQUdNTKj6An8Y7ehP445kFdy3_E/view?usp=sharing

Video demostrativo - Etapa 3

https://drive.google.com/file/d/1haL83-SoomdrZzMopLGrzALa0R_oFSIL/view?usp=sharing

XII. Referencias

- Saravia, Numa. (febrero 14, 2024). Los videojuegos y su impacto en la industria del entretenimiento. *Diario El Transmisor*. Recuperado el 28 de marzo de 2024 de <https://eltransmisor.com/2024/02/14/los-videojuegos-y-su-impacto-en-la-industria-del-entretenimiento/>
- The Conversation. (junio 10, 2017). El ejército de EEUU está utilizando videojuegos “caóticos, preciosos y violentos” para entrenar a los soldados. *Xataka*. Recuperado el 28 de marzo de 2024 de <https://www.xataka.com/videojuegos/el-ejercito-de-eeuu-esta-utilizando-videojuegos-caoticos-preciosos-y-violentos-para-entrenar-a-los-soldados>
- García-Bullé, S. (enero 25, 2019). Videojuegos: Una herramienta educativa en potencia. *Instituto para el Futuro de la Educación*. Recuperado el 28 de marzo de 2024 de <https://observatorio.tec.mx/edu-news/juegos-y-educacion/>

XIII. Anexos

Etapa 2

Anexo 1. Código de de10_lite.vhd

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY de10_lite IS
  PORT( CLOCK_50 : IN std_logic;
        KEY : IN std_logic_vector(1 DOWNTO 0);
        SW : IN std_logic_vector(9 DOWNTO 0);
        GPIO_24 : IN std_logic; --RX
        GPIO_25 : OUT std_logic; --TX
        LEDR : OUT std_logic_vector(9 DOWNTO 0);
        HEX0 : OUT std_logic_vector(7 DOWNTO 0)
      );
END;

ARCHITECTURE Structural OF de10_lite IS

COMPONENT uart IS
  GENERIC(
    clk_freq : integer := 50_000_000; --frequency of system clock in Hertz
    baud_rate : integer := 115_200; --data link baud rate in bits/second
    os_rate : integer := 16; --oversampling rate to find center of receive bits
    --(in samples per baud period)
    d_width : integer := 8; --data bus width
    parity : integer := 0; --0 for no parity, 1 for parity
    parity_eo : std_logic := '0'); --'0' for even, '1' for odd parity
  PORT(
    clk : IN std_logic; --system clock
    reset_n : IN std_logic; --asynchronous reset
    tx_ena : IN std_logic; --initiate transmission
    tx_data : IN std_logic_vector(d_width-1 DOWNTO 0); --data to transmit
    rx : IN std_logic; --receive pin
    rx_busy : OUT std_logic; --data reception in progress, LEDR(9)
    rx_error : OUT std_logic; --start, parity, or stop bit error detected
    rx_data : OUT std_logic_vector(d_width-1 DOWNTO 0); --data received
    tx_busy : OUT std_logic; --transmission in progress, LEDR(8)
    tx : OUT std_logic); --transmit pin
END COMPONENT;

COMPONENT debounce IS
  PORT ( Clock : IN STD_LOGIC;
         button : IN STD_LOGIC;
         debounced : BUFFER STD_LOGIC);
END COMPONENT;

COMPONENT bcd7seg_sec IS
  PORT ( bcd : in std_logic_vector(3 downto 0);
         display : out std_logic_vector(7 downto 0)
       );
end COMPONENT;

SIGNAL tx_ena_de10: std_logic := '1';
SIGNAL tx_busy_de10: std_logic;
SIGNAL tx_data_de10: std_logic_vector(7 DOWNTO 0);

SIGNAL rx_busy_de10: std_logic;
SIGNAL rx_error_de10: std_logic;
SIGNAL rx_data_de10: std_logic_vector(7 DOWNTO 0);

--Buttons
SIGNAL key0_db: std_logic;
SIGNAL key0_db_past: std_logic := '0';
SIGNAL key1_db: std_logic;
SIGNAL key1_db_past: std_logic := '0';

--Display
SIGNAL display : std_logic_vector(7 DOWNTO 0);
SIGNAL bcd : std_logic_vector(3 DOWNTO 0);

BEGIN
  --Show in the LEDs the received data
  LEDR <= "00" & rx_data_de10;

  uart_0 : uart PORT MAP( CLOCK_50, SW(9), tx_ena_de10, tx_data_de10, GPIO_24, rx_busy_de10, rx_error_de10, rx_data_de10, tx_busy_de10, GPIO_25 );
  button_0 : debounce PORT MAP( CLOCK_50, KEY(0), key0_db );
  button_1 : debounce PORT MAP( CLOCK_50, KEY(1), key1_db );
  display/seg : bcd7seg_sec PORT MAP( rx_data_de10(3 DOWNTO 0), HEX0);

```

```

--Process to receive
PROCESS(CLOCK_50)
BEGIN
    IF(rising_edge (CLOCK_50)) THEN
        bcd <= rx_data_de10(3 downto 0);
    END IF;
END PROCESS;

--Process to transmit data according to the interface inputs
PROCESS( CLOCK_50 )
BEGIN
    IF rising_edge( CLOCK_50 ) THEN
        IF SW(0) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000101"; -- Send value 5 when switch 0 is on FORWARD
        ELSIF SW(1) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000110"; -- Send value 6 when switch 1 is on BACKWARD
        ELSIF SW(2) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000010"; -- Send value 2 when switch 2 is on RIGHT
        ELSIF SW(3) = '1' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000011"; -- Send value 3 when switch 3 is on LEFT
        ELSIF key0_db = '1' and key0_db_past = '0' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000100"; -- Send value 4 when button is pressed
        ELSIF key1_db = '1' and key1_db_past = '0' THEN
            tx_ena_de10 <= '0';
            tx_data_de10 <= "00000001"; -- Send value 1 when button is pressed
        ELSE
            tx_ena_de10 <= '1';
            tx_data_de10 <= "00000000"; -- Send value 0 when neither switch 1 is on nor button is pressed
        END IF;
        key0_db_past <= key0_db;
        key1_db_past <= key1_db;
    END IF;
END PROCESS;

```

Anexo 2. Código de bcd7seg_sec.vhd

```

library ieee;
use ieee.std_logic_1164.all;

entity bcd7seg_sec is
    port(
        bcd: in std_logic_vector(3 downto 0);
        display: out std_logic_vector(7 downto 0)
    );
end entity;

architecture behavior of bcd7seg_sec is
begin
    process (bcd)
    begin
        case bcd is
            when "0000" =>
                display <= "11000000";
            when "0001" =>
                display <= "11111001";
            when "0010" =>
                display <= "10100100";
            when "0011" =>
                display <= "10110000";
            when "0100" =>
                display <= "10011001";
            when "0101" =>
                display <= "10010010";
            when "0110" =>
                display <= "10000010";
            when "0111" =>
                display <= "11111000";
            when "1000" =>
                display <= "10000000";
            when "1001" =>
                display <= "10011000";
            when "1010" =>
                display <= "10001000";
            when "1011" =>
                display <= "10000011";
            when "1100" =>
                display <= "10100111";
            when "1101" =>
                display <= "10100001";
            when "1110" =>
                display <= "10000110";
            when "1111" =>
                display <= "11111111";
        end case;
    end process;
end architecture;

```

```

when others =>
    display <= "10001110";
end case;

end process;

end architecture;

```

Anexo 3. Código de uart.vhd

```

25 LIBRARY ieee;
26 USE ieee.std_logic_1164.all;
27
28 ENTITY uart IS
29   GENERIC(
30     clk_freq : INTEGER      := 50_000_000; --frequency of system clock in Hertz
31     baud_rate : INTEGER     := 115_200;   --data link baud rate in bits/second
32     os_rate   : INTEGER     := 16;        --oversampling rate to find center of receive bits (in samples per baud period)
33     d_width   : INTEGER     := 8;         --data bus width
34     parity    : INTEGER     := 0;         --0 for no parity, 1 for parity
35     parity_eo : STD_LOGIC   := '0');    --'0' for even, '1' for odd parity
36
37 PORT(
38   clk      : IN STD_LOGIC;           --system clock
39   reset_n  : IN STD_LOGIC;          --asynchronous reset
40   tx_ena   : IN STD_LOGIC;          --initiate transmission
41   tx_data  : IN STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --data to transmit
42   rx       : IN STD_LOGIC;          --receive pin
43   rx_busy  : OUT STD_LOGIC;         --data reception in progress
44   rx_error : OUT STD_LOGIC;         --start, parity, or stop bit error detected
45   rx_data  : OUT STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --data received
46   tx_busy  : OUT STD_LOGIC;         --transmission in progress
47   tx       : OUT STD_LOGIC);        --transmit pin
48
49 ARCHITECTURE logic OF uart IS
50   TYPE tx_machine IS(idle, transmit);      --transmit state machine data type
51   TYPE rx_machine IS(idle, receive);        --receive state machine data type
52   SIGNAL tx_state   : tx_machine;           --transmit state machine
53   SIGNAL rx_state   : rx_machine;           --receive state machine
54   SIGNAL baud_pulse : STD_LOGIC := '0';     --periodic pulse that occurs at the baud rate
55   SIGNAL os_pulse   : STD_LOGIC := '0';     --periodic pulse that occurs at the oversampling rate
56   SIGNAL parity_error : STD_LOGIC;          --receive parity error flag
57   SIGNAL rx_parity  : STD_LOGIC_VECTOR(d_width DOWNTO 0); --calculation of receive parity
58   SIGNAL tx_parity  : STD_LOGIC_VECTOR(d_width DOWNTO 0); --calculation of transmit parity
59   SIGNAL rx_buffer  : STD_LOGIC_VECTOR(parity+d_width DOWNTO 0) := (OTHERS => '0'); --values received
60   SIGNAL tx_buffer  : STD_LOGIC_VECTOR(parity+d_width-1 DOWNTO 0) := (OTHERS => '1'); --values to be transmitted
61
62 BEGIN
63   --generate clock enable pulses at the baud rate and the oversampling rate
64   PROCESS(reset_n, clk)
65     VARIABLE count_baud : INTEGER RANGE 0 TO clk_freq/baud_rate-1 := 0; --counter to determine baud rate period
66     VARIABLE count_os   : INTEGER RANGE 0 TO clk_freq/baud_rate/os_rate-1 := 0; --counter to determine oversampling period
67   BEGIN
68     IF(reset_n = '0') THEN
69       baud_pulse <= '0';
70       os_pulse <= '0';
71       count_baud := 0;
72       count_os := 0;
73     ELSIF(clk'EVENT AND clk = '1') THEN
74       --create baud enable pulse
75       IF(count_baud < clk_freq/baud_rate-1) THEN
76         count_baud := count_baud + 1;
77         baud_pulse <= '0';
78       ELSE
79         count_baud := 0;
80         baud_pulse <= '1';
81         count_os := 0;
82       END IF;
83       --create oversampling enable pulse
84       IF(count_os < clk_freq/baud_rate/os_rate-1) THEN
85         count_os := count_os + 1;
86         os_pulse <= '0';
87       ELSE
88         count_os := 0;
89         os_pulse <= '1';
90       END IF;
91     END IF;
92   END PROCESS;

```

```

94  --receive state machine
95  PROCESS(reset_n, clk)
96  VARIABLE rx_count : INTEGER RANGE 0 TO parity+d_width+2 := 0; --count the bits received
97  VARIABLE os_count : INTEGER RANGE 0 TO os_rate-1 := 0; --count the oversampling rate pulses
98 BEGIN
99  IF(reset_n = '0') THEN
100   os_count := 0;
101   rx_count := 0;
102   rx_busy <= '0';
103   rx_error <= (OTHERS => '0');
104   rx_data <= (OTHERS => '0');
105   rx_state <= idle;
106   ELSIF(clk'EVENT AND clk = '1' AND os_pulse = '1') THEN --enable clock at oversampling rate
107   CASE rx_state IS
108    WHEN idle =>
109     rx_busy <= '0';
110     IF(rx = '0') THEN
111      IF(os_count < os_rate/2) THEN
112        os_count := os_count + 1;
113        rx_state <= idle;
114      ELSE
115        os_count := 0;
116        rx_count := 0;
117        rx_busy <= '1';
118        rx_buffer <= rx & rx_buffer(parity+d_width DOWNTO 1);
119        rx_state <= receive;
120      END IF;
121    ELSE
122      os_count := 0;
123      rx_state <= idle;
124    END IF;
125    WHEN receive =>
126     IF(os_count < os_rate-1) THEN
127       os_count := os_count + 1;
128       rx_state <= receive;
129     ELSIF(rx_count < parity+d_width) THEN
130       os_count := 0;
131       rx_count := rx_count + 1;
132       rx_buffer <= rx & rx_buffer(parity+d_width DOWNTO 1);
133       rx_state <= receive;
134     ELSE
135       rx_data <= rx_buffer(d_width DOWNTO 1);
136       rx_error <= rx_buffer(0) OR parity_error OR NOT rx;
137       rx_busy <= '0';
138       rx_state <= idle;
139     END IF;
140   END CASE;
141 END IF;

144  --receive parity calculation logic
145  rx_parity(0) <= parity_eo;
146  rx_parity_logic: FOR i IN 0 to d_width-1 GENERATE
147   rx_parity(i+1) <= rx_parity(i) XOR rx_buffer(i+1);
148  END GENERATE;
149  WITH parity SELECT --compare calculated parity bit with received parity bit to determine error
150   parity_error <= rx_parity(d_width) XOR rx_buffer(parity+d_width) WHEN 1, --using parity
151   '0' WHEN OTHERS; --not using parity

153  --transmit state machine
154  PROCESS(reset_n, clk)
155  VARIABLE tx_count : INTEGER RANGE 0 TO parity+d_width+3 := 0; --count bits transmitted
156 BEGIN
157  IF(reset_n = '0') THEN
158   tx_count := 0;
159   tx <= '1';
160   tx_busy <= '1';
161   tx_state <= idle;
162   ELSIF(clk'EVENT AND clk = '1') THEN
163   CASE tx_state IS
164    WHEN idle =>
165     IF(tx_ena = '0') THEN
166       tx_buffer(d_width+1 DOWNTO 0) <= tx_data & '0' & '1'; --latch in data for transmission and start/stop bits
167       IF(parity = 1) THEN
168         tx_buffer(parity+d_width+1) <= tx_parity(d_width); --if parity is used
169       END IF;
170       tx_busy <= '1';
171       tx_count := 0;
172       tx_state <= transmit;
173     ELSE
174       tx_busy <= '0';
175       tx_state <= idle;
176     END IF;
177    WHEN transmit =>
178     IF(baud_pulse = '1') THEN
179       tx_count := tx_count + 1;
180       tx_buffer <= '1' & tx_buffer(parity+d_width+1 DOWNTO 1); --shift transmit buffer to output next bit
181     END IF;
182     IF(tx_count < parity+d_width+3) THEN
183       tx_state <= transmit;
184     ELSE
185       tx_state <= idle;
186     END IF;
187   END CASE;
188   tx <= tx_buffer(0);
189 END IF;
190 END PROCESS;

191
192  --transmit parity calculation logic
193  tx_parity(0) <= parity_eo;
194  tx_parity_logic: FOR i IN 0 to d_width-1 GENERATE
195   tx_parity(i+1) <= tx_parity(i) XOR tx_data(i);
196  END GENERATE;
197
198 END logic;

```

Anexo 4. Código de debounce.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

-- This module can be used to debounce a KEY on the DE10-Lite board
ENTITY debounce IS
    PORT ( Clock      : IN      STD_LOGIC;
           button     : IN      STD_LOGIC;
           debounced  : BUFFER  STD_LOGIC);
END debounce;

ARCHITECTURE Behavior OF debounce IS
    SIGNAL COUNT : STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL done, counting : STD_LOGIC;
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND clock = '1';
        IF (done = '1' AND button = '1') THEN -- When counter expires and button is no longer
            debounced <= '0';                -- pressed, then set the debounced output to 0
        ELSIF (button = '0') THEN           -- When the "button" is pressed
            debounced <= '1';                -- set the debounced output to 1
        END IF;
    END PROCESS;

    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND clock = '1';
        IF (done = '1') THEN
            Count <= "000";
        ELSIF (debounced = '1') THEN
            Count <= Count + '1';
        END IF;
    END PROCESS;
    done <= '1' WHEN Count = "111" ELSE '0';
END Behavior;
```

Anexo 5. Código de de10.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System;
6  using System.IO.Ports;
7  using TMPro;
8
9  // Script de Unity (1 referencia de recurso) | 1 referencia
10 public class de10 : MonoBehaviour
11 {
12     public int Recio = 0;
13     public int Break = 0;
14     public int Right = 0;
15     public int Left = 0;
16     public int Backward = 0;
17     public int Forward = 0;
18     public SerialPort de10l = new SerialPort("COM4", 115200);
19
20     private TextMeshProUGUI CornText;
21     public PlayerInventory playerInventory;
22
23     // Mensaje de Unity | 0 referencias
24     void Start()
25     {
26         CornText = GetComponent<TextMeshProUGUI>();
27         if (!de10l.IsOpen)
28         {
29             de10l.Open();
30             de10l.ReadTimeout = 1;
31         }
32         if (de10l.IsOpen)
33         {
34             var dataByte = new byte[] { 0x00 };
35             de10l.Write(dataByte, 0, 1);
36         }
37     }
38 }
```

```

39     void Update()
40     {
41         if (!de10l.IsOpen)
42         {
43             de10l.Open();
44             de10l.ReadTimeout = 1;
45         }
46
47         if (de10l.IsOpen)
48         {
49             try
50             {
51                 int value;
52                 if (de10l.BytesToRead > 0)
53                 {
54                     value = de10l.ReadByte();
55                     de10l.DiscardInBuffer();
56                 }
57                 else
58                 {
59                     value = 0;
60                 }
61
62                 if (value == 0x01)
63                 {
64                     Recio = 1;
65                 }
66                 else if (value == 0x02)
67                 {
68                     Right = 1;
69                 }
70                 else if (value == 0x03)
71                 {
72                     Left = 1;
73                 }
74                 else if (value == 0x04)
75                 {
76                     Break = 1;
77                 }
78                 else if (value == 0x05)
79                 {
80                     Forward = 1;
81                     Break = 0;
82                 }
83             }
84             else if (value == 0x06)
85             {
86                 Backward = 1;
87                 Break = 0;
88             }
89             else
90             {
91                 Recio = 0;
92                 Right = 0;
93                 Left = 0;
94                 Forward = 0;
95                 Backward = 0;
96             }
97
98             // Contador de maíces
99             int MyInt = playerInventory.NumberOfCorns;
100            byte[] b = BitConverter.GetBytes(MyInt);
101            de10l.Write(b, 0, 1);
102        }
103    }
104
105
106
107    catch { }
108
109

```

Anexo 6. Código de Recio.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  3 referencias
6  public class Recio {
7
8      1 referencia
9      public Recio() {}
10
11     1 referencia
12     public void Boost(GameObject tractor, float boostForce)
13     {
14         Rigidbody R = tractor.GetComponent<Rigidbody>();
15         R.AddForce(tractor.transform.forward * boostForce);
16     }
17 }
```

Anexo 7. Código de carController.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  ① Script de Unity (1 referencia de recurso) | 0 referencias
6  public class WheelController : MonoBehaviour
7  {
8      public de10 de10l;
9      // Start is called before the first frame update
10     [SerializeField] WheelCollider frontRight;
11     [SerializeField] WheelCollider backRight;
12     [SerializeField] WheelCollider frontLeft;
13     [SerializeField] WheelCollider backLeft;
14
15     [SerializeField] Transform frontRightTransform;
16     [SerializeField] Transform frontLeftTransform;
17     [SerializeField] Transform SteeringWheel;
18
19     public float aceleracion = 500f;
20     public float fuerzaFreno = 300f;
21     public float anguloMaximo = 15f;
22     public float boost = 250f; // Add boostForce here
23
24     public float aAceleracion = 0f;
25     public float aFuerzaFreno = 0f;
26     public float aAnguloMaximo = 0f;
27 }
```

```

30     public void FixedUpdate()
31     {
32
33         // boost
34         if (de10l.Recio == 1)
35         {
36             Recio n = new();
37             n.Boost(gameObject, boost);
38         }
39
40         if (de10l.Forward == 1)
41         {
42             aAceleracion = aceleracion * de10l.Forward;
43         }
44         // Para moverse hacia atrás
45         else if (de10l.Backward == 1)
46         {
47             aAceleracion = -aceleracion; // Nota: Puedes ajustar el signo según la dirección deseada
48         }
49         else
50         {
51             aAceleracion = 0;
52         }
53
54         // Control de la dirección
55
56
57         if (de10l.Left == 1)
58         {
59             aAnguloMaximo = -anguloMaximo; // Girar a la izquierda con el máximo ángulo permitido
60         }
61         else if (de10l.Right == 1)
62         {
63             aAnguloMaximo = anguloMaximo * de10l.Right;
64         }
65

```

```

66
67         if (de10l.Break == 1)
68         {
69             aFuerzaFreno = fuerzaFreno*900000000000000000000000f;
70         }
71         else
72         {
73             aFuerzaFreno = 0;
74         }
75
76         frontRight.motorTorque = aAceleracion;
77         frontLeft.motorTorque = aAceleracion;
78
79         frontRight.brakeTorque = aFuerzaFreno;
80         frontRight.brakeTorque = aFuerzaFreno;
81         backRight.brakeTorque = aFuerzaFreno;
82         backLeft.brakeTorque = aFuerzaFreno;
83
84
85         //currentTurnAngle = maxTurnAngle * Input.GetAxis("Horizontal");
86         frontLeft.steerAngle = aAnguloMaximo;
87         frontRight.steerAngle = aAnguloMaximo;
88
89         UpdateWheel(frontLeft, frontLeftTransform);
90         UpdateWheel(frontRight, frontRightTransform); ;
91
92     }
93
94 }
95
96
97
98
99
100
101
102
103
104

```

2 referencias

```

void UpdateWheel(WheelCollider col, Transform trans)
{
    Vector3 position;
    Quaternion rotation;
    col.GetWorldPose(out position, out rotation);

    trans.position = position;
    trans.rotation = rotation;
}

```

```

70     if (de10l.Break == 1)
71     {
72         aFuerzaFreno = fuerzaFreno*90000000000000000000000f;
73     }
74     else
75     {
76         aFuerzaFreno = 0;
77     }
78
79     frontRight.motorTorque = aAceleracion;
80     frontLeft.motorTorque = aAceleracion;
81
82     frontRight.brakeTorque = aFuerzaFreno;
83     frontRight.brakeTorque = aFuerzaFreno;
84     backRight.brakeTorque = aFuerzaFreno;
85     backLeft.brakeTorque = aFuerzaFreno;
86
87     //currentTurnAngle = maxTurnAngle * Input.GetAxis("Horizontal");
88     frontLeft.steerAngle = aAnguloMaximo;
89     frontRight.steerAngle = aAnguloMaximo;
90
91     UpdateWheel(frontLeft, frontLeftTransform);
92     UpdateWheel(frontRight, frontRightTransform); ;
93
94 }
95
96     2 referencias
97     void UpdateWheel(WheelCollider col, Transform trans)
98     {
99         Vector3 position;
100        Quaternion rotation;
101        col.GetWorldPose(out position, out rotation);
102
103        trans.position = position;
104        trans.rotation = rotation;
105    }
106
107     0 referencias
108     void UpdateSteeringWheel(Transform steering)
109     {
110         Vector3 customRotationAxis = transform.position.normalized;
111
112         Quaternion rot = Quaternion.AngleAxis(aAnguloMaximo, customRotationAxis.normalized);
113
114         transform.rotation *= rot;
115     }
116
117 }
```

Anexo 8. Código de InventoryUI.cs

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System;
6  using System.IO.Ports;
7  using TMPro;
8
9      Script de Unity (4 referencias de recurso) | 0 referencias
10     public class InventoryUI : MonoBehaviour
11     {
12         public SerialPort de10l = new SerialPort("COM4", 115200);
13
14         private TextMeshProUGUI CornText;
15
16         // Start is called before the first frame update
17         Mensaje de Unity | 0 referencias
18         void Start()
19         {
20             CornText = GetComponent<TextMeshProUGUI>();
21
22             0 referencias
23             public void UpdateCornText(PlayerInventory playerInventory)
24             {
25                 // Actualiza el texto en Unity
26                 CornText.text = playerInventory.NumberOfCorns.ToString();
27             }
28         }
29     }
```

Anexo 9. Código de PlayerInventory.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  ...
5  using UnityEngine.Events;
6
7  public class PlayerInventory : MonoBehaviour
8  {
9      public int NumberOfCorns { get; private set; }
10
11     public UnityEvent<PlayerInventory> OnCornCollected;
12
13     public void CornCollected()
14     {
15         NumberOfCorns++;
16         OnCornCollected.Invoke(this);
17     }
18 }
```

Etapa 3

Anexo 10. Código en ensamblador

```
1  org 0x000 ; start here on reset
2  jmp main
3  ; Data memory layout
4  data
5  TX_DATA:    bss 1 ;0
6  TX_START:   bss 1 ;1
7  disp:        bss 1 ;2
8  KEY0:        bss 1 ;3
9  KEY1:        bss 1 ;4
10 SW0:         bss 1 ;5
11 SW1:         bss 1 ;6
12 SW_Int:      bss 1 ;7
13 int_mov:     bss 1 ;8
14 cont:        bss 1 ;9
15
16 ; Datos desde Unity
17 KEY0_data: byte 4
18 KEY1_data: byte 1
19 SW0_data: byte 5
20 SW1_data: byte 6
21 right_data: byte 2
22 left_data: byte 3
23 tx_disable: byte 1
24 SW_Aux:     byte 0
```

```

26 ; Main program
27     text
28     org 0x010
29
30 main: jsb counter_func ;saltar a funcion de contador
31     inp r0, SW_Aux ;obtenemos input de auxiliar de switch: ayuda a saber que entraremos a la secuencia
32     and r4, r0, 0
33     bnz main ;si no es 0, seguimos en main
34     jsb next_SW0; si es 0, saltamos a comprobar SW0
35
36 counter_func: inp r6, cont ;obtenemos el contador
37         out r6, disp ;lo desplegamos
38         ret
39
40 send_left: inp r1, SW_Int ;obtenemos el valor del switch que activa el acelerometro
41     inp r7, int_mov ;obtenemos los datos del acelerometro
42     and r1, r1, 1 ;corroboramos estado del switch de acelerometro
43     bz main ;si no esta levantado, ir a main
44     ; Check if (int_mov(7 downto 6) = "11" AND int_mov(9 downto 8) = "00") OR (int_mov(7 downto 6) = "00" AND int_mov(9 downto 8) = "11")
45     and r2, r7, 0x0C ; Mask bits 9 and 8
46     and r5, r7, 0x30 ; Mask bits 7 and 6
47     or r6, r2, r5 ; hacer OR para ir a la izquierda
48     bz main ; si no hay valores en la izquierda, ir a main
49     ldm r3, left_data ; si hay valores en la izquierda, mandar valor a Unity
50     out r3, TX_DATA
51     out r0, TX_START
52     ldm r3, tx_disable
53     out r3, TX_START
54     jmp main
55
56 next_acel: inp r1, SW_Int ;obtenemos el valor del switch que activa el acelerometro
57     inp r7, int_mov ;obtenemos los datos del acelerometro
58     and r1, r1, 1 ;corroboramos estado del switch de acelerometro
59     bz main ;si no esta levantado, ir a main
60     ; Check if (int_mov(1 downto 0) = "00" AND int_mov(3 downto 2) = "11") OR (int_mov(1 downto 0) = "11" AND int_mov(3 downto 2) = "00")
61     and r2, r7, 0x0C ; Mask bits 3 and 2
62     and r3, r7, 0x03 ; Mask bits 1 and 0
63     or r4, r2, r3 ; hacer OR para ir a la derecha
64     bz send_left ; si no hay valores en la derecha, ir a ver valores de izquierda
65     ldm r2, right_data ; si hay valores en la derecha, mandar valor a Unity
66     out r2, TX_DATA
67     out r0, TX_START
68     ldm r2, tx_disable
69     out r2, TX_START
70
71 next_KEY0: inp r2, KEY0 ; leemos estado de KEY0
72     and r2, r2, 1 ; corroboramos estado
73     bz next_KEY1 ; si es 0, leemos el siguiente KEY
74     ldm r3, KEY0_data ; si no es 0, mandamos el valor a Unity
75     out r3, TX_DATA
76     out r0, TX_START
77     ldm r3, tx_disable
78     out r3, TX_START
79
80 next_KEY1: inp r3, KEY1 ; leemos estado de KEY1
81     and r3, r3, 1 ; corroboramos estado
82     bz next_acel ; si es 0, leemos el acelerometro
83     ldm r4, KEY1_data ; si no es 0, mandamos el valor a Unity
84     out r4, TX_DATA
85     out r0, TX_START
86     ldm r4, tx_disable
87     out r4, TX_START
88
89 next_SW0: inp r4, SW0 ; leemos el estado de SW0
90     and r4, r4, 1 ; corroboramos estado
91     bz next_SW1 ;si es 0, leemos el siguiente switch
92     ldm r5, SW0_data ; si no es 0, mandamos el valor a Unity
93     out r5, TX_DATA
94     out r0, TX_START
95     ldm r5, tx_disable
96     out r5, TX_START
97     jmp main
98
99 next_SW1: inp r5, SW1 ; leemos el estado de SW1
100    and r5, r5, 1 ; corroboramos estado
101    bz next_KEY0 ;si es 0, leemos el siguiente switch
102    ldm r6, SW1_data ; si no es 0, mandamos el valor a Unity
103    out r6, TX_DATA
104    out r0, TX_START
105    ldm r6, tx_disable
106    out r6, TX_START

```

Anexo 11. Código de de10_lite.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use IEEE.NUMERIC_STD.ALL;

ENTITY de10_lite IS
    PORT( CLOCK_50      : IN  std_logic;
          KEY           : IN  std_logic_vector(1 DOWNTO 0);
          SW             : IN  std_logic_vector(9 DOWNTO 0);
          GPIO_24        : IN  std_logic; --RX
          GPIO_25        : OUT std_logic; --TX
          LEDR          : OUT std_logic_vector(9 DOWNTO 0);
          HEX0          : OUT std_logic_vector(7 DOWNTO 0);
          HEX1          : OUT std_logic_vector(7 DOWNTO 0);
          SW_Int         : IN  std_logic;
          GSENSOR_INT_I : IN  std_logic_vector(1 DOWNTO 0);
          GSENSOR_SDII_I : inout std_logic;
          GSENSOR_SDOI_I : inout std_logic;
          GSENSOR_CS_N_I : OUT std_logic;
          GSENSOR_SCLK_I : OUT std_logic
        );
END;

ARCHITECTURE Structural OF de10_lite IS

COMPONENT uart IS
    GENERIC(
        clk_freq      : integer      := 50_000_000; --frequency of system clock in Hertz
        baud_rate    : integer      := 115_200; --data link baud rate in bits/second
        os_rate       : integer      := 16; --oversampling rate to find center of receive bits
        --(in samples per baud period)
        d_width       : integer      := 8; --data bus width
        parity        : integer      := 0; --0 for no parity, 1 for parity
        parity_eo     : std_logic   := '0'); --'0' for even, '1' for odd parity
    PORT(
        clk          : IN  std_logic; --system clock
        reset_n      : IN  std_logic; --asynchronous reset
        tx_ena       : IN  std_logic; --initiate transmission
        tx_data      : IN  std_logic_vector(d_width-1 DOWNTO 0); --data to transmit
        rx           : IN  std_logic; --receive pin
        rx_busy      : OUT std_logic; --data reception in progress, LEDR(9)
        rx_error     : OUT std_logic; --start, parity, or stop bit error detected
        rx_data      : OUT std_logic_vector(d_width-1 DOWNTO 0); --data received
        tx_busy      : OUT std_logic; --transmission in progress, LEDR(8)
        tx           : OUT std_logic); --transmit pin
    END COMPONENT;

component gumnut_with_mem IS
    generic(
        IMem_file_name : string      := "gasm_text.dat";
        DMem_file_name : string      := "gasm_data.dat";
        debug         : boolean     := false
    );
    port(
        clk_i         : in  std_logic;
        rst_i         : in  std_logic;
        -- I/O port bus
        port_cyc_o   : out std_logic;
        port_stb_o   : out std_logic;
        port_we_o    : out std_logic;
        port_ack_i   : in  std_logic;
        port_adr_o   : out unsigned(7 downto 0);
        port_dat_o   : out std_logic_vector(7 downto 0);
        port_dat_i   : in  std_logic_vector(7 downto 0);
        -- Interrupts
        int_req      : in  std_logic;
        int_ack      : out std_logic
    );
end component gumnut_with_mem;

--Componentes para instanciar
--Componente de debounce de push button
COMPONENT debounce IS
    PORT ( Clock      : IN      STD_LOGIC;
           button    : IN      STD_LOGIC;
           debounced : BUFFER  STD_LOGIC);
END COMPONENT;
```

```

--Componente de acelerómetro
COMPONENT acel IS
  PORT( CLOCK_50 : IN std_logic;
        KEY      : IN std_logic_vector(1 DOWNTO 0);
        GSENSOR_INT : IN std_logic_vector(1 DOWNTO 0);
        GSENSOR_SDI : INOUT std_logic;
        GSENSOR_SDO : INOUT std_logic;
        GSENSOR_CS_N : OUT std_logic;
        GSENSOR_SCLK : OUT std_logic;
        LEDR      : OUT std_logic_vector(9 DOWNTO 0)
  );
END COMPONENT;

--Señales internas para instanciar
SIGNAL clk_i, rst_i          : std_logic;
SIGNAL port_cyc_o, port_stb_o : std_logic;
SIGNAL port_we_o, port_ack_i : std_logic;
SIGNAL int_req, int_ack      : std_logic;
SIGNAL port_adr_o            : unsigned(7 DOWNTO 0);
SIGNAL port_dat_o, port_dat_i : std_logic_vector(7 DOWNTO 0);

SIGNAL tx_ena_de10:    std_logic := '1';
SIGNAL tx_busy_de10:   std_logic;
SIGNAL tx_data_de10:   std_logic_vector(7 DOWNTO 0);

SIGNAL rx_busy_de10:   std_logic;
SIGNAL rx_error_de10:  std_logic;
SIGNAL rx_data_de10:   std_logic_vector(7 DOWNTO 0);

--Botones
SIGNAL key0_db:      std_logic;
SIGNAL key0_db_past: std_logic := '0';
SIGNAL key1_db:      std_logic;
SIGNAL key1_db_past: std_logic := '0';

--Display
SIGNAL display : std_logic_vector(7 DOWNTO 0);

SIGNAL bcd      : std_logic_vector(3 DOWNTO 0);
signal int_mov : STD_LOGIC_VECTOR(9 DOWNTO 0);
signal int_mov_modified: std_logic_vector(7 downto 0);
signal unids, decs : STD_LOGIC_VECTOR(3 downto 0);
signal num_int : integer range 0 to 999;

BEGIN
  --Si rst_i es 0, entonces siempre habrá un flujo de datos en Gummnut
  rst_i <= '0';

  --Instanciación de entidades: Gummnut, UART, debounce de botones y acelerómetro
  gummnut : gummnut_with_mem PORT MAP(CLOCK_50, rst_i, port_cyc_o, port_stb_o,
  port_we_o, '1', port_adr_o, port_dat_o, port_dat_i, int_req, int_ack);
  uart_0 : uart PORT MAP(CLOCK_50, '1', tx_ena_de10, tx_data_de10, GPIO_24, rx_busy_de10,
  rx_error_de10, rx_data_de10, tx_busy_de10, GPIO_25);
  button_0 : debounce PORT MAP(CLOCK_50, KEY(0), key0_db );
  button_1 : debounce PORT MAP(CLOCK_50, KEY(1), key1_db );
  acel_0 : acel PORT MAP(CLOCK_50, SW(5 downto 4), GSENSOR_INT_I(1 downto 0), GSENSOR_SDI_I, GSENSOR_SDO_I,
  GSENSOR_CS_N_I, GSENSOR_SCLK_I, int_mov(9 downto 0));
  --Asignación de los bits de derecha e izquierda del acelerómetro a una nueva señal
  int_mov_modified <= int_mov(9 downto 6) & int_mov(3 downto 0);
  --Mostrar datos de acelerómetro en los LEDs
  LEDR(9 downto 0) <= int_mov(9 downto 0);

  -- Output => TX_DATA -> data memory address: 0x00
  PROCESS(CLOCK_50, rst_i)
  BEGIN
    IF rst_i = '1' THEN
      tx_data_de10 <= ( OTHERS => '0' );
    ELSIF rising_edge(CLOCK_50) THEN
      IF port_adr_o = "00000000" and -- address port
         port_cyc_o = '1'           and -- control signals for I/O
         port_stb_o = '1'           and
         port_we_o = '1'             -- 'write' operation
      THEN
        tx_data_de10 <= port_dat_o;
    END IF;
  END PROCESS;

```

```

        END IF;
    END IF;
END PROCESS;

-- Output => TX_START -> data memory address: 0x01
PROCESS( CLOCK_50, rst_i )
BEGIN
    IF rst_i = '1' THEN
        tx_ena_de10 <= '1';
    ELSIF rising_edge( CLOCK_50 ) THEN
        IF port_adr_o = "00000001" and -- address port
            port_cyc_o = '1' and -- control signals for I/O
            port_stb_o = '1' and
            port_we_o = '1'      -- 'write' operation
        THEN
            tx_ena_de10 <= port_dat_o(0);
        END IF;
    END IF;
END PROCESS;

-- Output => DISPLAY -> data memory address: 0x02
PROCESS( CLOCK_50, rst_i )
BEGIN
    IF rising_edge( CLOCK_50 ) THEN
        IF port_adr_o = "00000010" and -- address port
            port_cyc_o = '1' and -- control signals for I/O
            port_stb_o = '1' and
            port_we_o = '1'      -- 'write' operation
        THEN
            num_int <= to_integer(unsigned(rx_data_de10));
            -- Convertir unidades, decenas y centenas
            decs <= std_logic_vector(to_unsigned(((num_int mod 100) / 10), 4));
            unids <= std_logic_vector(to_unsigned((num_int mod 10), 4));
            --Cases para unidades y decenas. Se prenden los respectivos segmentos
            --dependiendo del contador
            case unids is
                when "0000" =>
                    HEXO <= "11000000";
                when "0001" =>
                    HEXO <= "11111001";
                when "0010" =>
                    HEXO <= "10100100";
                when "0011" =>
                    HEXO <= "10110000";
                when "0100" =>
                    HEXO <= "10011001";
                when "0101" =>
                    HEXO <= "10010010";
                when "0110" =>
                    HEXO <= "10000010";
                when "0111" =>
                    HEXO <= "11111000";
                when "1000" =>
                    HEXO <= "10000000";
                when "1001" =>
                    HEXO <= "10011000";
                when "1010" =>
                    HEXO <= "10001000";
                when "1011" =>
                    HEXO <= "10000011";
                when "1100" =>
                    HEXO <= "10100111";
                when "1101" =>
                    HEXO <= "10100001";
                when "1110" =>
                    HEXO <= "10000110";
                when others =>
                    HEXO <= "10001110";
            end case;
        case decs is
            when "0000" =>
                HEX1 <= "11000000";

```

```

when "0001" =>
    HEX1 <= "11111001";
when "0010" =>
    HEX1 <= "10100100";
when "0011" =>
    HEX1 <= "10110000";
when "0100" =>
    HEX1 <= "10011001";
when "0101" =>
    HEX1 <= "10010010";
when "0110" =>
    HEX1 <= "10000010";
when "0111" =>
    HEX1 <= "11111000";
when "1000" =>
    HEX1 <= "10000000";
when "1001" =>
    HEX1 <= "10011000";
when "1010" =>
    HEX1 <= "10001000";
when "1011" =>
    HEX1 <= "10000011";
when "1100" =>
    HEX1 <= "10100111";
when "1101" =>
    HEX1 <= "10100001";
when "1110" =>
    HEX1 <= "10000110";
when others =>
    HEX1 <= "10001110";
end case;
END IF;
END IF;
END PROCESS;

--WHEN ELSE para port_dat_i. Dependiendo de si se está haciendo una operación de lectura y si coinciden las direcciones,
--se le asigna cada dato a port_dat_i
port_dat_i <= "0000000" & key0_db WHEN (port_adr_o = "0000001") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & key1_db WHEN (port_adr_o = "00000100") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW(0) WHEN (port_adr_o = "00000101") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW(1) WHEN (port_adr_o = "00000110") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
"0000000" & SW_int WHEN (port_adr_o = "00000111") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
int_mov_modified WHEN (port_adr_o = "00001000") and (port_cyc_o = '1') and (port_stb_o = '1') and (port_we_o = '0') ELSE
UNAFFECTED;
END Structural;

```