Desarrollo del Juego Tetris con Comunicación Serial en FPGA

Jossue Patricio Espinoza Tobar,
Gustavo Garciarreal Morales y
Karina Vergara Aguilar
Diseño con lógica programable
Tecnológico de Monterrey
CDMX, México

Resumen— En este proyecto se describe la recreación del juego Tetris en el cual se utilizan herramientas como FPGA, en conjunto con Processing (basado en JAVA). Se puede observar la adaptación visual al estilo retro del Tetris de Game Boy de 1989 [1] y la implementación de controles físicos para la manipulación de los tetrominos, utilizando un enfoque para explorar la interacción entre el hardware, el software, y la lógica de juego a través de un flujo de datos bien definido que maneja las operaciones del juego.

Palabras Clave — FPGA, processing, comunicación serial, VGA.

I. INTRODUCCIÓN

Las consolas de videojuegos han sido parte fundamental en la infancia de muchos días. Sin embargo, también han otorgado muchos avances en la industria tecnológica. En 1984 el matemático Alekséi Pázhitnov, Dmitry Pavlovsky, y Vadim Guerásimov durante el régimen comunista en la URSS, trabajaron en el centro de computación de Dorodnitsyn de la Academia de Ciencias en Moscú. La intención de los matemáticos era recrear en el ordenador un juego similar al Pentominó. Este se imaginó las piezas del Pentominó cayendo en un vaso, mientras los jugadores las podían rotar o desplazar para completar estructuras, el juego resultó demasiado difícil, por lo que modificó las piezas originales utilizando tetrominos convirtiéndose en uno de los motivos por los cuáles su nombre es Tetris. [2]

Se planteó diseñar el juego Tetris utilizando la herramienta Processing y el lenguaje de programación Java. Este debía ser capaz de utilizar periféricos externos a la computadora y desplegar una parte del videojuego en VGA. Se utilizó una FPGA para conectar tres botones, que nos permitieran controlar, inicio, caída y caída abrupta. Se colocaron dos potenciómetros que al implementar el convertidor analógico digital (ADC) incorporado dentro de la misma, haciendo posible que controlemos la ubicación horizontal y rotacional de la pieza. Además, con el uso de un Arduino se desarrolló la comunicación serial del proyecto, mediante el uso del protocolo UART.

II. MARCO TEÓRICO

Los sistemas ciberfísicos (CPS, Cyber Physical System) representan una nueva generación de sistemas con integración de computación, redes y procesos físicos. La computación incorporada y la conectividad permiten que los CPS interactúen con los seres humanos. Sin embargo, la construcción de sistemas CPS presenta muchos desafíos debido a su complejidad, la necesidad de escalabilidad, la capacidad de adaptarse a los cambios y la necesidad de manejar incertidumbres en el entorno. [3]

La integración de sistemas ciber-físicos conlleva el desarrollo de algoritmos y estructuras que faciliten la interacción entre los elementos físicos y digitales del sistema. Este proceso implica la creación de controladores capaces de gestionar la comunicación entre las distintas partes del sistema, y la posterior implementación de estos controladores tanto en hardware como en software. [4]

La comunicación serial half-duplex es un tipo de comunicación serial que puede ser implementada mediante el protocolo UART, esta se caracteriza porque los datos pueden fluir en ambas direcciones siempre y cuando no se ejecuten dos acciones al mismo tiempo, es decir solo se puede recibir o enviar, nunca las dos al mismo tiempo. Su principal ventaja es la poca necesidad de recursos lo cual es beneficioso cuando existen anchos de banda limitados, aunque esta puede llegar a tener tiempos de espera contraproducentes. [8]

Half duplex básicamente los datos pueden transmitirse en dos direcciones, pero no al mismo tiempo, por lo tanto es una dirección bidireccional, un ejemplo de este podría ser un walkie-talkie. Este tipo de redes requieren un mecanismo para evitar colisiones de datos por lo que verifica si hay algo transmitiendo antes de intentar enviar algo. El modo half duplex está presente en las redes wifi. De igual forma se destaca que cada carácter transmitido de nuestra en un monitor inmediatamente.

El protocolo UART (Universal Asynchronous Receiver-Transmitter) se utiliza para la comunicación en serie entre dispositivos. Este protocolo convierte los datos en serie para su transmisión y los datos recibidos en paralelo para su procesamiento. UART no requiere un reloj para la sincronización, lo que lo hace asincrónico. En lugar de un reloj, utiliza velocidades de baudios predefinidas para la transmisión y recepción de datos. Cada byte de datos se enmarca con un bit de inicio y uno o dos bits de parada. Además, UART puede utilizar un bit de paridad para la detección de errores. Aunque el protocolo UART es antiguo, sigue siendo ampliamente utilizado en aplicaciones de baja velocidad debido a su simplicidad y bajo costo. [9]

III. METODOLOGÍA

A. Desarrollo del Videojuego

El juego en el apartado visual debía contener una pantalla de inicio, un reloj, un marcador de puntuación, un tablero y un indicador de siguiente pieza.

Los tetrominos deben caer desde el borde superior del marco dentro del tablero, estos deben ser capaces de rotar en ambas direcciones sobre su propio eje, moverse de lado a lado y caer hasta la parte inferior del tablero. En el caso de que se complete una fila, esta se elimina y se suma 100 puntos al contador. Además, se debe poder visualizar el tetromino

siguiente. El nivel de dificultad del juego va aumentando conforme el tiempo de partida, aunque hemos decidido omitir la habilidad “hold”, es decir la capacidad de guardar una pieza y cambiarla, junto con la mecánica de aumento de nivel.

Se buscaba conseguir un aspecto retro, adaptado al desarrollo actual de la industria, por lo que inspirados en el Tetris para Game Boy de 1989 [1], se decidió construirlo en tonos grises, blancos y negros. Para ello es necesario establecer nuestras condiciones.

1) Visualización del Videojuego

En primera instancia debíamos encontrar la forma de representar un tablero amigable para el usuario por lo que basados en el principio de UI/UX se decidió darle un apartado visual fidedigno al funcionamiento del juego.

```
función drawForeground(){
    para cada y en el rango de 0 a mapHeight {
        para cada x en el rango de 0 a mapWidth {
            si map[y * mapWidth + x] es igual a 0 {
                // Dibuja el borde del marco.
            }
            sino {
                si map[y * mapWidth + x] es igual a 1 {
                    // Dibujar el exterior
                }
                sino {
                    // Dibuja el interior
                }

                si hay bloques para remove {
                    para cada bloque en blocksToRemove {
                        si el bloque coincide con la posición actual {
                            // Secuencia para remover colores
                        }
                    }
                }
            }

            // Mover la posición al siguiente tile
        }

        // Mover a la siguiente fila
    }
}
```

Fig. 1. Función drawForeground() que contiene la lógica para representar el tablero de tetris.

La segunda dificultad presentada era la creación de los tetrominos o piezas dentro del juego, por lo que utilizando un sistema de arreglos se elaboró la lógica de movimiento, rotación y visualización en pantalla nativa. [5]

Se genera un arreglo con espacios predefinidos que nos ayudarán a representar cada pieza que se encuentran marcadas con una x representativa (fig. 2), esto gracias a la función drawFallingPiece() la cual tiene como objetivo dibujar nuestra pieza en movimiento, itera respecto a X y Y entre el 0 y el 3 revisando nuestro arreglo, siempre que encuentre una X lo reemplazará por nuestra. Además esta considera nuestro estado de rotación, para lo cual es necesario redefinir nuestros arrays mediante la siguiente función rotate() (fig. 3) para cada uno de nuestros estados de rotación, esta tiene un valor de entero y funciona con rotación en X, rotación en Y y el estado de rotación. Finalmente utilizando la función lockCurrPieceToMap() (fig. 4) podemos colocar nuestra pieza en el borde inferior del tablero permitiéndole actuar como parte del mapa. Sin embargo, para ello es una buena práctica revisar primero si cabe en los espacios mediante una función booleana checkIfPieceFits().

```
definir tetrominoes como una lista de cadenas de texto de tamaño 7

tetrominoes[0] =  "--X-"
tetrominoes[0] += "--X-"
tetrominoes[0] += "--X-"
tetrominoes[0] += "--X-"

tetrominoes[1] =  "--X-"
tetrominoes[1] += "-XX-"
tetrominoes[1] += "-X--"
tetrominoes[1] += "----"

tetrominoes[2] =  "-X--"
tetrominoes[2] += "-XX-"
tetrominoes[2] += "--X-"
tetrominoes[2] += "----"

tetrominoes[3] =  "----"
tetrominoes[3] += "-XX-"
tetrominoes[3] += "-XX-"
tetrominoes[3] += "----"

tetrominoes[4] =  "--X-"
tetrominoes[4] += "-XX-"
tetrominoes[4] += "--X-"
tetrominoes[4] += "----"

tetrominoes[5] =  "----"
tetrominoes[5] += "-XX-"
tetrominoes[5] += "--X-"
tetrominoes[5] += "--X-"

tetrominoes[6] =  "----"
tetrominoes[6] += "-XX-"
tetrominoes[6] += "-X--"
tetrominoes[6] += "-X--"
```

Fig. 2. Representación aplicada en seudocódigo de los arreglos necesarios para generar cada pieza.

```
función int  rotate(int rx, int ry, int rState)
// Rotación en x = rx
// Rotación en y = ry
{
    seleccionar (rState)
    {
        caso 0:
            retornar ry * 4 + rx;
        caso 1:
            retornar 12 + ry - (rx * 4);
        caso 2:
            retornar 15 - (ry * 4) - rx;
        caso 3:
            retornar 3 - ry + (rx * 4);
    }

    retornar 0; // Valor predeterminado si no se cumple ningún caso
}
```

Fig. 3. Función que nos permite rotar la pieza según la orientación de la misma.

```
función lockCurrPieceToMap()
{
    // Inicializar contador de bloques que encajan en el mapa
    definir int blocksThatFit como 0;

    para cada fila y en el rango de 0 a 3 {
        para cada columna x en el rango de 0 a 3 {
            definir pieceIndex como rotate(x, y, rotationState);

            si el carácter en la posición pieceIndex de
tetrominoes[currPieceType] es 'X'  {
                // Calcular el índice del mapa correspondiente
                definir mapIndex como (currPieceY + y) * mapWidth +
(currPieceX + x);
                // Verificar que el índice del mapa esté dentro de los
límites y no esté ocupado
                si mapIndex es mayor o igual que 0 y map[mapIndex] es
igual a 0
                {
                    // Actualizar el valor en el mapa con el tipo de
pieza
                    map[mapIndex] = currPieceType + 1; // Se suma 1
porque los tipos de pieza comienzan en 1
                    // Actualizar el color de la ficha en el mapa
                    tileColors[mapIndex] = currPieceColor;
                    // Incrementar el contador de bloques que encajan
                    incrementar blocksThatFit en 1;
                }
            }
        }
    }
    si blocksThatFit es diferente de 4 {
        gameOver = verdadero; // Establecer el juego como finalizado
    }
    si no es gameOver {
        checkForRows(); // Verificar si hay filas completas
        updateGameSpeed(); // Actualizar la velocidad del juego
        getNewPiece(); // Obtener una nueva pieza
        getNextPiece(); // Obtener la siguiente pieza
    }
}
```

Fig. 4. Función que nos permite visualizar la pieza al momento de contactar con los tetrominos que cayeron con anterioridad.

2) *Físicas y Lógica del Videojuego*

Nuestro videojuego debe mantenerse desplegando en pantalla constantemente los cambios que se presentan dentro del tablero por lo tanto necesitamos generar una función update() la cuál además nos permite observar como desciende la pieza tras cierto tiempo incluso si el jugador no presiona el botón de la acción, esta como buena pràctica de programación debe revisar que no nos encontrames en game over ni en la pantalla de inicio. En el caso de que existan bloques a remover esta debe hacerlo mediante la función dissolveBlocks(), revisa que las piezas quepan adecuadamente con la función checkIfPieceFits(). Además en el caso de que haya pasado el tiempo suficiente, a la pieza que se encuentra cayendo debe sumarle 1 en el eje de las Y.

Siempre que una fila se complete esta deberá desaparecer y todas las que se encuentren por encima deberán decender a ocupar el lugar de la anterior fila por lo tanto necesitamos una función checkForRows() la cual cuenta el número de cuadros ocupados en nuestro tablero, si toda la fila llega a su máxima capacidad esta mandará a llamar a dissolveBlocks() en la que primero se calcula la fila más baja a ser eliminada, se almacenan en un arreglo temporal las alturas actuales de las filas por encima a la que debe ser eliminada, se actualizan todas las filas modificando su valor en el eje Y en el caso de ser necesario y se dispone el espacio disponible resultante a la eliminación de la fila.

```
función dissolveBlocks() {
        entero startHeight = (fila más baja del bloque a eliminar + 1) /
mapWidth
        arreglo de enteros rowsToRemoveHeights  // Se declara

        para cada i en rango(0, tamaño de blocksToRemove / 10) hacer {
            agregar ((elemento i * 10 de blocksToRemove + 1) / mapWidth)
a rowsToRemoveHeights
        }

        entero numRowsToDisplace = 0 // Contador de filas a reemplazar

        desde y = startHeight hasta 0 hacer {
            booleano doDisplace = verdadero
            para cada j en rango(0, tamaño de rowsToRemoveHeights) hacer
            {
                si (y es igual a rowsToRemoveHeights[j]) entonces {
                    incrementar numRowsToDisplace en 1
                    doDisplace = falso
                }
            }

            desde x = 1 hasta mapWidth - 1 hacer {
                si doDisplace = verdadero entonces {
                    mover bloques hacia abajo
                    restaurar espacio vacío
                }

                limpiar espacio vacío
            }
        }
        limpiar blocksToRemove
}
```

Fig. 5. Función que nos permite disolver bloques y descender todas las filas una ves se haya eliminado una.

Para saber en qué momento hemos perdido revisamos el valor booleano gameOver con valor false, si al momento de utilizar nuestra función lockCurrPieceToMap() (fig. 4) el valor es distinto a 4 que se calcula mediante la suma de el valor en y del tetromino más su posición actual en el eje Y después multiplicado por el ancho del mapa, más el valor el valor resultante en x del tetromino más su posición actual en el eje X.

3) *Controles y Serial*

Los controles funcionan con banderas booleanas, es decir que siempre que se reciba un valor correspondiente a una de las banderas esta se actualizará a verdadero y por ende ejecutará su respectiva acción, ya sea ir a la izquierda, ir hacia abajo, ir a la derecha, rotar en ambas direcciones, descender más rápido, descender instantáneamente e iniciar el juego, además se debe revisar que el serial se encuentre completamente vació para pasar a otra asignación, por lo tanto si se aplastasen dos botones al mismo tiempo, o estuviera activo un potenciómetro y un botón este tomará solo el valor del que haya entrado primero para evitar que se sature, se recomienda revisar que se sature el estado de rotación como buena práctica y finalmente se actualizan todas las banderas a su valor original una ves estas dejan de ser llamadas.

Para poder enviar datos desde el juego a nuestro sistema ciberfisico, se podrá realizar mediante el uso de un carácter.

B. *Conexiones físicas y lógica digital con el uso de un FPGA*

Por medio del FPGA Altera Max 10 integrado en la tarjeta DE10-Lite de Intel, se realizó la conexión entre los componentes de Hardware y la computadora donde se desee ejecutar Tetris. Debido a que en la tarjeta se encuentran

integrados componentes diversos de bastante utilidad, por ejemplo 20 pines de entrada y salida, conexiones de Arduino, un puerto VGA de 4 bits, 10 leds, 7 módulos de siete segmentos, entre otros. [6, p. 4]

Para la programación se utilizó VHDL como lenguaje descriptor de Hardware y Quartus como IDE y sintetizador de nuestro programa.

Los principales 4 procesos lógicos consisten en: interpretar las señales recibidas por los componentes electrónicos, enviar la señal mediante el protocolo UART (Transmisor Receptor Asíncrono Universal, por sus siglas en inglés), recibir una señal de protocolo UART, interpretar la señal recibida para mostrar una imagen en VGA.

1) Componentes auxiliares

Se añadió un divisor de frecuencia como componente de nuestra entidad principal, de esta manera se manipula el reloj de 50MHz para establecer una comunicación a 9600 baudios.

Así como un decoder de hexadecimal a 7 segmentos, dicho componente se utiliza dos veces, para mostrar en dos módulos de 7 segmentos la señal que se recibe desde el videojuego.

Un convertidor de señal analógica a digital (implementado a partir del catálogo IP), necesario para la implementación de dos potenciómetros. Cuyos valores resultantes van desde “000000000000” hasta “ 111111111111” binario (0-4095 entero).

2) Comunicación serial con protocolo UART

El protocolo UART se puede analizar en 3 partes, el reposo donde simplemente la transmisión se mantiene en ‘1’ lógico, el comienzo de la transmisión en el que se realiza un flanco de bajada y el envío de los datos. [7]

Para que la comunicación sea efectiva se necesita que los dos dispositivos entre los cuales se desea realizar la comunicación, trabajen a la misma frecuencia. Puesto que este protocolo no les envía a los dispositivos una señal de reloj, con la cual puedan sincronizar la llegada o salida de datos.

Para la comunicación mediante UART puede ser implementado un bit de paridad y uno de parada. En este caso no implementamos ninguno de los dos.

a) Componente para la recepción

Se desarrolló la lógica de la recepción por medio de una máquina de estados, donde existen 9 estados (reposo, s0, s1, s2, s3, s4, s5, s6, s7). Se utilizó un proceso para los cambios entre estados y un proceso distinto para asignar los bits de llegada a una señal.

Cuando el estado es “otro” se le asigna reposo, en el reposo se verifica si Rx (La recepción de bits) es 0, para implementar la lógica del protocolo UART. Posteriormente se cambia entre estados desde el 1 hasta el séptimo. Guardando los bits de llegada en una variable, al final se asigna a una señal el valor de la variable.

b) Componente para el envío

De la misma forma se implementa con una máquina de estados. (reposo, comienzo, s1, s2, s3, s4, s5, s6, s7, s8, s9). Como se desea mandar todo el tiempo una señal, sin un botón de envío, se pasa de reposo a comienzo directamente. En el estado comienzo a Tx se le asigna ‘0’ lógico y para asegurar que la transmisión no cambie durante el envío se utiliza una variable llamada *instruccion-aux* a la cual se le asigna el valor de la instrucción actual durante el estado comienzo. El estado cambia a s1 y a tx se le asigna el valor de *instruccion-aux(0)*, de s1 a s2 y a tx se le asigna instruccion-aux(1), así posteriormente. La asignación de la instrucción a la señal tx acaba en s8, donde se pasa a s9.

El motivo de utilizar un estado adicional s9 es pausar la transmisión si la instrucción que se desea enviar al videojuego no ha cambiado. Por lo que si la instrucción auxiliar es igual a la instrucción, se mantiene el estado s9. Donde a tx le es asignado un ‘1’ lógico, al igual que en el estado “reposo”.

3) Interpretación de los controles físicos

Se utilizaron dos botones y dos potenciómetros para la interacción del usuario con el videojuego, cuyas conexiones y señales se muestran en la Fig. 6.

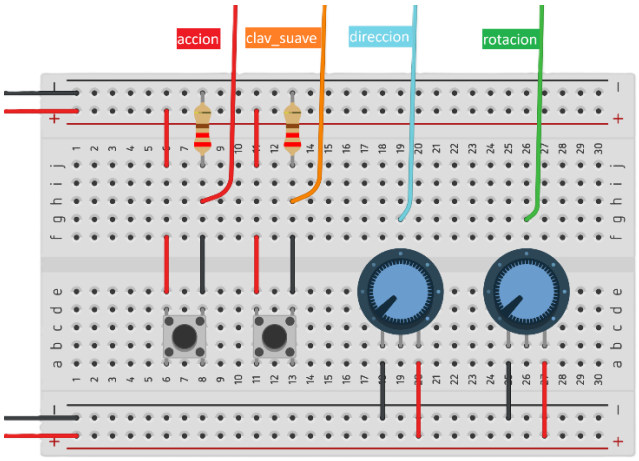

Fig. 6. Conexión del control y nombre utilizado para las señales.

Las señales *accion* y *clav_suave* son conectadas a dos de los pines digitales de entrada y salida, mientras que *direccion* y *rotacion* son conectados a los pines de entrada analógica.

Dentro de una entidad que recibe de entrada estas señales y tiene de salida la instrucción en 8 bits. Se ejecuta un proceso el cual asigna a una variable auxiliar un carácter que nosotros interpretamos en el videojuego como una instrucción. Por ser una estructura secuencial existe prioridad entre las instrucciones

Decidimos que el botón de acción tiene mayor prioridad por lo que si la señal *accion* es ‘1’ a la variable auxiliar le es asignada un espacio ‘ ’, la siguiente instrucción con prioridad es el clavado suave a la cual se le asignaba el carácter ‘s’ en caso de ser presionado el botón. Posteriormente el potenciómetro de rotación tendría prioridad, y por último el de desplazamiento. En el caso de los potenciómetros decidimos interpretar la señal como un entero y preguntar si la señal era menor que 1365 o mayor que 2730, números que equivalen 1/3 y 2/3 de 4095 respectivamente, por lo que de estar el potenciómetro en reposo (a la mitad) simplemente debes superar esos límites para leer estas señales. La instrucción para rotar a la izquierda es ‘I’, a la derecha ‘D’. Para desplazarse a la izquierda ‘a’ y a la derecha ‘d’. En el caso de que ningún botón es presionado y ningún potenciómetro girado, se asigna una ‘u’ a la variable auxiliar.

Después de verificar las condiciones se le asigna el valor del vector lógico del carácter en la variable a una señal llamada *instruccion* la cual es posteriormente usada como entrada en el componente para el envío de datos al videojuego en protocolo UART.

4) Interpretación de la llegada

Se decidió utilizar los datos de llegada para mostrar en un monitor: Una pantalla de inicio, las piezas siguientes a utilizar y una pantalla de fin de juego.

Lo primero que se hace con la señal de llegada, la cual es una salida de la entidad para recepción, es mostrarla en los displays de 7 segmentos, proceso útil para saber si la llegada desde el videojuego fue correcta. Lo siguiente es utilizarla como una entrada de la entidad encargada del VGA.

La entidad VGA recibe como entrada esta llegada, el reloj de 50MHz. Y como salida un valor de 4 bits para cada uno de los colores que se utilizan en VGA los cuales son rojo, verde y azul, además de las señales HS y VS necesarias para mostrar resultados en VGA.

En la arquitectura del componente, primero se hace un pequeño divisor de frecuencia, para utilizar un reloj de 25MHz. Posteriormente se crea por medio de un contador y condiciones descritas en el protocolo VGA una señal que indica si en el momento se encuentra en tiempo de pantalla.

En caso de encontrarse en tiempo de pantalla se pregunta por la señal de llegada, si esta señal es “00” en hexadecimal, se muestra en pantalla la pieza vertical, la cual tiene índice cero en el videojuego. Si la señal es “01” se despliega la pieza con índice uno en el videojuego, y así sucesivamente. Si la señal es “46” hexadecimal se muestra la pantalla de fin de juego y por último si la señal de llegada es “FF” hexadecimal se muestra la pantalla de inicio en el monitor.

Esto se hace por medio de condiciones en las que se abarcan bloques de pixeles, si las condiciones se cumplen, las secciones verificadas en las condiciones se pintan de un color específico. En el caso de los dibujos de piezas, si estas condiciones no se cumplían se le asigna un color blanco al resto de la pantalla. Mientras que en los despliegues de inicio y fin del juego al resto de la pantalla se le asignó un color negro.

5) Proyecto implementado en Quartus

Como se ha descrito anteriormente el proyecto fue desarrollado en Quartus utilizando el lenguaje VHDL. La entidad principal de este proyecto tiene una arquitectura estructural que se describe mejor en la Fig. 7 donde se puede observar una representación visual de la estructura y jerarquía del proyecto.

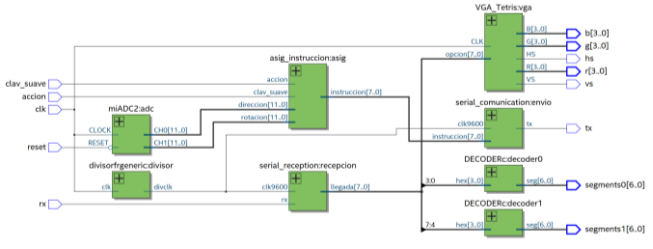

Fig. 7. Vista esquemática de la estructura interna de la entidad principal

Se puede analizar la Fig. 7 en dos secciones, para evitar la complejidad. Los componentes utilizados para el análisis de rx y su respuesta como salida en el monitor. Y los componentes utilizados para el análisis de la instrucción recibida de los componentes electrónicos y su transmisión en la señal tx.

IV. RESULTADOS

Cuando inicia el juego se despliega una pantalla de inicio, las piezas son capaces de rotar en ambas direcciones con el uso del potenciómetro derecho además de desplazarse alrededor del mapa con el potenciómetro izquierdo, el botón 1de inicio también sirve como botón de caída instantánea, en cambio el botón 2 sirve para una caída acelerada, el siguiente tetromino se muestra en la pantalla siguiente del VGA, además de mostrar una pantalla de fin cuando el juego termina.

Fig. 8. Proyección del VGA representado cuando inicia el videojuego

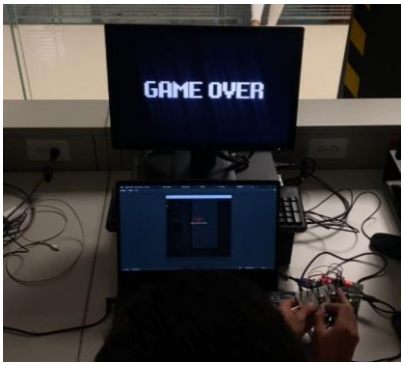

Fig. 9. Proyección del VGA cuando se ha perdido en el videojuego

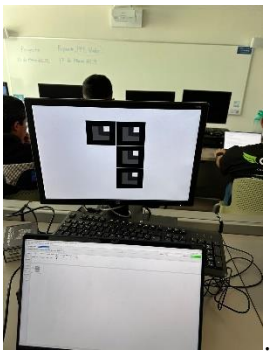

Fig. 10. Proyección del VGA representado al tetromino 5.

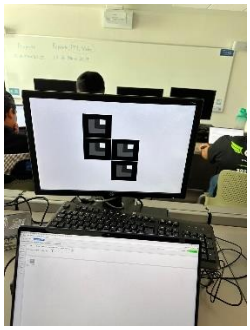

Fig. 11. Proyección del VGA representado al tetromino 2.

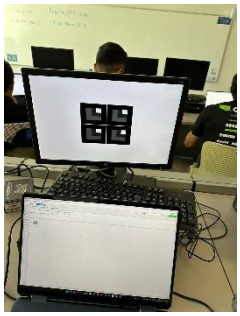

Fig. 12. Proyección del VGA representado al tetromino 3.

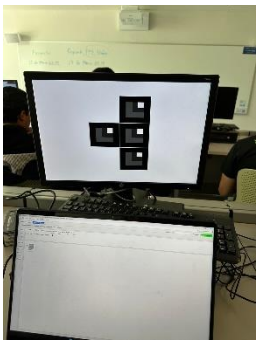

Fig. 13. Proyección del VGA representado al tetromino 4.

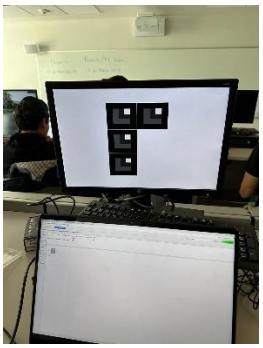

Fig. 14. Proyección del VGA representado al tetromino 6.

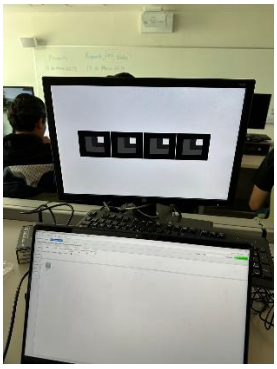

Fig. 15. Proyección del VGA representado al tetromino 1.

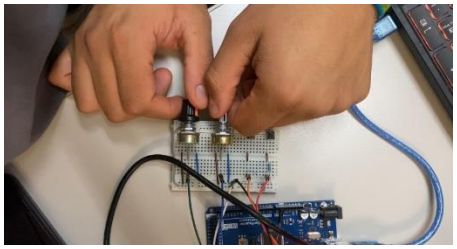

Fig. 16. Circuito del control físico

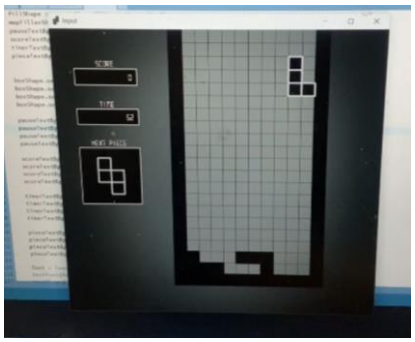

Fig. 17. Ejecución del videojuego desarrollado

V. CONCLUSIONES

Karina Vergara Aguilar:

Jossue Patricio Espinoza Tobar: Como estudiante de Ingeniería Robótica me ha parecido una experiencia única y enriquecedora que me ha abierto los ojos a un nuevo campo de posibilidades, en lo personal era un apasionado del Hardware en mi día a día por lo que tener la posibilidad de trabajarlo por mi cuenta y sobre todo ser capaz de crearlo me ha fascinado. Sin embargo, nuestro trabajo colaborativo posee áreas de oportunidad. Finalmente me voy con un buen sabor de boca de nuestro proyecto, le dedicamos muchas horas y atención a los detalles, proyectándonos en nuestra propia experiencia. Los FPGA han cambiado la manera de actualizar al mundo y yo solo tengo claro que espero ser parte de ello.

Gustavo Garciarreal Morales: Como estudiante de Ingeniería en Robótica, adentrarme en el mundo de los Arreglos de Puertas Lógicas Programables en Campo (FPGAs) ha sido una experiencia única. Estos dispositivos tan versátiles permiten el desarrollo de sistemas ciberfísicos avanzados, para tareas específicas y mejoran el rendimiento de estos. Mi fascinación por los FPGAs ha crecido mucho, me interesa complementar estos conocimientos con mi interés en las industrias automotriz y aeroespacial, donde la integración de sistemas robustos y confiables es de suma importancia. El potencial de los FPGAs para revolucionar estos sectores es

inmenso, ofreciendo un control y adaptabilidad sin precedentes. Desde mejorar la automatización de vehículos hasta avanzar en el diseño aeroespacial, los FPGAs brindan una oportunidad única para innovar y empujar los límites de lo posible en maravillas de ingeniería que dan forma a nuestra vida cotidiana y nos impulsan hacia el futuro.

REFERENCIAS

[1] Tetris. “History of Tetris®”. Tetris. Accedido el 15 de marzo de 2024. [En línea]. Accedido el 11 de marzo de 2024 Disponible: https://tetris.com/history-of-tetris

[2] D. Martínez. “La curiosa historia del Tetris, el juego que nació en el comunismo”. Hobby Consolas. Accedido el 11 de marzo de 2024. [En línea].Disponible: https://www.hobbyconsolas.com/reportajes/curiosa-historia-tetris-juego-que-nacio-comunismo-278241

[3] E. A. Lee, “Cyber Physical Systems: Design Challenges,” 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), 2008, pp. 363-369.

[4] R. Alur, “Principles of Cyber-Physical Systems,” MIT Press, 2015.

[5] javidx9, «Code-It-Yourself! Tetris - Programming from Scratch (Quick and Simple C++)», *YouTube*. 3 de abril de 2017. [En línea]. Accedido el 16 de marzo de 2024. Disponible en: https://www.youtube.com/watch?v=8OK8_tHeCIA

[6] Terasic Inc, *DE10-Lite User Manual*, 4ª ed. Hsinchu: Terasic Inc, 2016.

[7] “Protocolo de comunicación UART - ¿Cómo funciona? - Codrey Electronics | IWOFR”. IWOFR | Conference. Accedido el 18 de marzo de 2024. [En línea]. Disponible: https://iwofr.org/es/protocolo-de-comunicación-uart-cómo-funciona/

[8] Jimenez, J. (2023, 27 de diciembre). *Half Dúplex vs Full Dúplex: diferencias principales y funcionamiento*. RedesZone. https://www.redeszone.net/tutoriales/redes-cable/diferencias-full-half-duplex/

[9] “UART (Universal Asynchronous Receiver/Transmitter)”, Computer Networks, vol. 4, no. 2, pp. 123-128, 2020.