Instituto Tecnológico y de Estudios Superiores de Monterrey

TE2002B Diseño con lógica programable(Gpo 101)

Diseño y evaluación del CPU

Autores:

Mariam Landa Bautista // A01736672

Josué Aldemar Garduño Gómez // A01733320

Yestli Darinka Santos Sánchez // A01736992

Profesora:

David Antonio Torres

Jueves 14 de marzo de 2024 Semestre (4) Feb-Ago 2024

Campus Puebla

INTRODUCCIÓN

En este proyecto, optamos por utilizar Processing, un entorno de programación flexible diseñado para artistas y diseñadores, como nuestra plataforma de desarrollo principal para la interfaz de usuario y la lógica del juego, processing nos permite crear prototipos rápidos y experimentar con ideas visuales complejas de manera más accesible.

Nuestro enfoque radica en la incorporación de VHDL (VHSIC Hardware Description Language) y lenguaje ensamblador, dos herramientas para la programación de bajo nivel que nos permiten aprovechar al máximo las capacidades de la tarjeta Arty. VHDL se utiliza para describir el comportamiento y la estructura del hardware electrónico de manera precisa, permitiéndonos personalizar el funcionamiento de la tarjeta para necesidades específicas del juego. El lenguaje ensamblador ofrece un control granular sobre el hardware, optimizando el rendimiento y la eficiencia del juego.

Este proyecto no sería posible sin el apoyo de Intel, nuestro socio formador, cuya experiencia en el desarrollo de tecnología de punta ha sido invaluable, su colaboración no solo nos proporciona acceso a recursos y conocimientos técnicos avanzados, sino que también subraya la importancia de la sinergia entre la innovación en el software y el hardware en la creación de videojuegos modernos.

PROCEDIMIENTO

Primero definimos los puertos utilizados que cumplen con cada función, como leer el estado, para transmitir datos, verificar, además se asignaron nombres a los registros para mejorar la legilibidad.

```
; Definición de constantes para los puertos utilizados
CONSTANT PuertoBoton, 00            ; Puerto para leer el estado de los botones
CONSTANT PuertoLeeListoTX, 11       ; Puerto para verificar si está listo para transmitir
CONSTANT PuertoEscribeDatoTX, 12    ; Puerto para escribir datos a transmitir

; Asignación de nombres a registros específicos para mejorar la legibilidad
NAMEREG s7, Space           ; Registro para almacenar el estado de los botones
NAMEREG s5, DatoSerial      ; Registro para almacenar el dato a ser transmitido
NAMEREG s6, EstadoTX        ; Registro para almacenar el estado de listo para transmitir
```

```
; Dirección inicial de ejecución del programa
ADDRESS 000
loop:
      ; Leer el estado de los botones y almacenarlo en Space
      INPUT Space, PuertoBoton
      ; Filtrar solo los 4 bits menos significativos del estado de los botones
      AND Space, 0F

      ; Comparar el estado filtrado de los botones con 01 (primer botón presionado)
      COMPARE Space, 01
      ; Si el resultado de la comparación es cero (igual a 01), salta a salidaD (debería ser salidaSpace o salidaDefault)
      JUMP Z, salidaD

; Sección para manejar la salida cuando el primer botón está presionado
salidaSpace:
      ; Cargar el valor 01 en DatoSerial para su transmisión
      LOAD DatoSerial, 01
      ; Llamar a la subrutina de transmisión
      CALL tx_uart
      ; Volver al inicio del bucle principal
      Jump loop

; Sección para manejar la salida por defecto (ningún botón presionado o condiciones no cumplidas)
salidaDefault:
      ; Cargar el carácter "." en DatoSerial para su transmisión
      LOAD DatoSerial, "."
      ; Llamar a la subrutina de transmisión
      CALL tx_uart
      ; Volver al inicio del bucle principal
      Jump loop
```

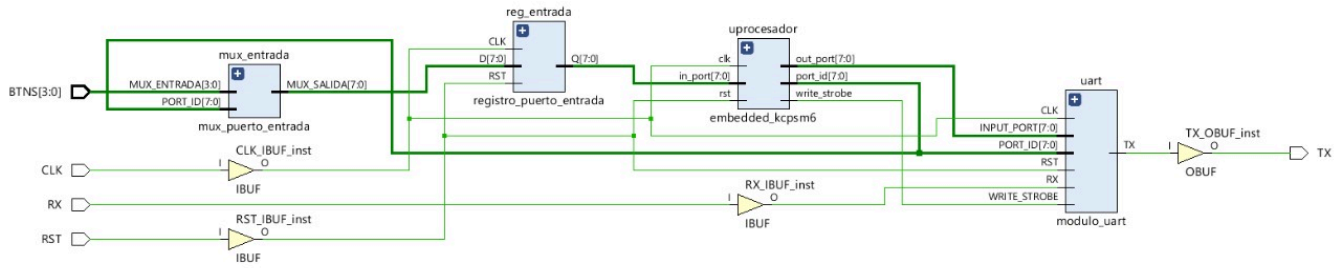
```
; Subrutina para manejar la transmisión de datos
tx_uart:
      ; Leer el estado del puerto de listo para transmitir y almacenarlo en EstadoTX
      INPUT EstadoTX, PuertoLeeListoTX
      ; Comparar EstadoTX con 01 (listo para transmitir)
      COMPARE EstadoTX, 01
      ; Si no está listo (resultado no es cero), volver a intentar
      JUMP Z, tx_uart
      ; Una vez listo, escribir el dato en DatoSerial al puerto de transmisión
      OUTPUT DatoSerial, PuertoEscribeDatoTX
      ; Llamar a la subrutina de retraso para controlar el tiempo entre transmisiones
      CALL delay
      ; Retornar de la subrutina
      RETURN

; Subrutina para crear un retraso por software
delay:
      ; Cargar valores iniciales para el conteo del retraso
      LOAD s2, 17
      LOAD s1, D7
      LOAD s0, 84

      ; Bucle de retraso, decrementa los contadores hasta que todos lleguen a 0
delay_loop:
      SUB s0, 1'd
      SUBCY s1, 0'd
      SUBCY s2, 0'd
      ; Si alguno de los registros aún no es 0, continuar el bucle
      JUMP NZ, delay_loop
      ; Retornar de la subrutina una vez completado el retraso
      RETURN
```

Para la parte del picoblaze definimos una entity, en donde registramos los puertos y componentes necesarios. En este caso es el top level de la arquitectura, nos permite leer y escribir de la tarjeta arty a la serial, en este caso estamos agregando también la entrada de nuestros componentes (Botones) con los que estaremos enviando señales a nuestra terminal serial y los cuales harán la función de periféricos.

```
12
13  entity picoblaze_uart is
14      Port ( CLK : in STD_LOGIC;
15             BOTONES : IN STD_LOGIC_VECTOR (2 downto 0);
16             RST : in STD_LOGIC;
17              TX : out STD_LOGIC;
18              RX : in STD_LOGIC);
19  end picoblaze_uart;
20
21  architecture Behavioral of picoblaze_uart is
22  --declaraciÃ³n de componentes
23  component modulo_uart is
24      Port (
25                       CLK : in STD_LOGIC;
26                       RST : in STD_LOGIC;
27                       --pines de comunicaciÃ³n con PicoBlaze
28                   PORT_ID : in STD_LOGIC_VECTOR (7 downto 0);
29                INPUT_PORT : in STD_LOGIC_VECTOR (7 downto 0);
30               OUTPUT_PORT : out STD_LOGIC_VECTOR (7 downto 0);
31              WRITE_STROBE : in STD_LOGIC;
32                        --pines de comunicaciÃ³n serial
33                       TX : out STD_LOGIC;
34                       RX : in STD_LOGIC
35                       );
36  end component;
```

Diagrama RTL

Para el diagrama tenemos el multiplexor que selecciona la entrada de los botones, que nos lleva al registro de entrada, de ahí selecciona el botón que hemos seleccionado, posteriormente pasamos a nuestro componente uart, en donde este funciona para la descripción de lectura y escritura.



El esquema RTL muestra botones o interruptores (BTNF[3:0]) que alimentan un multiplexor de entrada (MUX_ENTRADA), cuyas señales seleccionadas se pasan a través de un registro de puerto de entrada antes de ser procesadas por un microprocesador (uproc) y una unidad embebida (embedded_kcpsm6) y hay un componente UART para la comunicación serial. Señales como el reloj (CLK), recepción (RX) y reinicio (RST) se utilizan para la sincronización y la inicialización, los buffers de entrada y salida (IBUF/OBUF) ajustan los niveles de señal para la interfaz con otros dispositivos y por último las líneas verdes representan las conexiones entre los componentes y cómo fluyen las señales entre ellos.

IMPLEMENTACIÓN DEL VIDEOJUEGO

Para la implementación del video juego lo primero que hicimos fue importar las librerías necesarias, en este caso importamos la “serial” para hacer la conexión con el FPGA y la “mínim” que es para agregarle sonido al juego, como siguiente paso definimos todas las variables a utilizar, las de imágenes, objetos que en este caso fue el pez, obstáculos que se usó la imágen de una mina, para la música y para el serial.

sketch 240307a | prueba2

```
1  import processing.serial.*;
2  import ddf.minim.*;
3  Serial port;
4  PImage img, pezImage, topTubeImage, bottomTubeImage; // Variables globales para las imágenes
5  Pez pez;
6  ArrayList<Tube> tubes;
7  int score, level;
8  float tubeSpeed, bgX, bgSpeed;
9  boolean gameOver, resetRequested;
10 Minim minim;
11 AudioPlayer player;
12
13 void setup() {
14   size(800, 800);
15   img = loadImage("imagenocean.jpeg");
16   pezImage = loadImage("P.png");
17   topTubeImage = loadImage("mina2.png");
18   bottomTubeImage = loadImage("mina.png");
19   port = new Serial(this, "COM8", 115200);
20   minim = new Minim(this);
21   player = minim.loadFile("fs.mp3");
22   player.loop();
23   initGame();
24 }
```

En esta parte del código realizamos una función para que el fondo se pusiera correctamente y se pudiera repetir todas las veces del juego sin verse un salto dentro de la imágen, además pusimos la variable bgX para que el fondo se desplace adecuadamente y sea continuo.

```
// Función para mover y repetir el fondo
void moveBackground() {
  // Dibujo del fondo desplazándolo con bgX
  image(img, bgX, 0, width, height);
  image(img, bgX + width, 0, width, height);
  bgX -= bgSpeed; // Actualización de la posición del fondo
  if (bgX <= -width) {
    bgX = 0; // Reposicionamiento del fondo para efecto continuo
  }
}
```

En está sección del juego muestra la parte completa de la función del juego, se muestran los dibujos del pez, se añade los intervalos de las minas, su tamaño va variando constantemente, los valores que se arrojan para su medida son aleatorios, además se hace las colisiones y logra eliminar los tubos que se encuentran fuera de la pantalla, declaramos que cada 10 minas pasadas el nivel sube, por ende la velocidad será mayor.

```
// Lógica principal del juego
void gameLogic() {
  pez.update(); // Actualización del estado del pez
  pez.display(); // Dibujo del pez en pantalla
  // Añadir nuevos tubos a intervalos regulares, aumentando con el nivel
  if (frameCount % (100 - (level * 5)) == 0) {
    tubes.add(new Tube());
  }
  // Actualización y dibujo de cada tubo, detección de colisiones y eliminación de tubos fuera de pantalla
  for (int i = tubes.size() - 1; i >= 0; i--) {
    Tube tube = tubes.get(i);
    tube.update();
    tube.display();
    if (pez.hits(tube)) { // Si hay colisión, el juego termina
      gameOver = true;
    }
    if (tube.offscreen()) { // Si el tubo sale de pantalla, se elimina y aumenta el puntaje
      tubes.remove(i);
      score++;
      // Aumento de nivel cada 10 puntos
      if (score % 10 == 0) {
        level++;
        tubeSpeed += 2;
      }
    }
  }
  displayScore(); // Mostrar puntaje
}
```

La función display GameOver() está diseñada para mostrar una pantalla de "juego terminado" cuando el jugador pierde o el juego concluye.

```
// Función para mostrar la pantalla de juego terminado
void displayGameOver() {
  fill(255);
  textSize(32);
  textAlign(CENTER, CENTER);
  // Mostrar mensajes de juego terminado y puntaje
  text("Game Over", width / 2, height / 2);
  text("Score: " + score, width / 2, height / 2 + 30);
  text("Press Space to Restart", width / 2, height / 2 + 60);
  if (resetRequested) { // Si se solicita reinicio, inicializar el juego
    initGame();
  }
}
```

La función keyPressed() está diseñada para interactuar con el juego controlado por teclado, y también incorpora la capacidad de leer datos desde un puerto serial.

```
// Detección de pulsaciones de teclas
void keyPressed() {
  if (port.available() > 0) { // Leer datos del puerto serial si están disponibles
    int inByte = port.read();
    inChar = (char)inByte;
    println("Recibido:", inChar); // Imprimir el carácter recibido
  }
  if (key == ' ') { // Si se presiona espacio, el pez salta o se reinicia el juego
    if (!gameOver) {
      pez.up();
    } else {
      resetRequested = true;
    }
  }
}
```

La clase Pez está diseñada para modelar y controlar el comportamiento del personaje dentro del juego, en este caso un pez.

```
// Clase Pez para controlar al jugador
class Pez {
  float y, gravity, lift, velocity; // Posición, gravedad, impulso y velocidad

  Pez() {
    y = height / 2; // Posición inicial
    gravity = -0.9; // Afecta cómo de rápido cae el pez
    lift = 20; // Afecta el salto del pez
    velocity = 20; // Velocidad inicial
  }
```

```
void up() { // Método para hacer que el pez "salte"
  velocity += lift;
}
```

```
void update() { // Actualizar la posición del pez basado en la velocidad y gravedad
  velocity += gravity;
  y += velocity;
  // Restricciones para mantener al pez dentro de los límites de la pantalla
  if (y > height) {
    y = height;
    velocity = 0;
  } else if (y < 0) {
    y = 0;
    velocity = 0;
  }
}
```

Y por último la clase tube está diseñada para controlar los obstáculos donde el jugador debe navegar a través de una serie de obstáculos que se presentan en forma de tubos o columnas pero en nuestro caso los tubos los cambiamos por minas.

```
// Clase Tube para controlar los obstáculos
class Tube {
  float top, bottom, x, w; // Dimensiones y posición

  Tube() {
    // Dimensiones aleatorias para variar los obstáculos
    top = random(height / 2.05);
    bottom = random(height / 2.05);
    x = width; // Posición inicial fuera de pantalla
    w = 150; // Ancho
  }

  void update() { // Mover el tubo hacia la izquierda
    x -= tubeSpeed;
  }

  void display() { // Dibujar los tubos en sus posiciones actuales
    image(topTubeImage, x, 0, w, top);
    image(bottomTubeImage, x, height - bottom, w, bottom);
  }

  boolean offscreen() { // Comprobar si el tubo ha salido de pantalla
    return x < -w;
  }
}
```

VIDEO DEMOSTRATIVO:
https://youtube.com/shorts/uDp57x71Eh8?si=EpmdPJIz6H684V4d

https://youtu.be/z9SCNI2CL84?feature=shared

Conclusiones

Mariam Landa Bautista

El proyecto demuestra una integración efectiva entre diferentes disciplinas y tecnologías, al combinar el desarrollo de software con el uso de VHDL y lenguaje ensamblador para programar la tarjeta Arty, el equipo aprovecha lo mejor de ambos mundos: la flexibilidad y rapidez de prototipado de Processing, junto con el control granular y la optimización de rendimiento que ofrecen las herramientas de programación de bajo nivel.

Josué Aldemar Garduño Gómez

La elección de la tarjeta Arty y la colaboración con Intel muestran una estrategia consciente de aprovechar recursos avanzados y la experiencia técnica disponible, esto permite al equipo acceder a tecnologías de vanguardia y conocimientos especializados que contribuyen significativamente al éxito del proyecto.

Yestli Darinka Santos Sánchez

A pesar del enfoque en el desarrollo de hardware y bajo nivel, el proyecto mantiene una atención significativa en la experiencia del usuario y la jugabilidad, a implementación de elementos visuales y sonoros, junto con la interacción controlada por teclado y la capacidad de leer datos desde un puerto serial, muestra un compromiso con la creación de un videojuego inmersivo y atractivo para el usuario final.