DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2025)

Etapa 3 del Proyecto

Descripción

Su trabajo en esta etapa estará centrado exclusivamente en desarrollar programas para probar su computador, donde pueden utilizar el ensamblador.

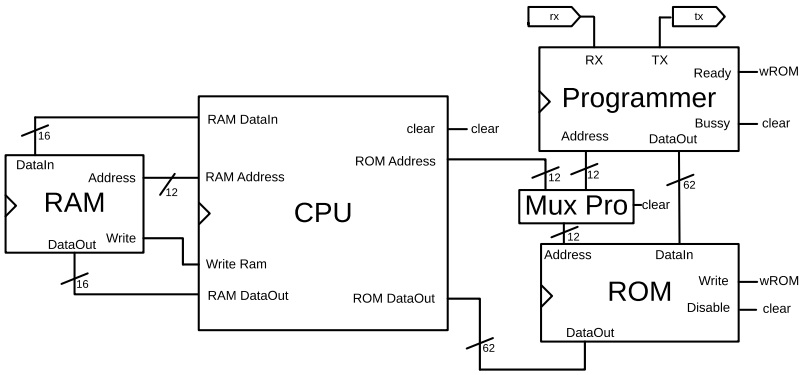

Figura 1: Diagrama parcial del computador básico de proyecto dentro del componente Basys3.

Con su computador básico completo y con las funciones de entrada y salida I/O mapeadas a memoria, solo falta probar su arquitectura. Por lo tanto, para esta entrega tendrán que generar programas en Assembly que utilicen su arquitectura. Además, deberán agregar y utilizar archivos adicionales para poder generar un programa que haga uso de la pantalla con VGA.

Introducción

Para probar las capacidades de su computador, se les pedirá generar dos programas con interacción del usuario a través de los dispositivos de entrada y salida de la placa. Estos programas deberán ser dos juegos, uno pensado para jugar solo a través de la placa y otro a través de un monitor externo. Para el segundo juego se les hará entrega de tres enunciados que contienen información sobre las opciones de juego que pueden realizar. Si deciden solo realizar el primer juego, **no es necesario agregar la funcionalidad de la VGA al proyecto**.

Juego 1: Juego de reacción

El primer juego deberá ser uno de reacción. El objetivo es ver quién tiene mejores reflejos para apretar rápidamente un botón de la placa. Cada jugador tendrá un botón asignado, por lo que podrán jugar como máximo 5 jugadores[1]. El flujo es el siguiente:

- Debe partir esperando que se le dé una señal de inicio por medio de uno de los *switches*, el que llamaremos el ***switch* de inicio**.
- Durante el periodo de espera, si un jugador presiona un botón, se debe mostrar su número en el *Display*.
- Una vez iniciado, debe hacer una cuenta regresiva en el *Display* partiendo en 3 y decrementando hasta llegar a 0.
- Se deberá detectar cuál fue el primer botón presionado después de que terminó la cuenta regresiva.
- Se debe mostrar el número del jugador ganador en el *Display* y la cantidad de milisegundos que le tomó presionar el botón en los LEDs.
- Se debe poder jugar nuevamente al regresar el ***switch* de inicio** a la posición original. Es decir, al volver el *switch* a la posición inicial, el juego debe volver a esperar a que el *switch* de inicio sea activado.

Escriba los programas que estimen convenientes para probar su implementación.

Juego 2: Juego con VGA

Por otra parte, el juego que usted implementará debe respetar las reglas especificadas en el enunciado correspondiente, donde encontrará además su distribución de puntaje. Podrá elegir entre los siguientes juegos:

1. *Space Invaders*
2. *Snake*
3. *Dinosaurio de Google*

Para facilitar las conexiones y funcionalidades necesarias para utilizar la pantalla, se les hará entrega de varios archivos que deben agregar. Estos se explican a detalle a continuación.

[1] No es recomendado utilizar el máximo, por favor piensen en el estrés de la pobre placa.

Hardware

A continuación, se describen los archivos que se proveen como base para trabajar los componentes videográficos. Este conjunto de archivos permite ejecutar un programa de muestra que consiste en una imagen estática que inicia centrada sobre el área visual, la que es posible mover utilizando los botones de la Basys3.

Basys3.xdc

Este archivo define las conexiones que la placa Basys3 habilita dentro de su arquitectura para el reconocimiento de las entradas y salidas. A lo largo del semestre, se ha trabajado con este archivo y se ha encargado de activar las conexiones asociadas a los botones, las luces **led**, los ***switches***, el puerto micro-usb, el *display* de 7 segmentos, entre otros.

El archivo anterior debe ser reemplazado por el entregado en esta etapa para que los *pins* de transmisión de video asociados al puerto **VGA** se activen y funcionen correctamente. Además, es importante actualizar la definición de la entidad `Basys3` dentro del archivo principal de la arquitectura, en el que se deben agregar las salidas asociadas a estas conexiones de la siguiente forma:

```
entity Basys3 is Port (
    -- Conexiones anteriores
    sw          : in   std_logic_vector (15 downto 0);
    btn         : in   std_logic_vector (4 downto 0);
    led         : out  std_logic_vector (15 downto 0);
    clk         : in   std_logic;
    seg         : out  std_logic_vector (7 downto 0);
    an          : out  std_logic_vector (3 downto 0);
    tx          : out  std_logic;
    rx          : in   std_logic;
    -- Conexiones nuevas
    Hsync       : out std_logic;
    Vsync       : out std_logic;
    vgaRed      : out std_logic_vector(3 downto 0);
    vgaGreen    : out std_logic_vector(3 downto 0);
    vgaBlue     : out std_logic_vector(3 downto 0)
    );
end Basys3;
```

Xelor.vhd

El trabajo del protocolo de *Video Graphics Array* requiere de señales de sincronización vertical y horizontal. Este archivo define un componente único que recibe el reloj reducido de la placa ($25Mhz$) y extrae las señales de sincronización necesarias para que la transmisión funcione adecuadamente. Además, provee otras señales que están asociadas a la información de recorrido del puntero visual (*i.e.* el rayo catódico que dibuja la imagen). El componente completo que debe ser incluido en el archivo que se encarga de manejar el video es el siguiente:

```
component Xelor is Port(
    clock_25mhz: in std_logic;
    h_sync: out std_logic;
    v_sync: out std_logic;
    valid_area: out std_logic;
    x_coordinate: out integer;
    y_coordinate: out integer
    );
```

```
end component;
```

Donde:

- `clock_25mhz`: es el reloj reducido de la placa.
- `h_sync`: es la señal de sincronización horizontal.
- `v_sync`: es la señal de sincronización vertical.
- `valid_area`: es un *bit* de activación de transmisión de video. Este es necesario porque el vector de color **debe ser cero fuera del área de visualización**.
- `x_coordinate`: un *integer* que indica la posición del puntero visual en el eje horizontal.
- `y_coordinate`: un *integer* que indica la posición del puntero visual en el eje vertical.

GraphicsProcessor.vhd

Este es el componente de procesamiento videográfico. Su estructura es la siguiente:

```
component GraphicsProcessor is Port(
    clock_25Mhz: in std_logic;
    ram_address: in std_logic_vector(11 downto 0);
    ram_datain: in std_logic_vector(15 downto 0);
    ram_write: in std_logic;
    clear: in std_logic;
    horizontal_syncro: out std_logic;
    vertical_syncro: out std_logic;
    red_vector: out std_logic_vector(3 downto 0);
    green_vector: out std_logic_vector(3 downto 0);
    blue_vector: out std_logic_vector(3 downto 0)
    );
end component;
```

Este componente recibe la información que trabaja la **CPU** para determinar cuándo es necesario actualizar algún valor asociado a los elementos presentes en la pantalla. Debido a que la arquitectura del computador funciona con mapeo de memoria, necesita recibir la dirección de la **RAM** a la que se accede, así como el vector de datos hacia la **RAM** y si se trata o no de una escritura (`ram_write`).

Exporta directamente las señales que deben ser enviadas a través del puerto **VGA**.

SM.vhd

El *Streaming Multiprocessor* se encarga de la lógica que determina el vector de color específico para un *sprite* de acuerdo con las coordenadas actuales del puntero visual. Es el punto de conexión entre el componente de datos de cada imagen y el procesador gráfico. Su estructura es la siguiente:

```
component SM is Port(
    clock: in std_logic;
    current_x: in integer;
    current_y: in integer;
    pixel_color: out std_logic_vector(11 downto 0);
    inside_area: out std_logic;
    ram_write: in std_logic;
    ram_address: in std_logic_vector(11 downto 0);
    ram_datain: in std_logic_vector(15 downto 0)
```

```
    );
end component;
```

Este componente maneja internamente la posición del objeto en el área de la pantalla, por ello, necesita recibir las actualizaciones de posición que sean comunicadas por la **CPU**. De acuerdo con la posición y las dimensiones del objeto que se trabaja, determina si es que el puntero se encuentra dentro del área del *sprite* y lo comunica con la señal `inside_area`. De estar dentro del área, el vector de color asociado es calculado según los *offsets* necesarios y enviado a través de la señal `pixel_color`.

Su_logo.vhd

Este es el componente de datos específico de la imagen de muestra. Su función es leer los bloques alocados en la memoria **BROM** de la placa y retornar los componentes del vector de color según la dirección de acceso.

```
component Su_logo is Port(
    clock: in std_logic;
    addr: in std_logic_vector(13 downto 0);
    red_color: out std_logic_vector(3 downto 0);
    green_color: out std_logic_vector(3 downto 0);
    blue_color: out std_logic_vector(3 downto 0)
    );
end component;
```

Es importante notar que `addr` corresponde a una dirección especial calculada por el *Streaming Multiprocessor* según los *offsets* de posición.

Archivos de colores

Corresponden a `su-logo.png-BLUE.coe`, `su-logo.png-GREEN.coe` y `su-logo.png-RED.coe`. Estos archivos son utilidades para que Vivado pueda cargar los *bytes* de información de la imagen a la memoria de la placa. Estos pueden ser generados utilizando el *MinkiAssembler* tras entregarle una imagen. Cada uno de estos archivos generará un componente interno que debe ser referido de la siguiente forma:

```
component NOMBRE_ASIGNADO is Port(
    clka : in std_logic;
    addra : in std_logic_vector(XX DOWNTO 0);
    douta : out std_logic_vector(3 DOWNTO 0)
  );
end component;
```

Donde:

- `NOMBRE_ASIGNADO`: es el nombre con que se designa al componente en el que se carga cada archivo de colores.
- `XX`: dependerá de las dimensiones de cada *sprite*. Es necesario calcular, dado el número de pixeles presentes, el mínimo ancho del vector de direcciones que actúa sobre un espacio de memoria que contenga los valores de los *bytes*.

movimiento_logo.suasm

Un archivo de código *assembly* que contiene el comportamiento del programa de ejemplo. Se encarga de posicionar el logo en el centro de la pantalla y reconocer cuando se apretan los botones respectivos para actualizar las coordenadas de la imagen.

Movimiento continuo y discreto

En las reglas de cada juego se hace referencia al movimiento discreto y continuo.

Movimiento discreto es cuando un objeto se mueve "por bloques" o "por estados", donde se puede estar o no estar en uno. En el movimiento discreto no existen "estados intermedios". Un ejemplo de esto sería el ajedrez, una figura está o no está en cierta casilla, no puede estar parcialmente en una casilla.

Por otro lado, el movimiento continuo sí permite estos "estados intermedios". Los objetos no se mueven por bloques o estados, sino que pueden ocupar cualquier posición dentro de un rango continuo. Esto implica que su desplazamiento se representa mediante variaciones suaves y graduales en el espacio o el tiempo, en lugar de saltos entre posiciones fijas.

Presentación

Se aceptarán envíos de *commits* hasta **el martes 18 de noviembre a las 13:30**, momento en el que se recolectarán todos los repositorios. **Se tendrá que presentar en los computadores del laboratorio en no más de 10 minutos**. Esto se llevará a cabo con la descarga del comprimido de su repositorio que el equipo docente le proveerá. El día de presentación, deberá mostrar y cargar con el ensamblador los programas desarrollados. Nuevamente, tendrá que explicar cuáles fueron las decisiones de diseño que realizaron, y subir al repositorio de su grupo en *GitHub* lo siguiente:

- Su proyecto en Vivado.
- Los programas en *Assembly* de sus juegos.

Al finalizar esta presentación, deberá entregar:

- La Basys3 y cualquier otro *hardware* que se les haya prestado para el desarrollo de las actividades.

Si su grupo no se presenta, independiente del motivo, **debe entregar todo *hardware* prestado por el equipo docente**. En caso contrario, su grupo será reprobado con nota 1.1 y notificado a la Dirección de Pregrado de la Escuela de Ingeniería.

Puntaje

La nota de los juegos será calculada en base a **tópicos**. Cada **tópico** es evaluado en base a tres estados: **Logrado** (todo el puntaje), **parcialmente logrado** (mitad del puntaje) y **no logrado** (sin puntaje). Para esta entrega, se evaluarán los siguientes **tópicos**, además, se correrá un test que prueba la totalidad de su entrega 2:

- **Test**: Correctitud de entrega 2 completa. [**10 puntos**]
- **Flujo de Juego de reacción:** Se cumplen los 6 puntos del flujo descrito en el enunciado.
  - Se muestra el número de jugador en la sala de espera. [**1 puntos**]
  - El juego comienza con el *switch* de inicio. [**3 puntos**]
  - El juego realiza la cuenta regresiva correctamente, partiendo en 3 y terminando en 0. [**3 puntos**]

- El juego detecta al ganador correctamente. **[4 puntos]**
- El juego muestra en el *display* el número del jugador ganador; y en los LEDs los milisegundos de su tiempo de reacción. **[4 puntos]**
- Al bajar el *switch* de inicio, se vuelve a la sala de inicio y se reinicia el flujo, permitiendo iniciar otra partida. **[5 puntos]**

▪ **Flujo de Juego creado por usted con uso de VGA:** Se cumple con las restricciones dadas en el enunciado correspondiente. **[30 puntos]**

Cálculo de nota entrega: $round((10 + PuntajeObtenido)/10,\ \ 2)$