Arquitectura de Computadoras 2017

Laboratorio 1 - MIPS en VHDL

Objetivos

- Desarrollar códigos en lenguaje VHDL para describir circuitos secuenciales y combinacionales vistos en el teórico y el práctico.
- Utilizar la herramienta QUARTUS para analizar y sintetizar el código VHDL.
- Aprender a reutilizar código VHDL mediante componentes.
- Mediante el uso del *University Program VWF*, analizar las formas de onda y testear el los resultados obtenidos.
- Implementar un microprocesador MIPS en una FPGA y verificar su correcto funcionamiento utilizando su interfaz de salida visual (LEDs).

Condiciones

- Realizar el trabajo práctico en grupos de **3 personas**.
- Entregar el trabajo resuelto antes del **jueves 19 de Octubre** a delfinavelez@gmail.com. Los trabajos entregados después de esa fecha se consideran desaprobados.
- Defender en un coloquio el trabajo presentado. Deben presentarse todos los integrantes del grupo y responder preguntas acerca del trabajo realizado. El coloquio es el **viernes 20 de Octubre**, en el horario del práctico. En caso de no poder asistir al coloquio en esa fecha, coordinar **previamente** con los docentes.
- El trabajo práctico y su coloquio deben estar aprobados para obtener la promoción de la materia (y deberán aprobar al menos un laboratorio para obtener la regularidad).

Formato de entrega

- Deben entregar un archivo comprimido (tar, zip, rar, etc.), con el nombre: Lab1_ApellidoNombre1_ApellidoNombre2_ApellidoNombre3.tar.gz, respetando mayúsculas y minúsculas.
- Se deben utilizar los mismos nombres de los componentes de alto nivel pedidos.
- Junto con el código, deben entregar un pequeño informe (en formato pdf) en el cual se explique la estructuración del código (entidades de mayor nivel), decisiones de diseño tomadas (si las hubiere), dificultades con las que se encontraron y cómo las resolvieron, testbench, etc.
- No está permitido compartir código entre grupos.

Calificación

El laboratorio no lleva nota numérica, solo aprobado (A) si se cumplen con las actividades básicas propuestas, o desaprobado (D) si no se cumplieran. En caso de resolver el Desafío extra, la clasificación será de (A+) y será tenido en cuenta a favor al momento del cierre de los promedios para las condiciones de promoción y regularidad.

Para aprobar, el código debe realizar la tarea pedida, que será corroborada grabando el MIPS en la FPGA y verificando la correcta ejecución de los programas. Finalmente, todos los integrantes del grupo deberán defender el trabajo oralmente.

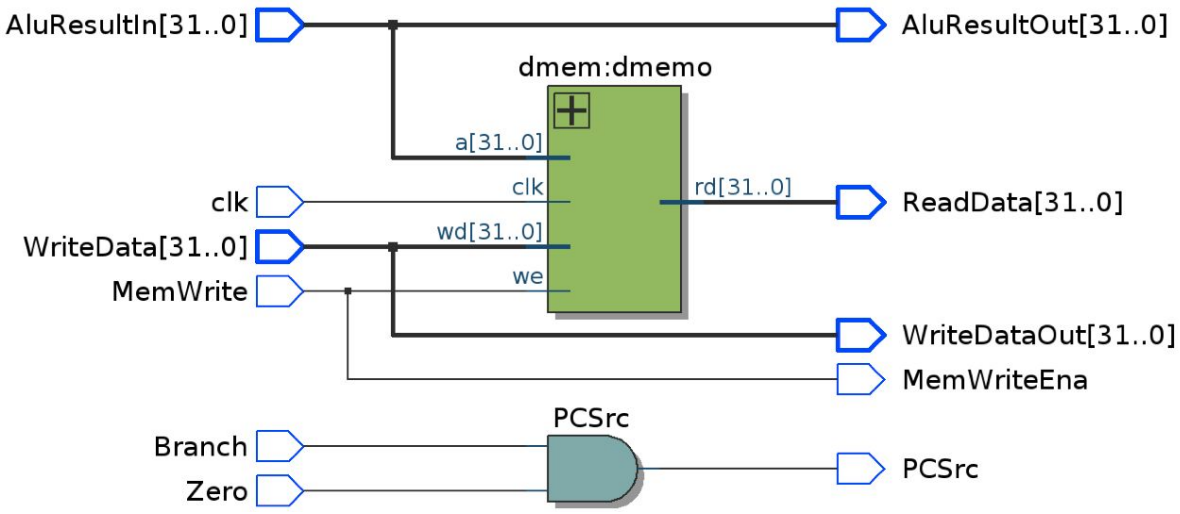
Aunque no se califica: estilo de código, simpleza, elegancia, comentarios adecuados y velocidad; si el código está muy por afuera de los parámetros aceptables, se podrá desaprobar el trabajo aunque sea funcionalmente correcto.

PROYECTO

El proyecto se basa en el desarrollo e implementación de un microprocesador MIPS con *Pipeline*, en una versión simple, sin control de *hazards*. Partiendo del código de la versión *single cycle* culminada en el práctico 3, se pide que introduzcan las modificaciones necesarias en el diseño realizado para introducir el pipeline e implementar el MIPS en la FPGA *Cyclone IV EP4CE22F17C6N.*

Implementación de la interfaz de salida visual (LEDs)

Descargar del enlace el proyecto a utilizar que ya contiene la lógica de interfaz visual. En el archivo top (DE0_NANO.vhd) instanciar el MIPS realizado en los practico 1 a 3. Es necesario realizar algunas modificaciones para que esta lógica funcione. Primero se deben agregar como puertos de salida, en la jerarquia memory, las señales de address, enable y data de la memoria, como se muestra en la siguiente figura:



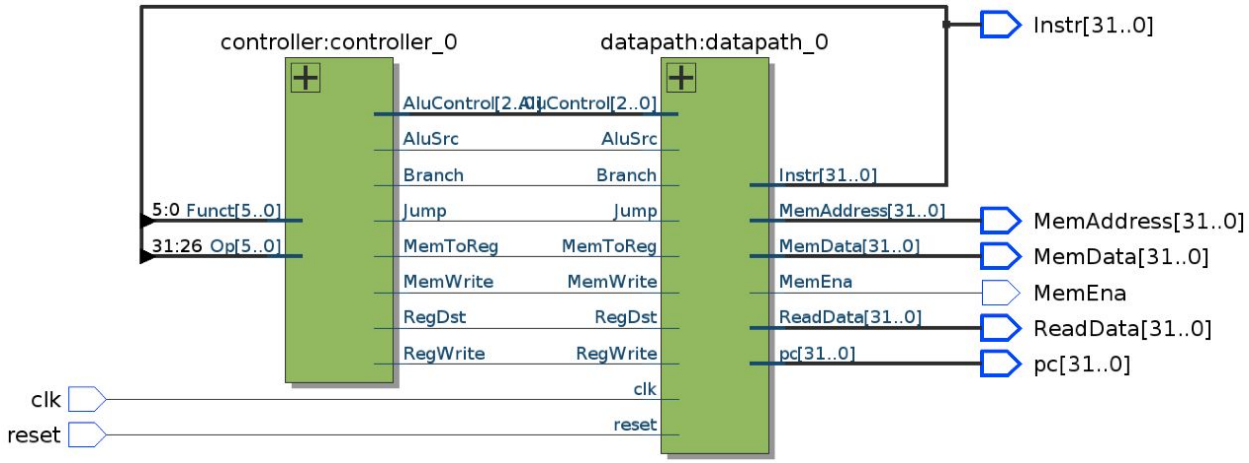
Luego, en la jerarquía del *Datapath,* crear puertos de salida para estas señales, respetando los nombres:

MemData <= WriteDataOut

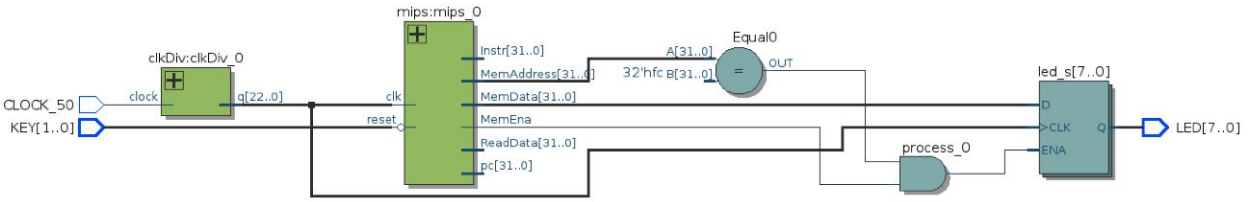
MemAddress <= aluResult

MemEna <= MemWriteEna

Dichos puertos también deben crearse en la jerarquía top, como se muestra en la imagen:



Los puertos utilizados para la simulación (pc, instr, ReadData) deben quedar desconectados en la nueva jerarquía top. La instanciación del MIPS se muestra a continuación:

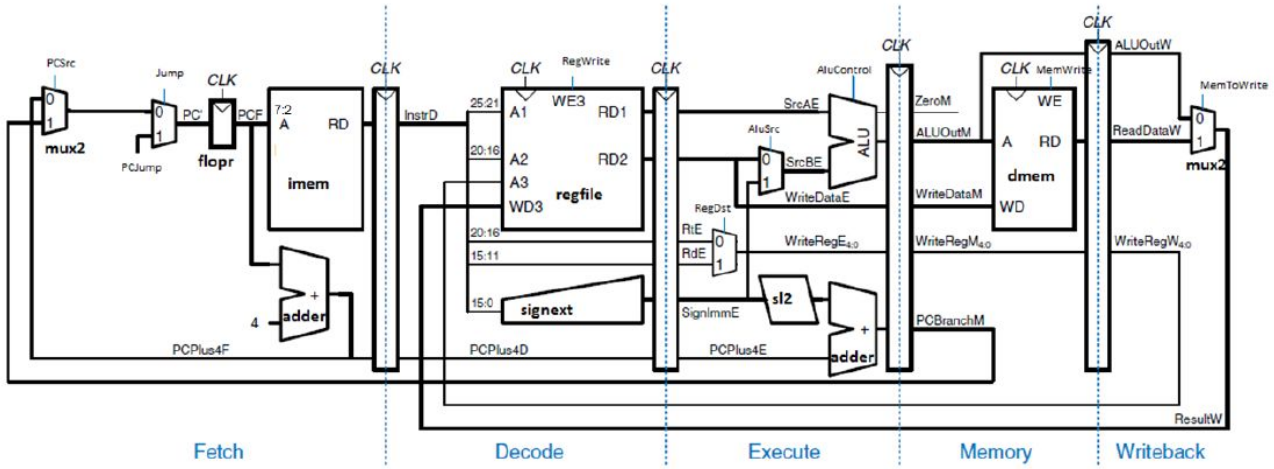


Los puertos declarados son utilizados por la lógica de interfaz de salida ya implementada en el proyecto. Cuando se ejecuta una instrucción **sw** con dirección de la última posición de memoria (0x64) el dato se almacena en los flip-flop *led_s*. A la salida de dichos registros se conectan los leds de la placa de desarrollo.

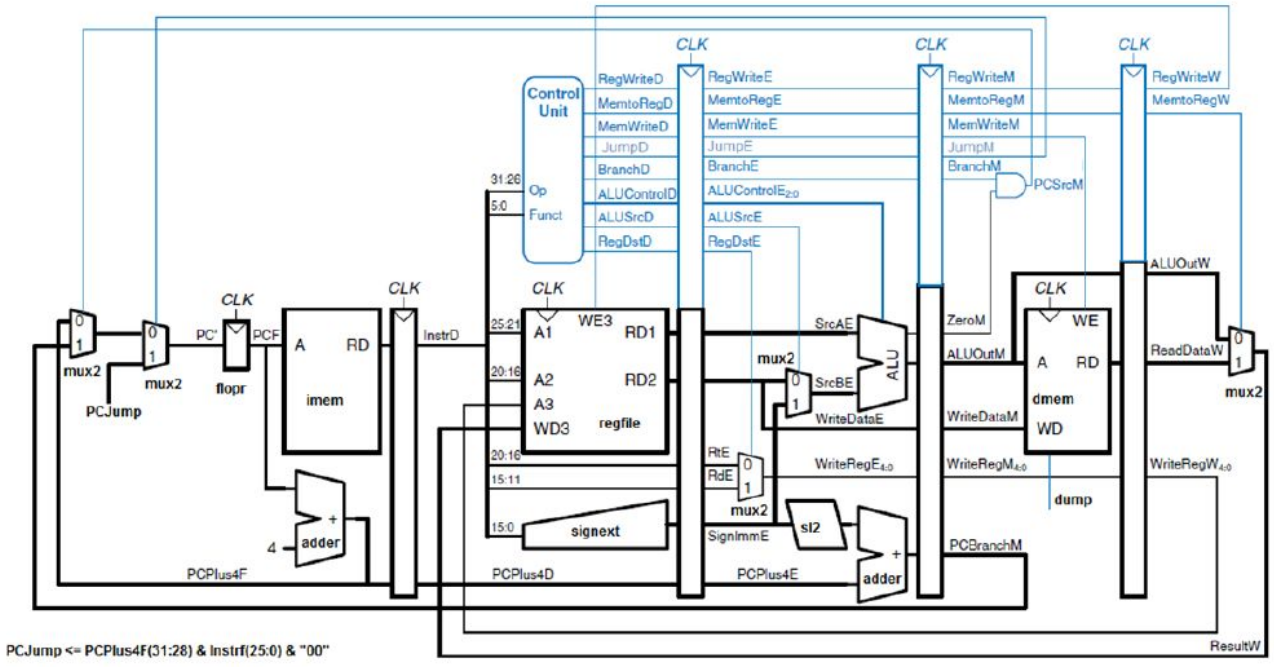
Observar en el diagrama que, además, se agrega un bloque llamado *clkDiv* con el objetivo de disminuir la velocidad de trabajo del microprocesador y poder observar el encendido y apagado de los leds.

Recomendaciones

Antes de agregar *Pipeline*, probar que el diseño del MIPS *single cycle* (del práctico 3) funcione correctamente . Luego, agregar la división en etapas DENTRO del componente "*datapath*" según el diagrama:



Recuerden que deben agregar las señales de control al *pipeline*:



Elaboración 1

Cargar el siguiente programa en su MIPS con *pipeline* y analizar su funcionamiente utilizando el *University Program VWF*.

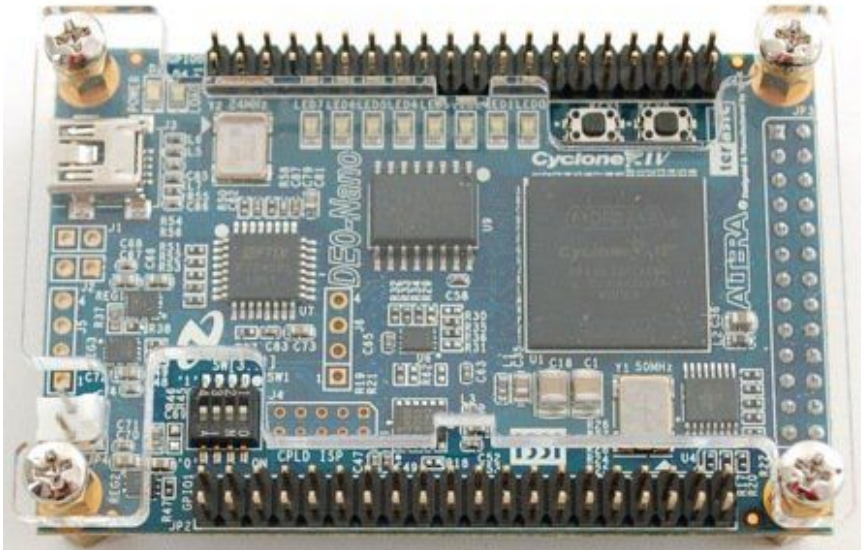
```
addi $t0, $zero, 1
addi $t1, $t0, 2
addi $t2, $t1, 2
addi $t3, $t2, 2

sw $t0, 0($zero)
sw $t1, 4($zero)
sw $t2, 8($zero)
sw $t3, 12($zero)

exit: beq $zero $zero exit
```

Se observa que el programa no funciona correctamente por la presencia de *hazards.*

a) Qué tipos de *hazards* se producen?
b) Una técnica común para evitar *pipeline stalls* es el reordenamiento estático. Puede utilizar esta técnica para evitar los *stalls*? Justifique su respuesta.
c) Puede evitarlos con instrucciones “nop”? Justifique su respuesta.
d) Re-escribir el código de la manera que se considere más conveniente para solucionar los *hazards* y probar su correcto funcionamiento en el MIPS, utilizando el *University Program VWF*.

Elaboración 2

Desarrollar un código simple que aproveche el uso de los leds de la placa para generar un efecto visual deseado. Tener en cuenta que no hay implementado ningun tipo de control de Hazards.



Desafío extra (no obligatorio)

Como se desprende de los diagramas de las actividades propuestas, el MIPS a implementar no tiene la capacidad de detectar la ocurrencia de Hazards de ningún tipo. Es sabido que el hazard de ocurrencia más frecuente es el *data hazard*, y es por esto que se propone la implementación de un bloque de detección de hazards (*Hazard detection unit*) a fin de poder insertar stalls en forma automática hasta que dicha condición desaparezca. Vale aclarar que esto NO IMPLICA la implementación de la lógica necesaria para la técnica del *data forwarding*.

Algunas aclaraciones respecto a la implementación:

- El *Hazard detection unit* debe implementarse en la instancia del *Instruction decode* (ID) y las diversas condiciones para la detección de un data hazzard están tratadas en el capítulo 4.7- Data Hazards: Forwarding vs Stalling del libro “Computer Organization and Design” de D.Patterson y J. Hennessy. Se deben considerar TODAS las condiciones para los tres tipos de dependencias:
    - En ciclo de EX entre REG-REG (entre instrucciones tipo R en 1er orden)
    - En ciclo de DM entre REG-REG (entre instrucciones tipo R en 2er orden)
    - En ciclo de DM entre MEM-REG (entre instrucciones tipo lw y tipo R)
- Esta unidad de detección debe, al encontrar una condición de hazzard verdadera, generar la condición de *stall* en el procesador a partir del ciclo ID, esto es:
    - Evitar que el PC avance a la siguiente instrucción en el siguiente CLK (congelar su valor)
    - Evitar que el registro de pipeline IF/ID cambie de valor en el siguiente CLK (congelar su valor)
    - Forzar a que todas las señales de control a partir del ciclo EX en adelante valgan “0” (Ver implementación de referencia en Fig 4.60, pág. 316)

Para corroborar el correcto funcionamiento de la unidad de detección de hazard, generar un programa que incluya los tres tipos de dependencias antes mencionadas y simular el comportamiento del MIPS por medio de la herramienta *University Program VWF*, dando visibilidad a las señales de salida generadas por la unidad de detección.