Laboratorio ARMv8 en SystemVerilog

Objetivos

- Desarrollar códigos en lenguaje SystemVerilog para describir circuitos secuenciales y combinacionales vistos en el teórico y el práctico.
- Utilizar la herramienta Vivado para analizar y sintetizar el código SystemVerilog.
- Aprender a reutilizar código SystemVerilog mediante módulos estructurales.
- Mediante el uso de test bench, analizar las formas de onda y testear los resultados.
- Aplicar los conceptos aprendidos sobre microprocesadores y la técnica de mejora de rendimiento: segmentación de cauce (*pipeline*).

Condiciones

- Realizar el trabajo práctico en los grupos de **3 personas** ya creados.
- Crear un tag en el repositorio que se llame EntregaLab1. La fecha límite de entrega es el **martes 11 de noviembre** (inclusive). No se considerarán modificaciones después de esa fecha.

Formato de entrega

- Utilizar los nombres de los módulos y señales indicados en las guías del práctico.
- Al repositorio se debe subir el proyecto de Vivado completo y el readme que funcionará de informe (**en formato md**). Éste último debe incluir el nombre de los integrantes del laboratorio, una aclaración de qué ejercicios resolvieron y los requerimientos descriptos al final de cada ejercicio.
- El trabajo es incremental, se pide que se entregue un único procesador con todas las modificaciones correspondientes a los ejercicios que se decidieron resolver.
- No está permitido compartir código entre grupos.
- No está permitido subir el código en repositorios públicos.
- El código se entrega mediante el repositorio creado por la cátedra mediante tags: para el ejercicio 1: “LAB_ej1”, para el ejercicio 2: “LAB_ej2”.
- Con respecto al repositorio, pueden usar distintos branch si lo desean, pero todo código entregable debe estar en la principal.

Calificación

El ejercicio 1 es obligatorio. Su resolución debe estar aprobada para obtener la regularidad de la materia. Quienes resuelvan el ejercicio 2 (partiendo del procesador modificado en el ejercicio 1) estarán habilitados para promocionar, si cumplen con los demás requisitos.

***Importante*:** verificar que no se produzcan advertencias de la herramienta durante el proceso de síntesis relacionadas con una mala interpretación del circuito a implementar.

DESARROLLO

El laboratorio está basado en la implementación del microprocesador ARMv8 en versión reducida obtenida en el trabajo práctico 2, antes de comenzar el diseño crear un tag que se llama "uP-SingleCycle". Luego, descargar de Moodle el set de archivos <PipelinedProcessorPatterson-Modules> con la descripción de algunos módulos del procesador con pipeline y reemplazar los archivos con el mismo nombre del repositorio. Finalmente verificar las conexiones resultantes según los diagramas de las figuras 1 y 2.

Recomendaciones importantes:

- Recordar inicializar los registros X0 a X30 con los valores 0 a 30 respectivamente en la implementación del bloque *#regfile*, o con los valores que consideren necesarios para la resolución de los problemas.

- Se debe modificar el bloque *#decode* a fin de agregar el puerto de entrada resaltado con un círculo rojo en la Fig. 2.

- Se debe modificar el bloque *#regfile* a fin de que si alguno, o ambos registros leídos por una instrucción en la etapa *#decode,* están siendo escritos como resultado de una instrucción anterior en la etapa *#writeback* se obtenga a la salida de *#regfile* el valor actualizado del registro.

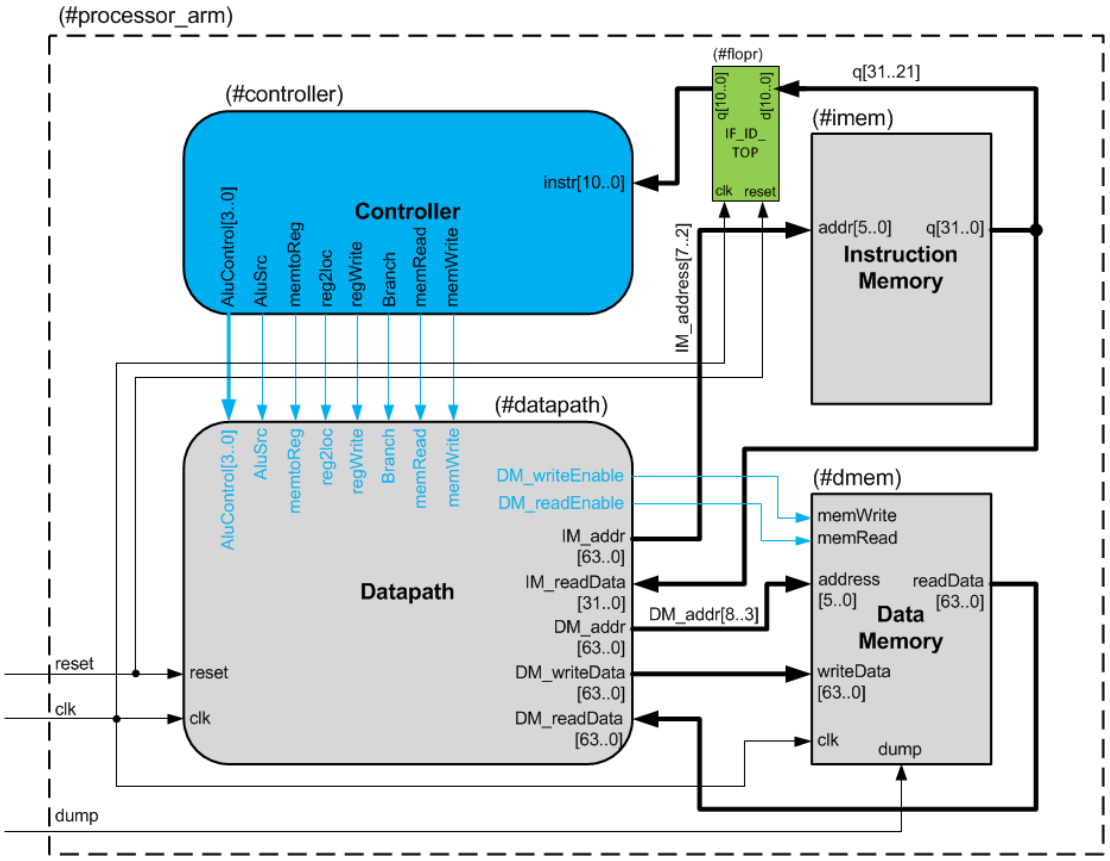


Figura 1: Esquema del *top_level_entity*

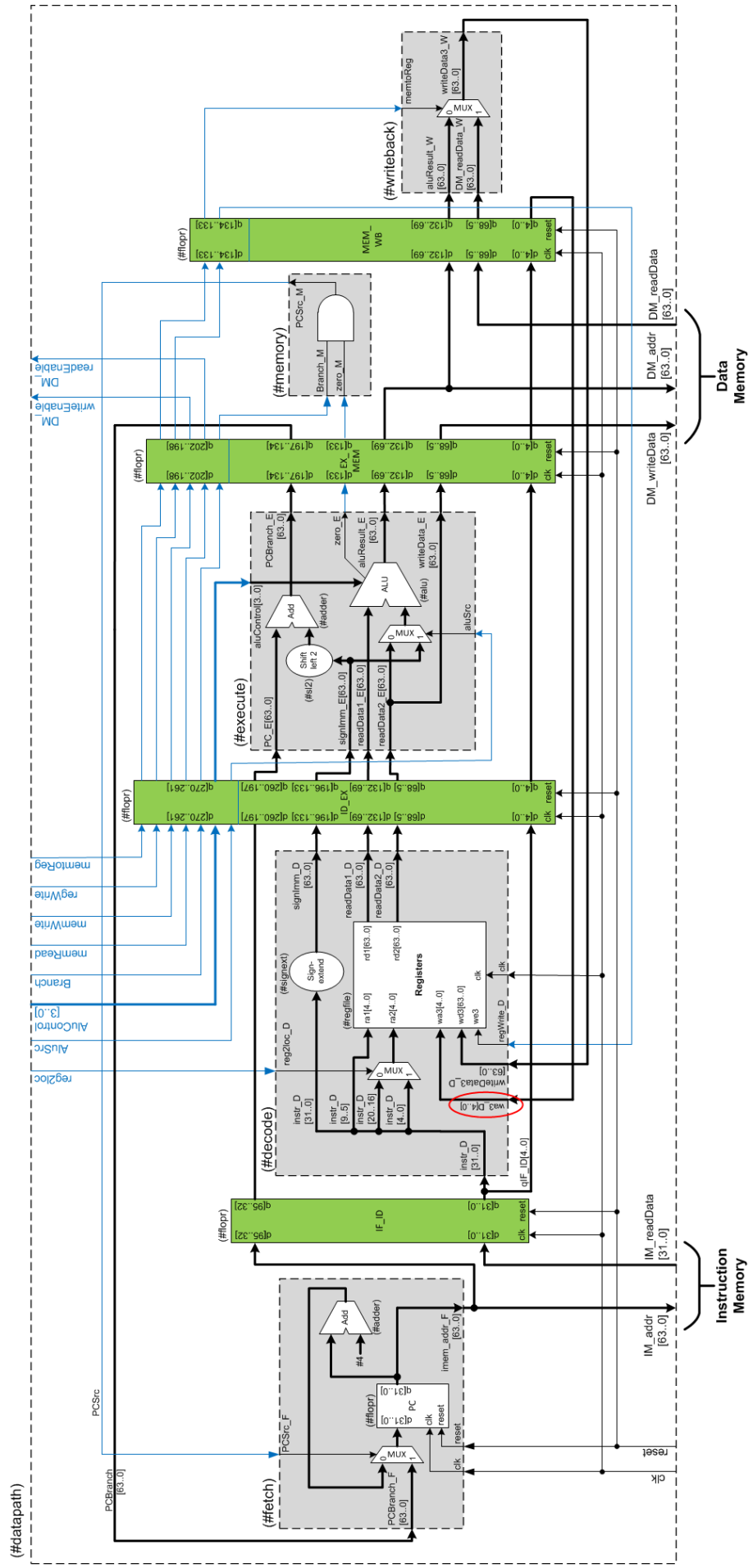


Figura 2: Esquema del *datapath*

Una vez concluida la implementación del procesador completo con pipeline, se debe verificar su correcto funcionamiento utilizando el siguiente código:

```
                                // Dirección:valor
            STUR X1, [X0, #0]     // MEM 0:0x1
            STUR X2, [X0, #8]     // MEM 1:0x2
            STUR X3, [X16, #0]    // MEM 2:0x3
            ADD X3, X4, X5
            STUR X3, [X0, #24]    // MEM 3:0x9
            SUB X3, X4, X5
            STUR X3, [X0, #32]    // MEM 4:0xFFFFFFFFFFFFFFFF
            SUB X4, XZR, X10
            STUR X4, [X0, #40]    // MEM 5:0xFFFFFFFFFFFFFFF6
            ADD X4, X3, X4
            STUR X4, [X0, #48]    // MEM 6:0xFFFFFFFFFFFFFFF5
            SUB X5, X1, X3
            STUR X5, [X0, #56]    // MEM 7:0x2
            AND X5, X10, XZR
            STUR X5, [X0, #64]    // MEM 8:0x0
            AND X5, X10, X3
            STUR X5, [X0, #72]    // MEM 9:0xA
            AND X20, X20, X20
            STUR X20, [X0, #80]   // MEM 10:0x14
            ORR X6, X11, XZR
            STUR X6, [X0, #88]    // MEM 11:0xB
            ORR X6, X11, X3
            STUR X6, [X0, #96]    // MEM 12:0xFFFFFFFFFFFFFFFF
            LDUR X12, [X0, #0]
            ADD X7, X12, XZR
            STUR X7, [X0, #104]   // MEM 13:0x1
            STUR X12, [X0, #112]  // MEM 14:0x1
            ADD XZR, X13, X14
            STUR XZR, [X0, #120]  // MEM 15:0x0
            CBZ X0, L1
            STUR X21, [X0, #128]  // MEM 16:0x0(si falla CBZ =21)
L1:         STUR X21, [X0, #136]  // MEM 17:0x15
            ADD X2, XZR, X1
L2:         SUB X2, X2, X1
            ADD X24, XZR, X1
            STUR X24, [X0, #144]  // MEM 18:0x1 y MEM 19=0x1
            ADD X0, X0, X8
            CBZ X2, L2
            STUR X30, [X0, #144]  // MEM 20:0x1E
            ADD X30, X30, X30
            SUB X21, XZR, X21
            ADD X30, X30, X20
            LDUR X25, [X30, #-8]
            ADD X30, X30, X30
            ADD X30, X30, X16
            STUR X25, [X30, #-8]  // MEM 21:0xA (= MEM 9)
finloop:    CBZ XZR, finloop
```

TENER EN CONSIDERACIÓN LOS EVENTUALES PROBLEMAS DE HAZARD (de datos y de control) QUE CONTIENE EL PROGRAMA UTILIZADO Y PLANTEAR LAS MODIFICACIONES DE CÓDIGO NECESARIAS PARA EVITAR SU OCURRENCIA. Para esto, agregar instrucciones tipo “nop”, las cuales se pueden implementar como ADD XZR, XZR, XZR.

Ejercicio 1 (regularidad)

Sin afectar el funcionamiento de las instrucciones ya implementadas en la versión reducida del microprocesador con pipeline, **agregar** las instrucciones ***LSL*** y ***LSR (Logical Shift)***. Introducir en el procesador todas las modificaciones necesarias, tanto en el datapath como en las señales de control.

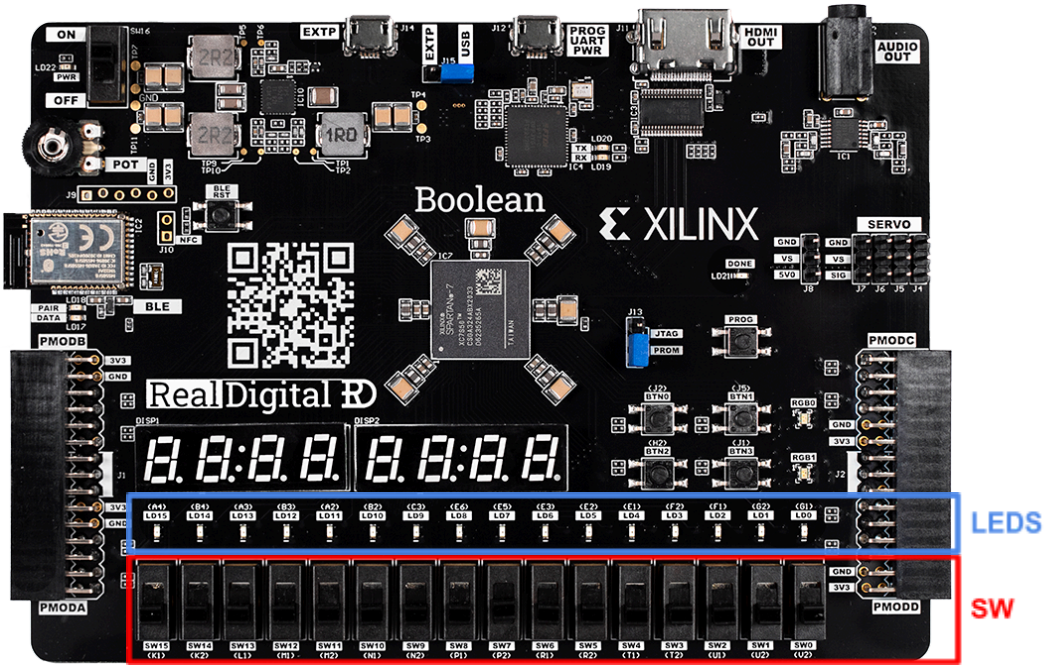
Una vez finalizadas las modificaciones, agregar al código dado (en la página anterior) instrucciones *LSL* y *LSR* con distintos valores numéricos y verificar su correcta implementación. Es decir, se debe analizar que todo el set de instrucciones continúe funcionando correctamente y también las instrucciones agregadas (para esto pueden escribir los resultados en memoria y verificar si obtienen los valores correctos en “mem.dump” al finalizar la ejecución del código). No olvidar resolver los eventuales problemas de hazard agregando instrucciones “nop”.

Estas dos instrucciones nuevas serán muy útiles para implementar la segunda parte de este ejercicio. Se debe elaborar un nuevo programa en assembler para gestionar recursos de E/S agregados a nuestro micro. En este caso se han mapeado en memoria (a partir de la direccion 0x8000) los registros de control y status de un bloque GPIO (General Purpose I/O) que tiene conectados un arreglo de 16 leds y 16 llaves según el siguiente detalle:

Register (Address): **GPIO Otput (0x008000)**									
31	**…**	**17**	**16**	**15**	**…**	**3**	**2**	**1**	**Bit 0**
N/U	…	N/U	N/U	LED15	…	LED3	LED2	LED1	LED0: State: “1” → on / “0” → off

Register (Address): **GPIO Input (0x008008)**									
31	**…**	**17**	**16**	**15**	**…**	**3**	**2**	**1**	**Bit 0**
N/U	…	N/U	N/U	SW15	…	SW3	SW2	SW1	SW0: State: “0” → open / “1” → close

El objetivo de este nuevo código assembler es generar un juego simple o algún efecto llamativo utilizando la totalidad de los leds (distribuidos en línea) y las llaves que se consideren necesarias. Tengan en consideración los tiempos en que los leds permanecen encendidos y apagados en la generación del efecto. Les recomendamos que implementen lazos de espera parametrizables a fin de poder ajustar dichos tiempos al momento de probarlos en la placa FPGA:



Para el readme:

- Describir brevemente qué modificaciones se introdujeron (en qué entidades y con qué finalidad). Mostrar el diagrama del nuevo microprocesador, indicando las señales y entidades agregadas (de ser necesario).
- Mostrar el programa en assembler LEGv8 modificado que se utilizó para verificar el comportamiento del procesador.
- Mostrar el nuevo programa en assembler LEGv8 que implementa el juego o efecto visual sobre los leds y una breve explicación de su funcionamiento.

Ejercicio 2 (para promoción)

El procesador LEGv8 desarrollado no tiene la capacidad de detectar la ocurrencia de *hazards* de ningún tipo. En este ejercicio se propone la implementación de un bloque de detección de *hazards* (*Hazard Detection Unit*) y otro de *forwarding (Forwarding Unit),* a fin de aplicar la técnica de *forwarding-stall* en caso de la ocurrencia de un ***data hazard****,* hasta que el mismo desaparezca.

Algunas aclaraciones respecto a la implementación:

- La *HDU* debe implementarse en la instancia del *Instruction decode* (ID).
- Las diversas condiciones para la detección de un *data hazard* se analizan en el capítulo 4.7- “Data Hazards: Forwarding vs Stalling” del libro “Computer Organization and Design - ARM Edition” de D.Patterson y J. Hennessy. Se deben considerar TODAS las condiciones para todos los tipos de dependencias de datos.
- Para generar la condición de *stall* en el procesador es necesario:
    - Evitar que el PC avance a la siguiente instrucción en el siguiente CLK y evitar que el registro de pipeline IF/ID cambie de valor en el siguiente CLK

(congelar su valor). Para esto deberán diseñar una nueva entidad **FLOPRE** similar al **FLOPR**, pero agregando una señal de *enable* (habilitación). Funcionamiento: *enable* = 1 el funcionamiento es normal, *enable* = 0 no altera el valor de salida al detectar un flanco de CLK (síncrono).

- Forzar que todas las señales de control a partir del ciclo EX en adelante tomen el valor “0” (Ver implementación de referencia en Fig 4.59).

- ¡No olvidar que una parte del registro IF/ID está en la entidad *#processor_arm*!

Para el readme:

- Mostrar un diagrama de bloques general del nuevo procesador, indicando las señales y módulos agregados.
- Correr nuevamente el código desarrollado para el ejercicio 1, sin el agregado de instrucciones “nop”, y tomar una captura de la pantalla del simulador de Vivado donde se vea un caso de *stall* de instrucciones y otra donde se vea el *forwarding* de datos entre instrucciones. Explicar brevemente qué se observa en la imagen.