Desarrollo de Hardware Digital
Práctica 1

Diseño de un procesador en VHDL
Análisis y optimización.
Ampliación del repertorio de instrucciones

Objetivos:

- Comprender la descripción inicial en VHDL sintetizable de un procesador sencillo de 4 instrucciones.
- Conocer la temporización de las operaciones de lectura y escritura en la memoria del computador elemental basado en dicho procesador.
- Analizar las prestaciones en velocidad del diseño y proponer soluciones de mejora.
- Realizar la síntesis automática del computador elemental.
- Observar el consumo de recursos de la FPGA.
- Realizar la verificación del diseño mediante simulación, análisis temporal y visualización de esquemas resultantes de la síntesis.
- Especificar en VHDL sintetizable versiones progresivamente más complejas del procesador ampliando el repertorio de instrucciones

1. Introducción

Se va a implementar en la FPGA de la DE2-115 un computador elemental, cuya estructura de bloques se muestra en la Figura 1. En esta práctica se genera e instancia el core IP de la memoria, se realiza un análisis detallado del procesador de partida (capaz de ejecutar sólo cuatro instrucciones), se generan nuevas versiones optimizadas para mejorar sus prestaciones y se propone finalmente una ampliación del repertorio de instrucciones.

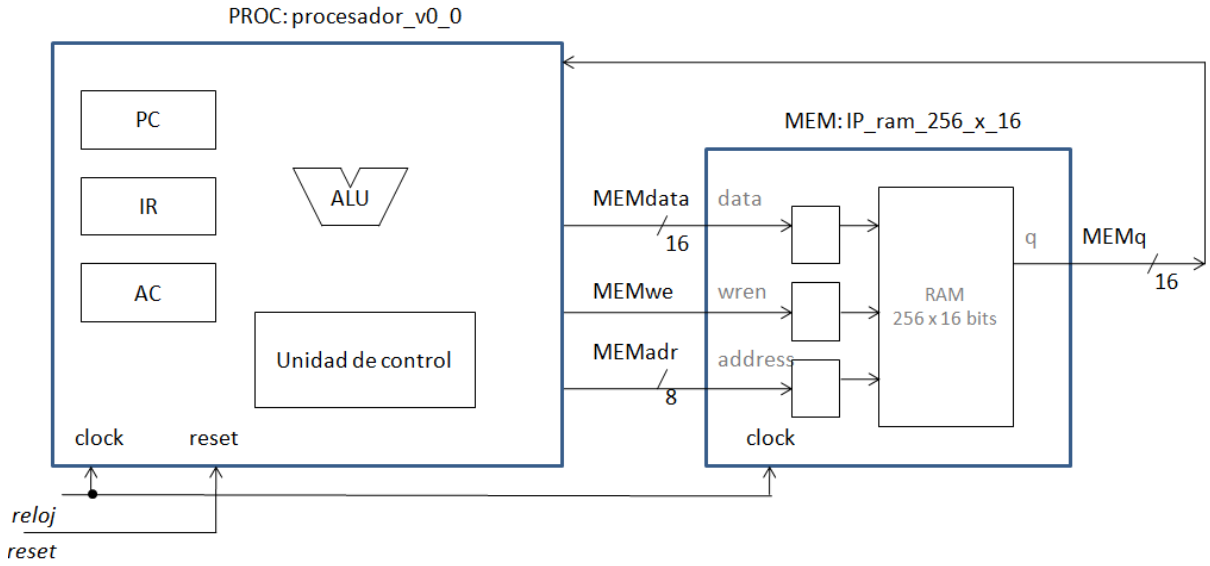

Figura 1: Esquema de bloques del computador elemental. Los componentes `procesador_v0_0` e `IP_ram_256_x_16` se instancian en la descripción `my_scomp_v0_0.vhd`. En la figura se han omitido los puertos de salida que muestran los contenidos de los registros AC, IR y PC.

2. Repertorio de instrucciones y redacción de programas

El procesador de partida puede ejecutar las cuatro primeras instrucciones de la Tabla 1 del Apéndice. El repertorio inicial de instrucciones corresponde al del computador elemental presentado en el capítulo 9 de [1]. Cada instrucción ocupa 16 bits. En la versión original, los 8 bits más significativos corresponden al código de operación (*CODOP*) y los restantes a una dirección (*address*).

Redactar en hexadecimal un programa que sume el contenido de tres posiciones consecutivas de memoria, y almacene el resultado en la posición de memoria siguiente. Crear con la opción *New → Memory Files → Memory Initialization File* un fichero denominado `programa.mif` para almacenar el contenido de una memoria de 256 palabras y 16 bits por palabra. Almacenar el programa a partir de la dirección 00h, y los datos en las posiciones consecutivas que se hayan asignado. NOTA: Puede resultar conveniente que tanto el contenido como las direcciones de memoria se muestren en hexadecimal ajustando las opciones *View → Memory Radix* y *View → Address Radix*, respectivamente.

3. Generación e instanciación del módulo de memoria

La memoria de datos e instrucciones del procesador se implementa en los bloques de memoria RAM embebidos [2]. Para ello, se puede a) realizar una descripción funcional de la memoria siguiendo las recomendaciones de estilo para Quartus II en [3]; b) instanciar en la descripción del computador elemental el componente `altsyncram` de la biblioteca `altera_mf` dando los valores oportunos a sus parámetros; o c) utilizar un catálogo de cores IP configurables y optimizados para las FPGAs de Altera y generar mediante un asistente un módulo de memoria con las características deseadas para luego instanciarlo en el diseño.

Emplearemos el asistente para generar un *core* IP (denominado `IP_ram_256_x_16`) correspondiente a una memoria RAM de 1 sólo puerto con 256 palabras y 16 bits por palabra. Las entradas de datos, direcciones y control de escritura deben estar REGISTRADAS. Elegiremos que las salidas de datos de la memoria sean NO REGISTRADAS. Por tanto, en un ciclo se ordena la operación (lectura/escritura), se indica la dirección y el dato (si es escritura), y después del flanco (en el ciclo siguiente) se efectúa la operación en el bloque RAM.

Realizar las siguientes instrucciones para generar el componente:
- Iniciar el asistente desde *Tools → MegaWizard Plug-In Manager*
- Crear un nuevo componente seleccionando *Create a new custom megafunction variation*
- Seleccionar *Installed Plug-Ins → Memory Compiler → RAM: 1-PORT,* familia Cyclone IV E y lenguaje VHDL. El nombre del fichero de salida debe ser `IP_ram_256_x_16` y ha de estar ubicado en el directorio del proyecto.
- A partir de este momento, se van indicando las características del componente. Se pueden dejar por defecto todas las opciones excepto lo siguiente: indicar que el bus de salida 'q' es de 16 bits (primera ventana), desmarcar la celda del bus de salida 'q' para que no sea registrado (segunda ventana) e indicar que el contenido inicial de la memoria se establece en el fichero `programa.mif` ubicado en el directorio del proyecto (cuarta ventana).

Una vez finalizado el proceso, se han generado en el directorio los ficheros `IP_ram_256_x_16.vhd` (con una instancia del componente `altsyncram` con los valores de sus parámetros correspondientes a las características que se han seleccionado con el asistente); `IP_ram_256_x_16.qip` (que contiene información para poder integrar y compilar el core IP en el proyecto) y `IP_ram_256_x_16.cmp` (que incluye la declaración del componente tal como debería incluirse en una descripción VHDL para luego ser instanciado).

Editar el fichero `my_scomp_v0_0.vhd` para declarar el componente de la memoria, e instanciarlo conectando sus puertos según se indica en la Figura 1.

Dibujar un cronograma que muestre los valores de las diferentes señales que se conectan a la memoria (reloj, bus de entrada de datos, bus de direcciones, entrada de control de escritura y bus de salida de datos) para realizar consecutivamente dos operaciones de escritura seguidas de dos operaciones de lectura (se leen las dos direcciones de memoria en las que se haya escrito previamente, en el mismo orden).

4. Análisis y verificación de la versión inicial del procesador

Crear un proyecto para la descripción `my_scomp_v0_0.vhd` (entidad de nivel superior: `my_scomp_v0_0`; ficheros del proyecto: `my_scomp_v0_0.vhd`, `procesador_v0_0.vhd` e `IP_ram_256_x_16.qip`; dispositivo: `EP4CE115F29C7`; herramienta EDA para simulación: `ModelSim-Altera`, lenguaje `VHDL`).

Analizar la descripción VHDL del componente `procesador_v0_0` y dibujar el diagrama de estados de la unidad de control del procesador en su versión inicial. Junto a cada estado se deben indicar las operaciones que se realizan en el correspondiente ciclo de reloj. NOTA: Observar que en el proceso `FSMD` de la descripción `procesador_v0_0.vhd` se distinguen las asignaciones que se realizan a registros (sincronizadas con el flanco de reloj) de las asignaciones realizadas a los buses de entrada a la memoria (que generan lógica combinacional, dado que los registros se han generado dentro del componente IP).

Realizar la síntesis RT-lógica *(Processing → Start → Analysis y Synthesis, Ctrl-K)* y visualizar la netlist resultante de la síntesis RT inicial *(Tools → Netlist Viewers → RTL Viewer)*. Localizar los registros que se infieren. Identificar la memoria RAM, y comprobar los estados que aparecen en el diagrama de estados de la FSM (bloque amarillo).

Realizar con Modelsim-Altera una simulación funcional del diseño `my_scomp_v0_0.vhd`. Definir las formas de onda para los estímulos de entrada y simular para obtener los resultados de la ejecución del programa `programa.mif` realizado en el apartado 2. Se deben visualizar al menos todos los puertos de E/S del computador elemental y el valor del estado del procesador. NOTA: Se puede añadir a las formas de onda la señal `state` localizándola en la ventana izquierda al desplegar la instancia `PROC` del procesador y seguidamente el proceso `FSMD`.

5. Evaluación de prestaciones y optimización

Compilar el diseño *(Processing → Start Compilation, Ctrl-L)* y examinar en el informe de compilación el porcentaje de ocupación de los recursos de la FPGA. Consultar también la frecuencia máxima estimada por el analizador temporal en la opción *TimeQuest Timing Analyzer → Slow 1200mV 85C Model → Fmax Summary* del informe de compilación, y calcular el tiempo de ejecución del programa que se diseñó en el apartado 2.

A partir de la versión anterior, reducir el número de ciclos por instrucción y generar una nueva versión del procesador (`procesador_v1_0`) eliminando en lo posible estados que únicamente se utilizan para ordenar operaciones con la memoria (reutilizando los estados en el ciclo anterior para asignar valor a los buses de entrada a memoria).

Deducir cuál es la instrucción que limita la frecuencia máxima, proponer alguna solución de mejora, y generar una nueva versión del procesador (`procesador_v1_1`) para comprobar que aumenta la frecuencia máxima estimada.

Analizar cómo se ha visto afectado el porcentaje de ocupación de recursos de la FPGA al generar las dos últimas versiones del procesador, y calcular el porcentaje de mejora en el

tiempo de ejecución del programa que se diseñó en el apartado 2 con respecto a la versión inicial.

6. Ampliación del repertorio de instrucciones

En este apartado se propone una ampliación del repertorio de instrucciones del procesador basada en algunos de los ejercicios propuestos en el capítulo 9 de la referencia [1]. En la Tabla 1 del Apéndice se muestra el repertorio completo de instrucciones del procesador. En relación a la ampliación del repertorio de instrucciones se proponen las siguientes actividades:

6.1. Comenzar la ampliación del repertorio añadiendo a la descripción original las instrucciones SUB y NAND. Redactar un programa de prueba que utilice ambas instrucciones y verificar mediante simulación su correcto funcionamiento.

6.2. Añadir las instrucciones de salto condicional JNEG, JPOS y JZERO. Al redactar los programas de prueba, es preciso considerar tanto casos en los que la condición sea cierta, como casos en los que la condición sea falsa.

6.3. Realizar una nueva ampliación con las instrucciones de desplazamiento lógico a izquierda (SHL) y derecha (SHR). El número de bits que se desplaza el acumulador viene indicado en el argumento "v". Para implementar estas instrucciones pueden resultar útiles las funciones SHL y SHR definidas en el paquete `std_logic_arith`, cuya definición se puede consultar en el fichero `<direct_instalacion_quartus>/libraries/vhdl/synopsys/ieee/syn_arit.vhd`. Comprobar por simulación el correcto funcionamiento de ambas instrucciones.

6.4. Añadir las instrucciones IN y OUT. Cuando se ejecuta la instrucción IN, se almacenan en 8 bits del acumulador los valores presentes en un puerto de entrada denominado `IO_input` (también de 8 bits). Dependiendo del valor del bit menos significativo de la instrucción, se escribe en el byte más significativo del acumulador o en el menos significativo. Análogamente, la instrucción OUT visualiza en un puerto de salida denominado `IO_ouput` el valor actual del byte más significativo o del byte menos significativo del acumulador. Verificar por simulación el correcto funcionamiento de ambas instrucciones.

6.5. ***Opcional:***
Añadir al procesador instrucciones de llamada a subrutina (CALL *address*), retorno de subrutina (RET) y STOP. Comprobar por simulación el correcto funcionamiento de las tres instrucciones. Es recomendable que en los programas de prueba de las llamadas y retornos de subrutina se incluyan llamadas anidadas a subrutinas.

7. Documentación

[1] Hamblen, J.O., Hall T.S., Furman, M.D.: *Rapid Prototyping of Digital Systems : SOPC Edition*, Springer 2008 (Recurso electrónico)

[2] Guía de usuario de las memorias embebidas en FPGAs de Altera. https://www.altera.com/en_US/pdfs/literature/an/an207.pdf

[3] Recomendaciones de estilo para inferir megafunciones en Quartus II. https://www.altera.com/literature/hb/qts/archives/quartusii_handbook_archive_120.pdf *(Capítulo 11: "Recommended HDL Coding Styles")*

Apéndice

Tabla 1: Repertorio de ampliado de instrucciones. *Basada en [1].*

CODOP[1]	Instrucción	Descripcion
00	ADD address	AC[2] <= AC + M(address)
01	STORE address	M(address) <= AC
02	LOAD address	AC <= M(address)
03	JUMP address	PC[2] <= address
04	SUB address	AC <= AC - M(address)
05	NAND address	AC <= AC NAND M(address)
06	JNEG address	Si AC < 0, entonces PC <= address
07	JPOs address	Si AC > 0, entonces PC <= address
08	JZERO address	Si AC = 0, entonces PC <= address
09	SHL v	Desplazamiento lógico a la izquierda de AC el número de posiciones indicado en el argumento "v"
0A	SHR v	Desplazamiento lógico a la derecha de AC el número de posiciones indicado en el argumento "v"
0B	IN v	Si v(0) = '0', entonces AC(7 downto 0) <= SW[3](7 downto 0) Si v(0) = '1' , entonces AC(15 downto 8) <= SW(7 downto 0)
0C	OUT v	Si v(0) = '0', entonces LEDG[3](7 downto 0) <= AC(7 downto 0) Si v(0) = '1' , entonces LEDG(7 downto 0) <= AC(15 downto 8)
0D	CALL address	Llamada a subrutina que comienza en la dirección "address"[4]
0E	RET	Retorno de subrutina[4]
0F	STOP	Detiene la ejecución del programa hasta que se actúa[5] sobre una entrada externa (CONT)

[1] El CODOP se representa en hexadecimal.

[2] AC es el registro acumulador y PC es el contador de programa.

[3] Los puertos de entrada (instrucción IN) y de salida (instrucción OUT) se conectan, respectivamente, a los conmutadores (SW) y LEDs (LEDG) de la tarjeta.

[4] La pila se implementa en la memoria RAM. Debe crecer desde la dirección más alta de memoria, hacia direcciones más bajas.

[5] El procesador espera hasta que se activa la señal CONT, y la ejecución del programa continúa cuando dicha señal deja de estar activa.