CPU Monociclo

Diseño de Procesadores

Carlos Pérez Fino alu0101340333@ull.edu.es
Cheuk Kelly Ng Pante alu0101364544@ull.edu.es

13 de julio de 2024

Índice general

1. Introducción

En el ámbito del diseño y desarrollo de sistemas digitales, las Field-Programmable Gate Arrays (FPGAs) se han convertido en herramientas esenciales gracias a su flexibilidad y capacidad de reconfiguración. Estos dispositivos permiten la implementación de diversos componentes de hardware programables, lo que posibilita la adaptación y optimización de sistemas en tiempo real. Este informe se centra en la ampliación de una Unidad Central de Procesamiento (CPU) en una FPGA utilizando el lenguaje de descripción de hardware Verilog.

2. Decisiones del diseño

En nuestro diseño de nuestra CPU ampliada, nos hemos centrado en cubrir en base a lo aprendido las necesidades que se nos exigen en la realización del proyecto intentando hacer de la manera más sencilla y lógica posible. En este proyecto nos hemos basado en que mediante las interrupciones solicitamos que un led se encienda una vez se mantiene el botón que hemos configurado.

3. Camino de datos

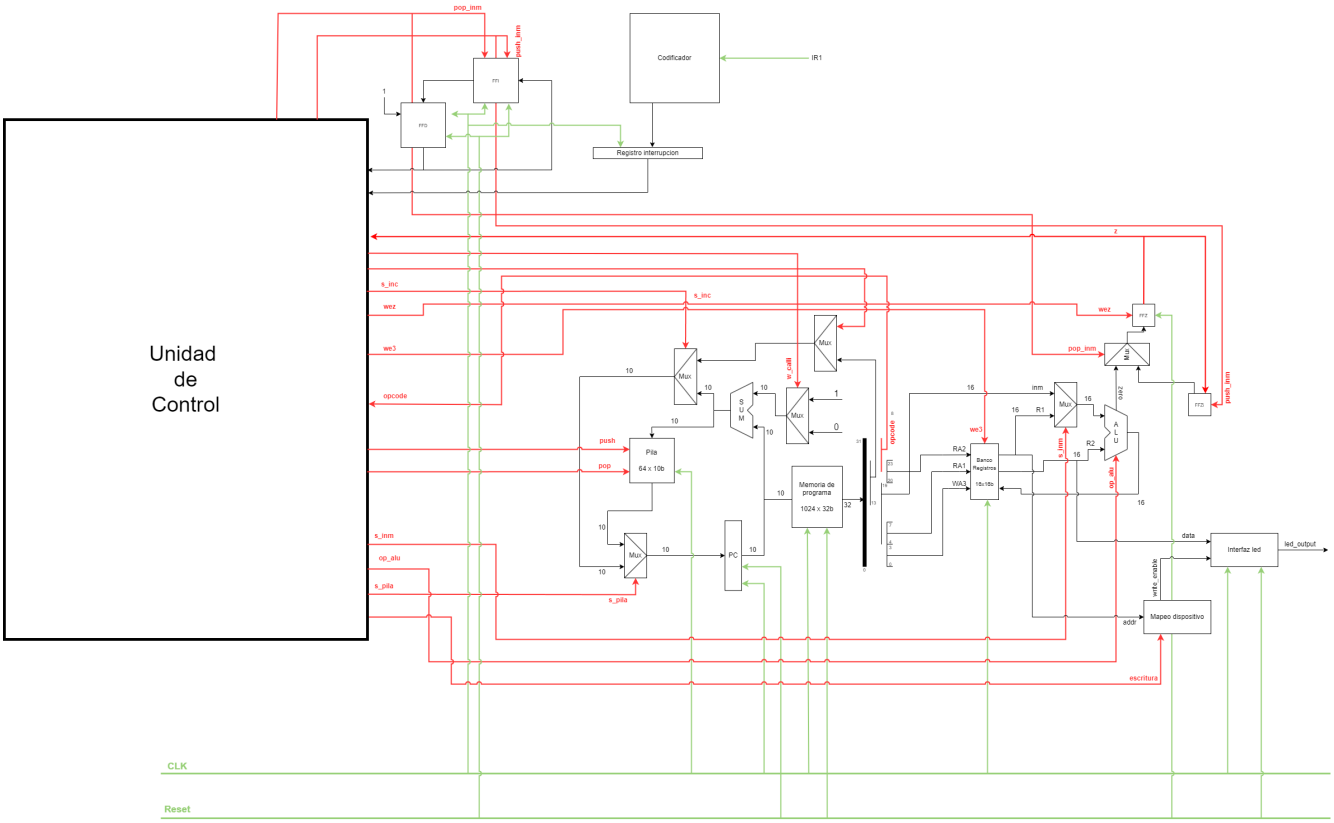


Figura 3.1: Camino de datos

4. Codificación de las instrucciones

Instrucción	OPCODE	Orden en 32 bits
`Aritméticas`	10XXXXXX	0-3: Registro destino 4-7: Registro 1 8-19: X 20-23: Registro 2
`Aritméticas inmediatas`	11XXXXXX	0-3: Registro destino 4-19: Constante 20-23: Registro 2
`Carga inmediata`	00XXXXXX	0-3: Registro destino 4-19: Constante 20-23: Registro 2
`Salto incondicional`	01000100	0-3: Registro destino 4-19: Constante 20-23: Registro 2
`Salto si cero`	01001000	0-13: X 14-23: Salto
`Salto si no cero`	01001100	0-13: X 14-23: Salto
`Llamada a subrutina`	01110000	0-13: X 14-23: Salto
`Retorno de subrutina`	01100000	0-23: X
`Retorno de subrutina de interrupcion`	01111000	0-23: X
`Habilitar escritura entrada/salida`	01000000	0-3: X 4-7: Registro 1 8-19: X 20-23: Registro 2

Cuadro 4.1: Codificación de las instrucciones

- **Aritmético-lógicas:** La operación de la ALU se encuentra en el `opcode[5:2]`, donde el registro 1 y el registro 2 son los operadores. Debido a esto hemos añadido a nuestra ALU operaciones adicionales. Que la salida sea igual al segundo operando `op_alu = 1001` o poder realizar una resta con el segundo operando menos el primero `op_alu = 1000`.
- **Carga inmediata:** Se coge el primer operando que corresponde como constante.
- **Saltos:** Los saltos condicionales se basan en si la última operación de la ALU es 0 o no para realizar el salto.
- **Llamadas a subrutinas:** Saltan a la dirección donde se encuentra la subrutina.
- **Retorno de subrutinas:** Se extrae la dirección de retorno de la pila.
- **Retorno de subrutinas de interrupción:** Se extrae la dirección de retorno de la pila y se vuelven a habilitar las interrupciones.

5. Características añadidas a diseño final

1. Se ha aumentado el tamaño de los registros a 16 bits.
2. Se ha aumentado el tamaño de las instrucciones a 32 bits.
3. Ampliación a instrucciones aritmético-lógicas con datos inmediatos como comentamos anteriormente.
4. Se ha añadido una pila que almacena direcciones de retorno de subrutinas y de interrupción.
   - **Push:** se escribe la dirección de retorno y se incrementa el stack pointer.
   - **Pop:** saca el elemento anterior al stack pointer, lo borra y decrementa el stack pointer.
5. Se ha añadido un sistema de interrupciones para permitir a los dispositivos externos solicitar atención de la CPU.
   - **Codificador:** Este módulo que hemos creado se encarga de codificar las líneas de interrupción, asignando una prioridad a cada línea de interrupción. Estas líneas se encuentran conectadas a los botones de la FPGA, por lo que se activarán cuando presionemos nuestro botón. En este caso será con el único botón que hemos conectado.
   - **Registro_interrupcion:** Almacena la interrupción hasta que pueda realizarse, esto ocurrirá cuando acabe el flanco de reloj o cuando acabe una interrupción previa.
   - **ffi:** Determina si pueden proceder las interrupciones o no.
   - **ffd:** Guarda el estado del ffi y lo reenvía al ffi y luego también a la unidad de control.
   - **dir_sal_in:** Se encarga de darnos una dirección de salto a la que saltar para atender la instrucción requerida, esto sucede cuando nos llega la señal de nuestro botón de la señal ir_attended.
   - Hemos añadido señales nuevas para el control de las interrupciones:

     pop_inm: Se encarga de indicar cuando se tiene que habilitar de nuevo las interrupciones.

     push_inm: Se encarga de indicar cuando una interrupción tiene que ser atendida.

     wcalli: Se utiliza para saber que dirección de salto se debe seleccionar.
   - Se han añadido 3 funciones adicionales para el flag de cero de nuestra ALU, esto lo hemos tenido que hacer para poder mantener dicho flag cuando tenemos que retornar de la subrutina.

     ffzi: Este biestable tipo d guarda el estado del flag de cero.

     zero_guard: Este mux se encarga de restaurar el flag cuando tengamos que retornar de la interrupción.
6. Se ha añadido un sistema de buses exteriores.
   - **addr:** Bus de direcciones de 16 bits. Se ha conectado al registro 1 donde se indica que led se enciende.
   - **data:** Bus de datos de 16 bits. Se ha conectado al registro 2 donde se indica el valor a escribir.

- **mapeo_dis:** Este módulo parte de la dirección del bus de direcciones, donde se obtiene el led que se pretende escribir, esto se hace mediante la señal **"escritura"** que habilita la escritura en mapeo_dis.
- **interfaz_led:** Este módulo tiene una salida hacia un led de la placa, donde se produce un mapeo del valor de bus de datos a un valor que puede ser alto o bajo.

6. Programa

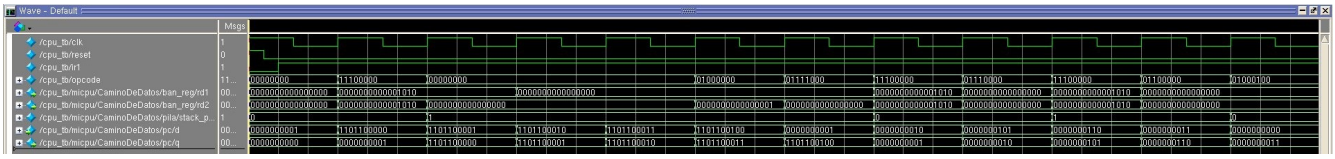
Hemos realizado un programa donde cargamos un valor se realiza una resta, luego se hace un salto a una subrutina donde se realiza otra resta. Además, también tenemos el código donde se muestra el uso de las interrupciones para encender el led que tenemos.

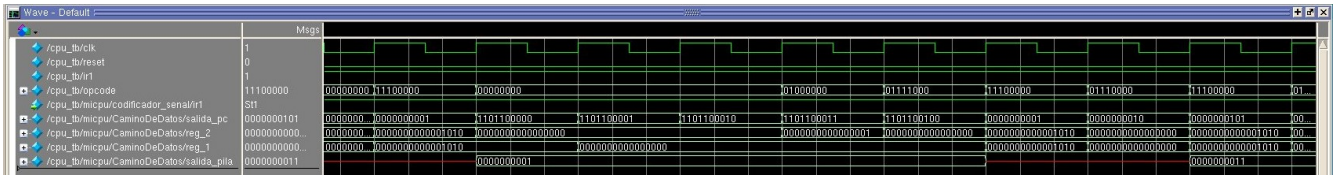
```
00000000000000000000000000011111  // Carga inmediata de un 1 en el registro 15
00000000000000000000000000001110  // Carga inmediata de un 0 en el registro 14
00000000000000000000000000001101  // Carga inmediata de dir led0 en el registro 13
01000000111100000000000011010000  // Ponemos en alto el led 0
01111000000000000000000000000000  // Retorno de subrutina de interrupcion
```

```
1  00000000000000000000000010100001  // Carga inmediata de un 10 en el registro 1
2  11100000000100000000000000010010  // Resta de constante inmediata 1 con registro 1 y guardado en registro 2
3  01110000000000010100000000000000  // Salto a subrutina
4  01000100000000000000000000000000  // Salto incondicional a la primera instruccion
5  00000000000000000000000000000000
6  11100000000100000000000000100010 // Resta de constante 2 con registro 1 y guardado en resultado en registro 2
7  01100000000000000000000000000000 // Se regresa a la direccion de retorno
```

7. Demostración

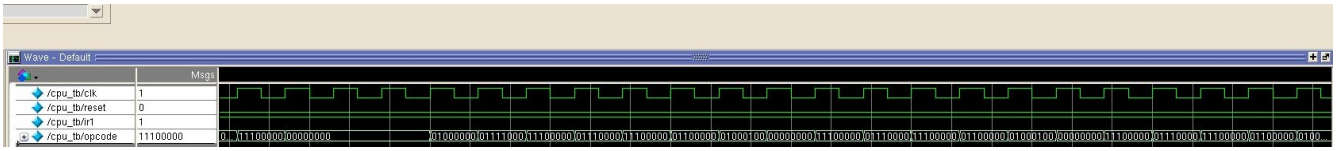
- Simulación RTL, hemos realizado dos simulaciones para verificar el programa:





Aquí podemos observar cómo se producen las cargas inmediatas, resta, y el salto a la subrutina que hemos creado. Además, podemos observar el funcionamiento de nuestra pila.

- La simulación Slow Model para verificar el comportamiento de nuestro programa, debido a las limitaciones que nos daba el Slow Model, debido a que no nos dejaba manejar más señales, en comparación con el RTL lo comprobaremos en base al opcode:



- Resultados que hemos obtenido con el Slow Model:

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	54.65 MHz	54.65 MHz	clk	

Slow Model Setup Summary

	Clock	Slack	End Point TNS
1	clk	1.702	0.000

Enlace al video

`https://www.youtube.com/shorts/RaqYaGooHBU`