Cátedra de Arquitectura de Computadoras

Trabajo Práctico Nº III - BIP I

Integrantes:

López Gastón (gopezlaston@gmail.com)
Kleiner Matías (kleiner.matias@gmail.com)

Fecha de entrega:

17 de octubre de 2018

Introducción

Para el Trabajo Práctico N° 3, se realizó la implementación en Verilog de un procesador **BIP I** (basic instruction-set processor I). Al mismo se lo conectó a un módulo UART, el cual había sido diseñado e implementado en el Trabajo Práctico N° 2. El sistema se completa con una PC, a partir de la cual se envían 2 comandos sucesivos. El primero resetea por software el sistema y el segundo inicia el procesador (señal init). Dicho procesador ejecuta las instrucciones que se encuentran en la memoria de programa. Al ejecutar la instrucción HALT, el valor del acumulador (ACC) y del contador de ciclos (CC) en ese momento deben ser enviados hacia la PC vía UART.

Para la implementación se usará una FPGA, la cual es un dispositivo programable que contiene bloques de lógica que, al interconectarlos, genera una funcionalidad. La mencionada interconexión y funcionalidad se configuran mediante un lenguaje de descripción de hardware (Verilog).

Resumen de la consigna

Características del procesador BIP:

- Procesador monociclo.
- Todas las instrucciones en el ISA están basadas en un único formato.
- No posee una arquitectura load/store debido a que instrucciones lógicas y aritméticas pueden acceder a la memoria de datos.
- La versión I no soporta saltos con y sin condición. La versión II sí los soporta.
- Arquitectura con acumulador.
- Datos de 16 bits.
- Instrucciones de 16 bits.
- Únicamente dos modos de direccionamiento (inmediato y directo).
- I/O mapeada en memoria.
- Set de instrucciones reducido.

- Si bien tiene muchas características de los procesadores RISC no se lo considera uno de ellos por no poseer una arquitectura load/store.

En el presente trabajo se diseñará e implementará el BIP I.

Formato de la instrucción del BIP I:

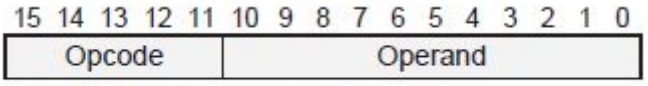

Figure 1. Instruction format

- **Opcode:** campo de 5 bits de longitud que identifica la operación a ser realizada por la instrucción.
- **Operand:** campo de 11 bits de longitud para identificar el operando de la instrucción. Éste puede representar una constante (dato inmediato) o una dirección de la memoria de datos.

Registros de la CPU del BIP I:

- **PC:** el contador de programa. Almacena la dirección de la instrucción actual.
- **ACC:** el acumulador. Trabaja como un operando implícito en muchas instrucciones.

Organización de la memoria:

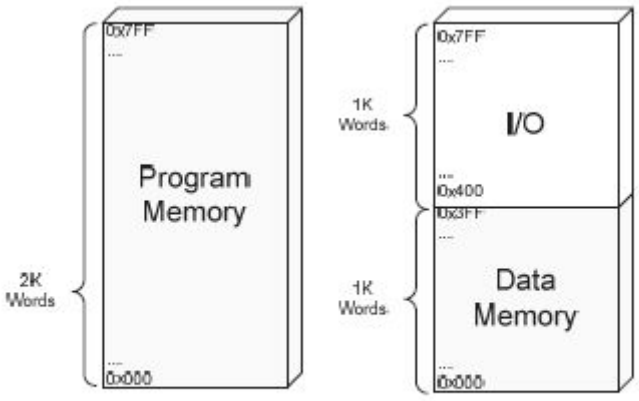

Figure 2. Addressing spaces

- **Memoria de programa:** 2k.
- **Memoria de datos:** 1 k.
- **Memoria I/O:** en este trabajo práctico no fue necesaria.

Set de instrucciones del BIP I:

TABLE I. INSTRUCTION SET

Operation	Opcode	Instruction	Data Memory (DM) and Accumulator (ACC) Updating	Program Counter (PC) updating	Affected Flags	BIP Model
Halt	00000	`HLT`		`PC ← PC`		I, II
Store Variable	00001	`STO operand`	`DM[operand] ← ACC`	`PC ← PC + 1`		I, II
Load Variable	00010	`LD operand`	`ACC ← DM[operand]`	`PC ← PC + 1`		I, II
Load Immediate	00011	`LDI operand`	`ACC ← operand`	`PC ← PC + 1`		I, II
Add Variable	00100	`ADD operand`	`ACC ← ACC + DM[operand]`	`PC ← PC + 1`	Z,N	I, II
Add Immediate	00101	`ADDI operand`	`ACC ← ACC + DM`	`PC ← PC + 1`	Z,N	I, II
Subtract Variable	00110	`SUB operand`	`ACC ← ACC - DM[operand]`	`PC ← PC + 1`	Z,N	I, II
Subtract Immediate	00111	`SUBI operand`	`ACC ← ACC - operand`	`PC ← PC + 1`	Z,N	I, II

Estructuración del BIP I:

El BIP I se encuentra estructurado en 2 bloques principales:

- **Control:** este bloque busca las instrucciones de la memoria del programa, las decodifica y dirige las operaciones en el bloque Datapath. Está compuesto por el PC, un sumador de 11 bits y un decodificador de instrucciones combinacional.
- **Datapath:** este bloque procesa los datos bajo la supervisión del bloque de Control. Incluye el registro ACC, 2 multiplexores, un

bloque de extensión de signo de 11 bits a 16 bits y una unidad aritmética que consiste en un sumador/restador de 16 bits.

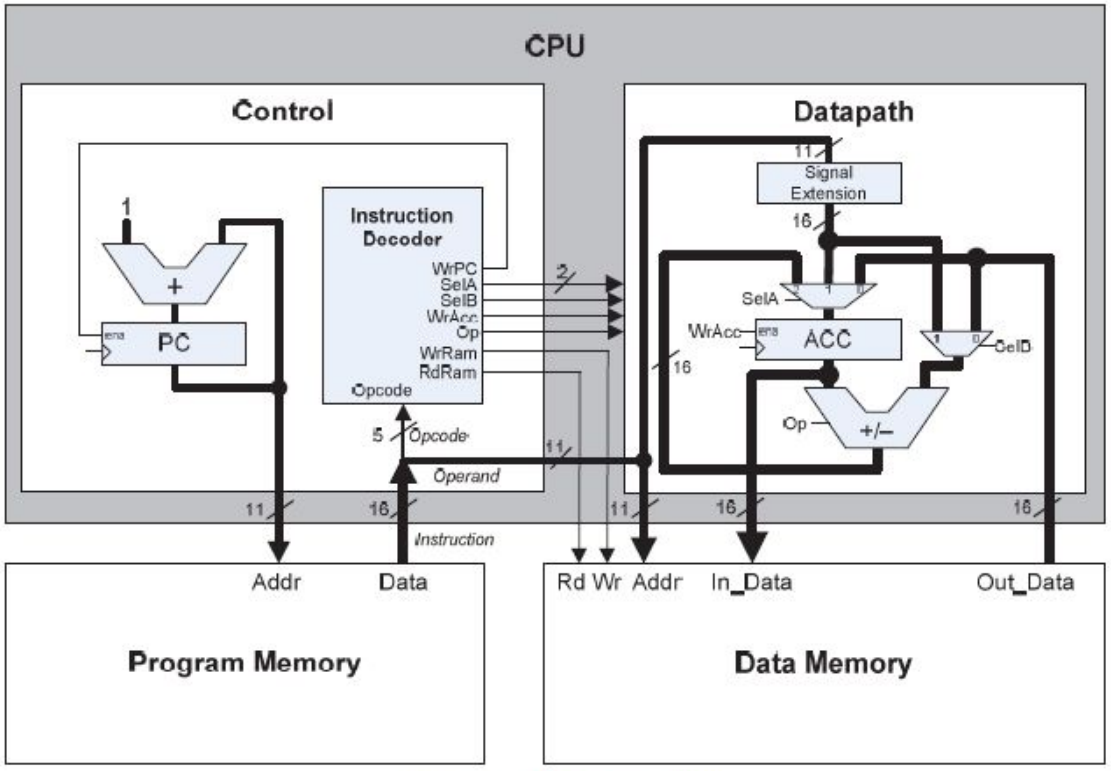

Figure 3. BIP I organization

Herramientas y elementos utilizados

Entre los dispositivos disponibles para la implementación del BIP I integrado con la UART, se seleccionó la placa de desarrollo ARTY, que puede verse en la siguiente figura.

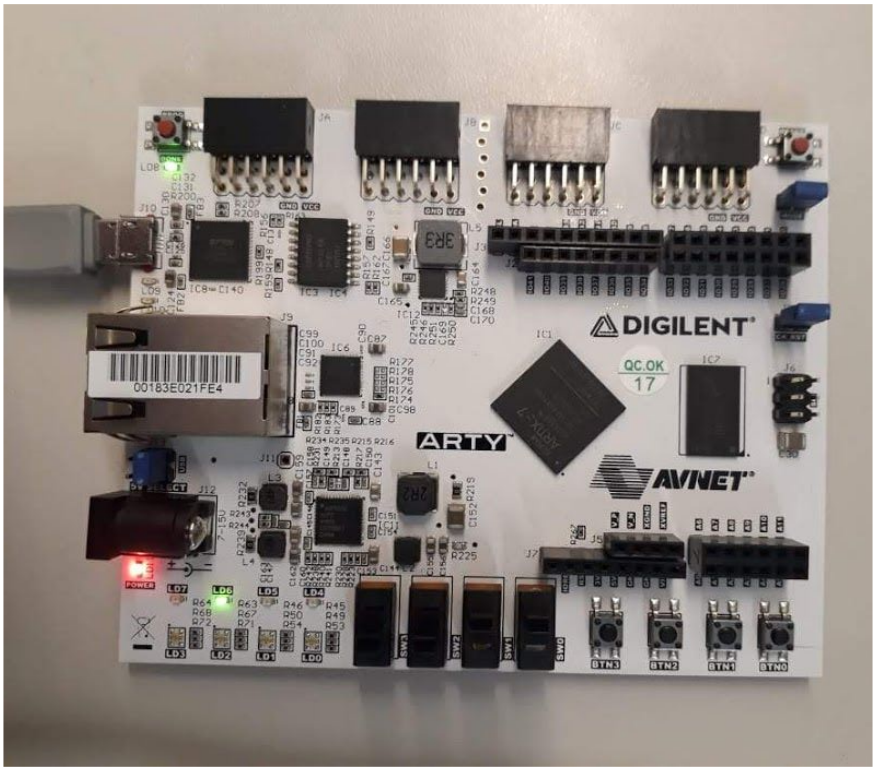

Figura 1 - FPGA Utilizada para el desarrollo del proyecto.

La principal razón de la selección fue que esta placa es una de las que soportan la interfaz de desarrollo Vivado, sobre la cual poseemos algo de experiencia previa. Cabe destacar que la FPGA que contiene la placa es una Artix-7 de Xilinx.

Diseño - Módulo TOP.

Se definió un módulo Top, que contiene la instanciación de los siguientes módulos:

- Tx.
- Rx.
- Baud Rate Generator.
- Interface Circuit.
- Control. (Incluye dos módulos).

- Datapath.
- Memoria de programa.
- Memoria de datos.
- Contador de ciclos.

Las entradas y salidas de este módulo son:

- input i_clock: clock.
- input i_reset: reset.
- input uart_txd_in: transmisor de PC.
- output uart_rxd_out: receptor de PC.

El baud rate (parametrizable) utilizado en este trabajo práctico es de 9600 bps, con una frecuencia del clock de 100 MHz. La cantidad de bits de stop es 2 y la cantidad de bits de datos es 8, generando con el bit de start una trama con longitud igual a 11 bits.

Esquema general del top:

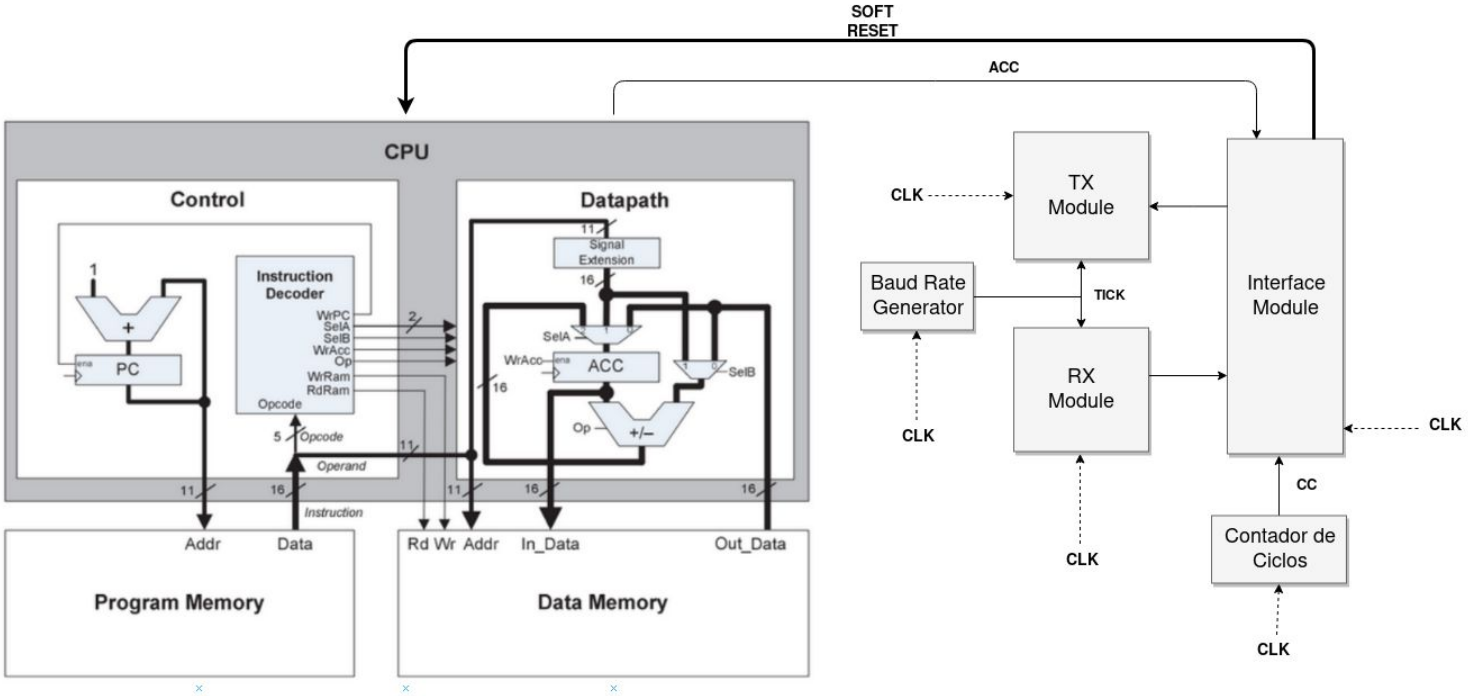

Figura 2 - Esquema general del top.

Notas sobre el diseño:

Es necesario destacar que a la hora de diseñar un procesador monociclo, se nos apareció el problema de que debíamos hacer los siguientes pasos en un solo ciclo de clock:

1. Actualizar el contador de programa.
2. Buscar la siguiente instrucción a ejecutar.
3. Decodificar la instrucción.
4. Trabajar con la memoria de datos o realizar la operación aritmética.
5. Escribir en el registro ACC.

Todos esos pasos se deben realizar en un mismo pulso de clock. Es por ello, que se deben utilizar los flancos ascendentes y descendentes de dicho pulso. Teniendo en cuenta que el paso 3 es combinacional y que la realización de la operación aritmética también, nos queda lo siguiente:

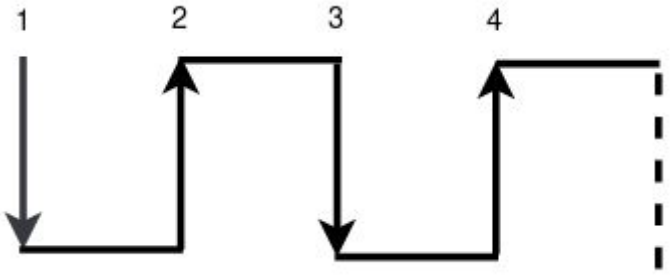

Figura 3 - Utilización de los flancos del pulso de clock.

Explicación de las etiquetas en la Figura 3:

1: En el flanco descendente, se actualiza el valor del contador de programa.

2: En el flanco ascendente, se obtiene la instrucción apuntada por el contador de programa.

3: En el flanco descendente se trabaja con la memoria de datos (lectura o escritura).

4: En el flanco ascendente, se actualiza el registro ACC, por ejemplo, en el caso de una instrucción LD.

Si entramos más en detalle, una instrucción LD, ocuparía 4 flancos de reloj. Sin embargo, el procesador monociclo se obtiene considerando lo siguiente:

- En el flanco 3, se está actualizando el contador de programa para la instrucción siguiente.
- En el flanco 4, mientras que en el módulo datapath se está actualizando el valor del ACC, la memoria de programa está devolviendo la instrucción siguiente.

Diseño - Módulo Baud Rate Generator.

Este módulo genera ticks con los que alimenta a los módulos Tx y Rx para coordinar las señales recibidas o transmitidas. La generación de ticks es 16 veces por baud rate. Consiste en un contador, que cuando alcanza un límite, genera un tick. Para calcular dicho límite la fórmula es la siguiente:

$$Límite = \frac{Clock}{Baud\ Rate * 16}$$

Diseño - Módulo RX.

Consiste en una máquina de estados. La representación de estados utilizada es la one-hot. Observando la Figura 4, se tiene que para pasar del estado de Espera al estado de Start se requiere la llegada del bit de start (bit en nivel bajo). Cuando se cuentan 8 ticks se está en la mitad de dicho bit de start que se recibió, por lo que se cambia al estado Read. En este estado, se cuentan los 8 bits de datos (cada 16 ticks se toma el valor recibido debido a

que se está en el medio del bit transmitido desde la PC). Luego de contar esos 8 bits (valor parametrizable) se pasa al estado de Stop. Allí se espera una cantidad de bits de stop (en nivel alto) igual al valor definido en la parametrización del módulo. Cuando llegan dichos bits de stop, se coloca el bit Rx Done en alto y en la salida del módulo Rx también se inserta el dato recibido (los 8 bits de dato). Luego se pasa al estado de Espera. En caso de que se necesiten 2 bits de stop, si el primero que llega es un 1 y el segundo un cero, se desincroniza todo el sistema debido a que faltan 8 ticks para pasar el segundo bit de stop. Esta tarea de dejar pasar dicha cantidad de ticks se realiza en el estado de Error.

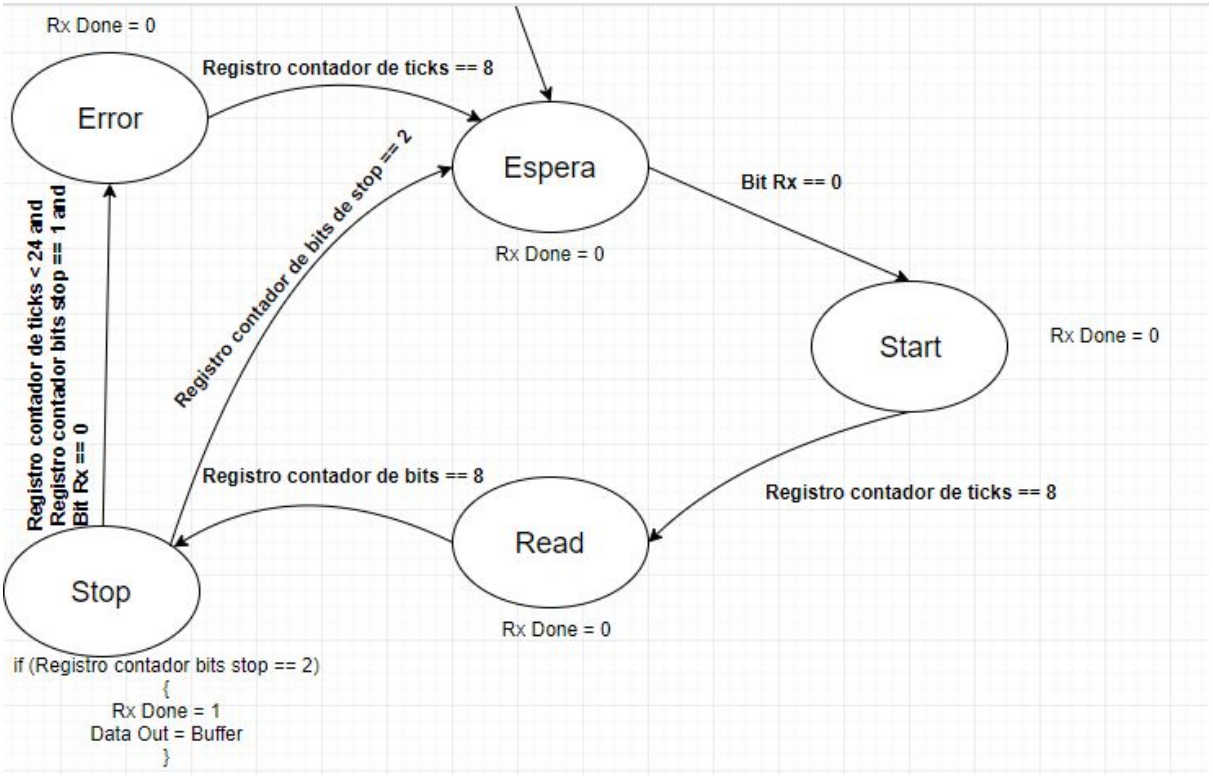

Figura 4 - Máquina de Estados del módulo Rx.

Diseño - Módulo TX.

Consiste en una máquina de estados similar a la anterior. La representación de estados utilizada es la one-hot. Observando la Figura 5, se tiene que para pasar del estado de Espera al estado de Start se requiere un nivel en alto de la señal Tx Start. Luego el pasaje entre los demás estados se

debe a la finalización en la generación de los distintos bits de la trama. Mientras se están transmitiendo dichos bits la salida Tx Done se encuentra en un nivel bajo para que el módulo Interface Circuit no envíe otro dato a transmitir. En los estados Read y Stop las salidas están sujetas a condiciones para prevenir errores, colocándose en el diagrama anterior aquellas más representativas.

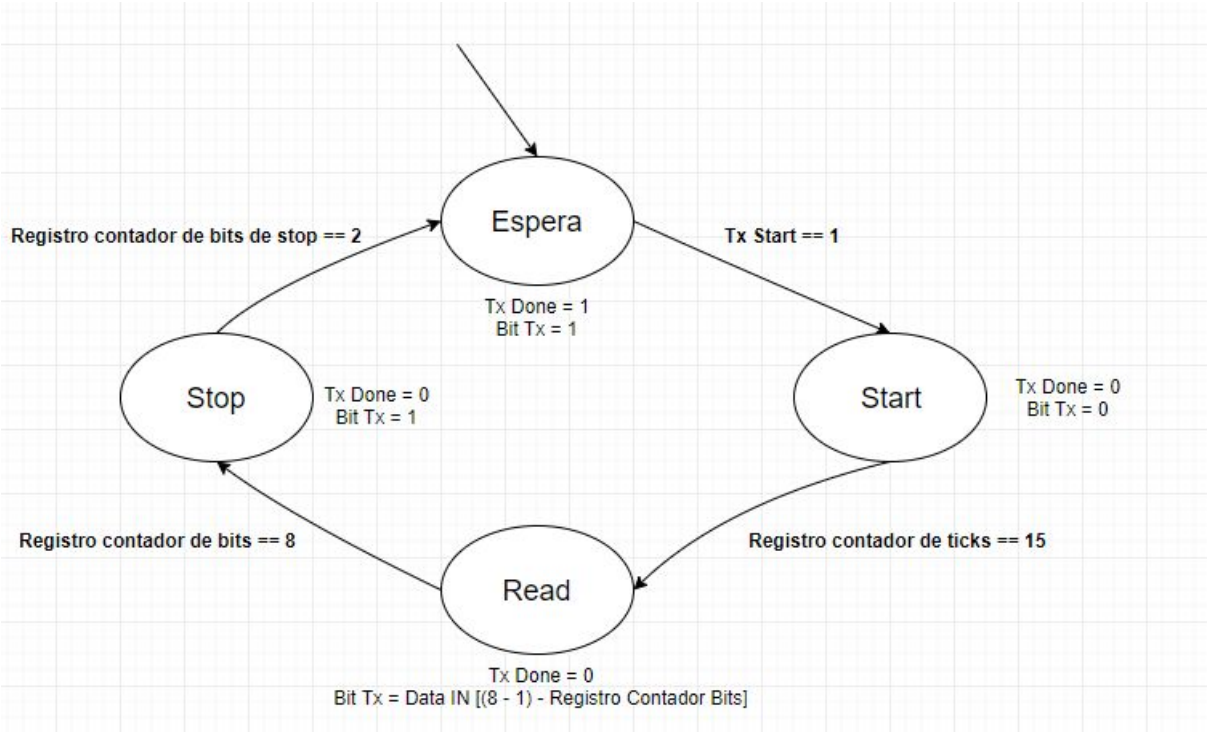

Figura 5 - Máquina de Estados del módulo Tx.

Diseño - Módulo Interface Circuit.

Consiste también en una máquina de estados. Cumple con las siguientes funciones:

- Permite la comunicación entre el procesador BIP I y los módulos Tx y Rx que conforman la UART.
- Recibe los comandos del módulo Rx, verifica que sean correctos y en base a dichos valores, modifica el valor del soft reset. El

reseteo del procesador **BIP I** es por software y por nivel. Un nivel bajo del soft reset mantiene reseteado dicho procesador. Cuando llega desde la PC el comando de init, se pone en uno el soft reset (señal de init), lo cual inicia el funcionamiento del procesador.

- Detecta la instrucción HALT y almacena en registros propios los valores del CC (Contador de Ciclos) y del ACC (acumulador), los cuales son enviados hacia el transmisor de a 8 bits (se transmiten 4 bytes).

La máquina de estados del Interface Circuit utiliza las flancos ascendentes de las señales Rx Done (salida del módulo Rx que indica que se recibió en forma completa un dato) y Tx Done (señal de salida del módulo Tx que indica que se completó la transmisión de un dato) para efectuar los cambios de estados, los cuales se representan mediante lógica one-hot. Cuando diseñamos este módulo no tuvimos en cuenta de incluir como estados los correspondientes a la etapa de recepción de comandos enviados desde la PC. Sin embargo, quedaron implícitos en la implementación, por lo que se los incluye en la máquina de estados que se muestra a continuación

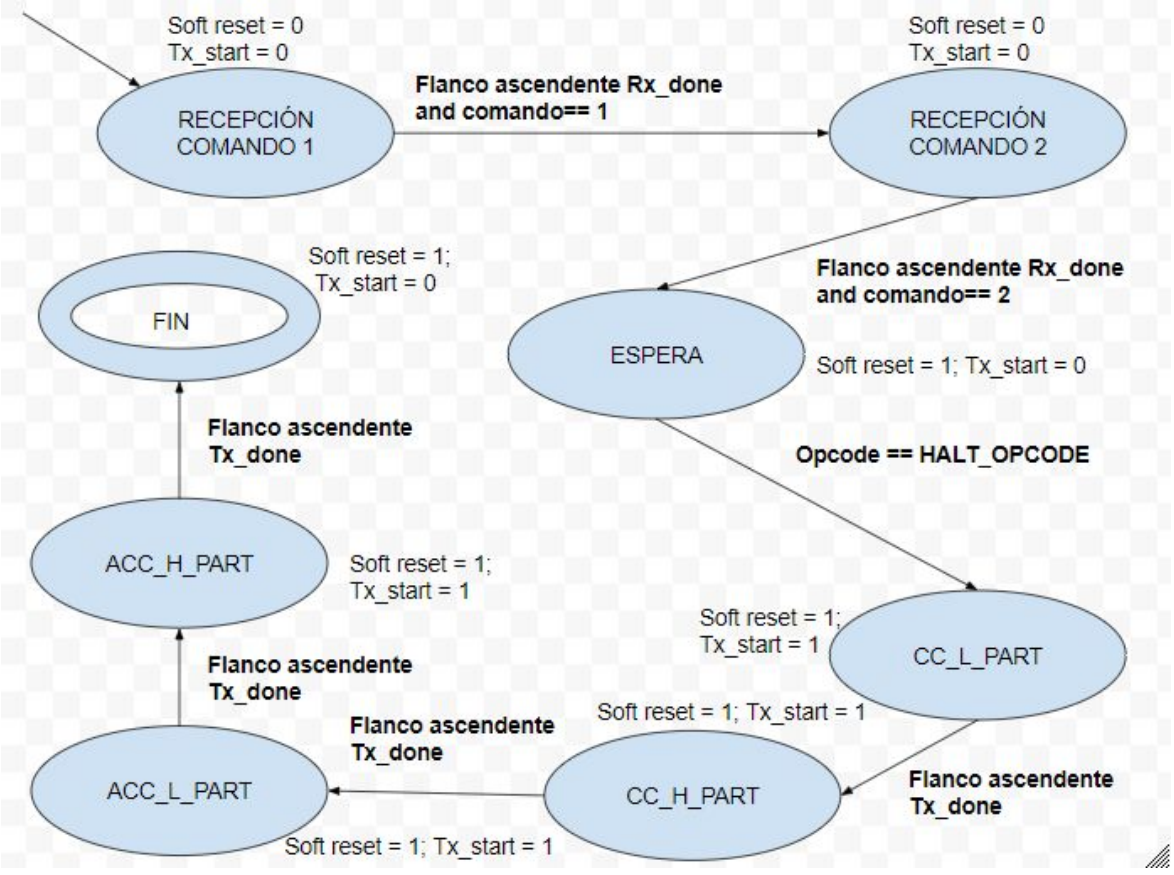

Figura 6 - Máquina de Estados del módulo Interface Circuit.

Significados a aclarar:

- CC_L_PART: 8 bits menos significativos del contador de ciclos.
- CC_H_PART: 8 bits más significativos del contador de ciclos.
- Tx_start: señal de salida del módulo Interface Circuit para indicarle al Tx que comience la transmisión.

Diseño - Módulo Contador de Ciclos.

El módulo contador de ciclos consiste en un contador que cuenta los pulsos de clock, desde que llega la señal de init desde la PC hasta que el procesador lee la instrucción HALT.

Diseño - Módulo Control.

El módulo de control cumple la función de instanciar y conectar dos módulos:

- Address Calculator.
- Instruction decoder.

Diseño - Módulo Address Calculator.

El módulo Address Calculator está compuesto por el contador de programa y el sumador de 11 bits que se utiliza para obtener la siguiente dirección de instrucción. Cabe destacar que el contador de programa no cuenta a menos que llegue la señal de init desde la PC.

Diseño - Módulo Instruction Decoder.

El módulo Instruction Decoder, recibe los 5 bits del Opcode de la instrucción leída de la memoria de programa y mediante un circuito combinacional configura los valores de las señales de control del módulo Datapath, de la memoria de datos y del módulo Address Calculator.

Diseño - Módulo Datapath.

El módulo Datapath posee el registro ACC y toda la lógica para poder realizar la operación especificada por la instrucción. Dicha lógica consiste en:

- Un sumador/restador de 16 bits que permite realizar las operaciones de adición y resta solicitadas por las instrucciones ADD, ADDI, SUB y SUBI.
- Un módulo de extensión de signo de 11 bits a 16 bits. Esto es porque existe una diferencia de bits entre un operando obtenido de la memoria, el cual es de 16 bits, y uno presente en la instrucción, el cual posee 11 bits.
- 2 multiplexores, los cuales son controlados por señales que provienen del módulo Instruction Decoder. Dichos mux controlan cuáles serán los operandos aritméticos o cuál será el valor del ACC en función de la operación a realizar. Por ejemplo, la instrucción LD 10, carga en el acumulador un operando que se obtiene a partir de la dirección 10 de la memoria de datos, mientras que la instrucción LDI 10, carga el valor 10 en el ACC. Estas diferencias se solucionan con la lógica de los mux.

Diseño - Módulo Memoria de programa.

La memoria de programa se implementó en base al template Xilinx Single Port No Change RAM. Cabe destacar que dicha memoria se diseñó sin reset. Algunas consideraciones de diseño:

- Tamaño de la memoria: 2048 x 16 bits.
- Performance de la memoria: HIGH_PERFORMANCE.
- Configuración de los instrucciones: la memoria se carga con instrucciones en código máquina que se encuentran en el archivo *init_ram_file.txt.* En caso de no existir el archivo, se carga con todos ceros.

Diseño - Módulo Memoria de Datos.

La memoria de datos se implementó bajo el mismo template con la cual se desarrolló la memoria de programa. Cabe destacar que dicha memoria se diseñó sin reset. Algunas consideraciones de diseño:

- Tamaño de la memoria: 1024 x 16 bits.
- Performance de la memoria: LOW_LATENCY. (Sin registro de salida, necesario por velocidad debido a que no puedo utilizar otro flanco del clock).
- Configuración de los datos: en cada posición de memoria se guarda como dato el valor de dicha posición. Es decir, en la posición o dirección 10 de memoria, se guarda el valor 10 (16 bits).

Simulaciones de los módulos.

Se efectuaron test benches para todos los módulos y para el top.

- **Top:** se comprueba el reset y que el sistema devuelva y transmita los resultados correctos (valor del ACC y valor del CC) luego de que se envíen los comandos correspondientes y el procesador ejecute las instrucciones cargadas en la memoria de programa.

- **Baud Rate Generator:** se comprueba la generación correcta de los ticks.
- **Rx:** se envían bits y se comprueba el pasaje entre los estados del módulo y las salidas en cada uno de dichos estados. Se envían tramas correctas e incorrectas.
- **Tx:** se ingresa un valor a transmitir y se eleva en alto el bit de Tx Start. Se comprueba el pasaje entre los estados del módulo y las salidas en cada uno de dichos estados.
- **Interface Circuit:** se comprueba el pasaje entre los estados del módulo y las salidas en cada uno de dichos estados. Se comprueba que se estén cargando correctamente los valores en los distintos registros. Se prestó especial atención a la evolución del soft reset a medida que arriban los comandos desde la PC.
- **Contador de ciclos:** se testeó que el contador verdaderamente cuente con cada pulso de clock a partir de la señal de init y que finalice dicho conteo cuando el procesador lea una instrucción HALT.
- **Memoria de programa y memoria de datos:** se comprobó la carga de datos, que sean síncronas con el clock y que el flanco en el cual devuelven el dato solicitado sea el correcto.
- **Address Calculator:** se testéo la evolución del contador de programa a partir de la llegada de una señal de init (soft reset en alto).
- **Instruction Decoder:** se comprobó que las salidas sean las correctas cuando ingresa al módulo un Opcode determinado.
- **Datapath:** se testeó que el módulo se comporte adecuadamente frente a señales de control determinadas que representan una operación o instrucción. Por ejemplo, si dichas señales de control corresponden a una instrucción LDI se espera que el registro ACC se cargue con el valor inmediato del operando presente en la instrucción.

Implementación en FPGA.

Para la implementación, se instanció el módulo top, sobre el cual se declararon entradas y salidas que se mapean con pines de la FPGA a utilizar en un archivo denominado constraints. El mismo consiste en un archivo con extensión *.xdc*.

Traducción de assembler a código máquina.

Es importante mencionar que la memoria de programa se carga con instrucciones binarias que se encuentran en un archivo de nombre *init_ram_file.txt.* Para hacer más fácil la tarea de escribir dichas instrucciones binarias, se generó un script en python que traduce el archivo *assembler_BIP_I.txt*, que se encuentra escrito en assembler, al archivo *init_ram_file.txt*, escrito en código máquina.

```
#A 10
#B 2
#C 5
LDI A
STO 1
LD B
STO 3
LD A
ADD B
ADDI 1
SUBI 3
SUB C
STO A
HLT
//11 ciclos y resultado en ACC igual a 5.
```

Figura 7 - Assembler utilizado para este práctico.

Hay que aclarar en la figura anterior que las líneas que arrancan con el caracter # corresponden a asignaciones de variables.

Interfaz en la PC.

Se generó una GUI en Python con la librería Tkinter que permite enviar el comando de reset y, seguidamente, el comando de init por medio del botón *Iniciar BIP I*. Cuando el procesador ejecuta la instrucción HALT y envía los valores del ACC y del CC hacia la PC, dicha GUI los muestra en pantalla.

Figura 8 - GUI efectuada en Python.

Conclusión.

Una vez terminado el práctico, a través de la herramienta Vivado se obtuvieron las especificaciones en la utilización de los recursos de la FPGA y en la potencia que consume dicho circuito instanciado. Además, en este práctico se afianzaron los conceptos en cuanto al lenguaje de descripción de hardware Verilog.

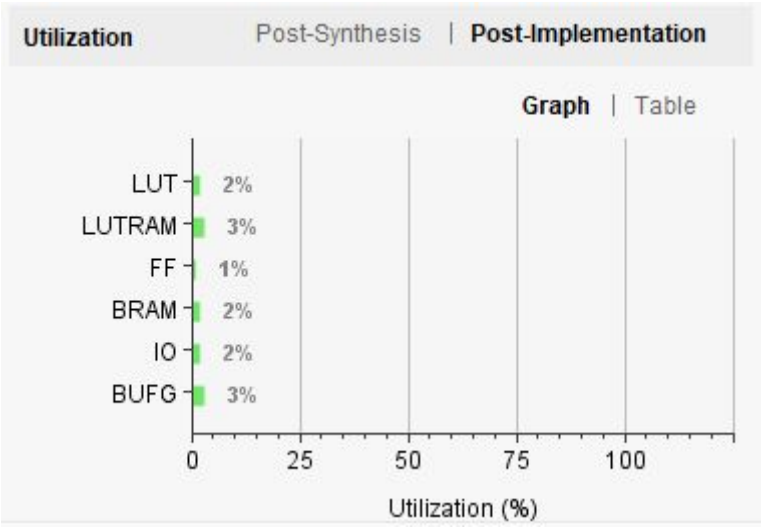

Figura 9 - Recursos utilizados para el desarrollo del proyecto.

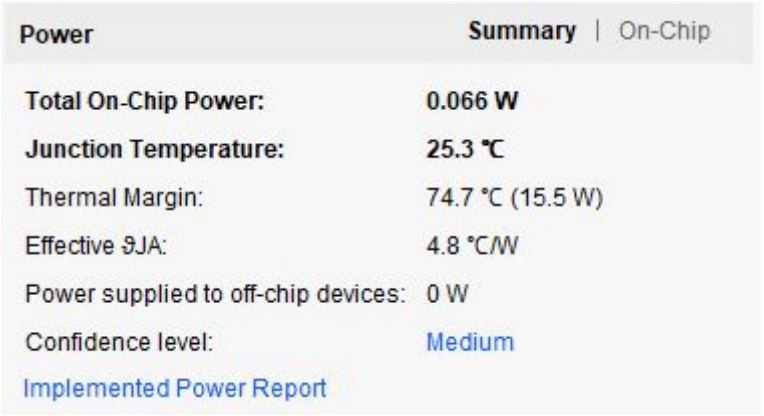

Figura 10 - Potencia.