Arquitectura de computadoras 2020

Trabajo Práctico N° 3 - BIP

Docentes:
Pereyra Martin (martinmiguelpereyra@gmail.com)
Pinzani Paulo (ppinzani89@gmail.com)
Rodriguez Santiago (srodriguez@unc.edu.ar)

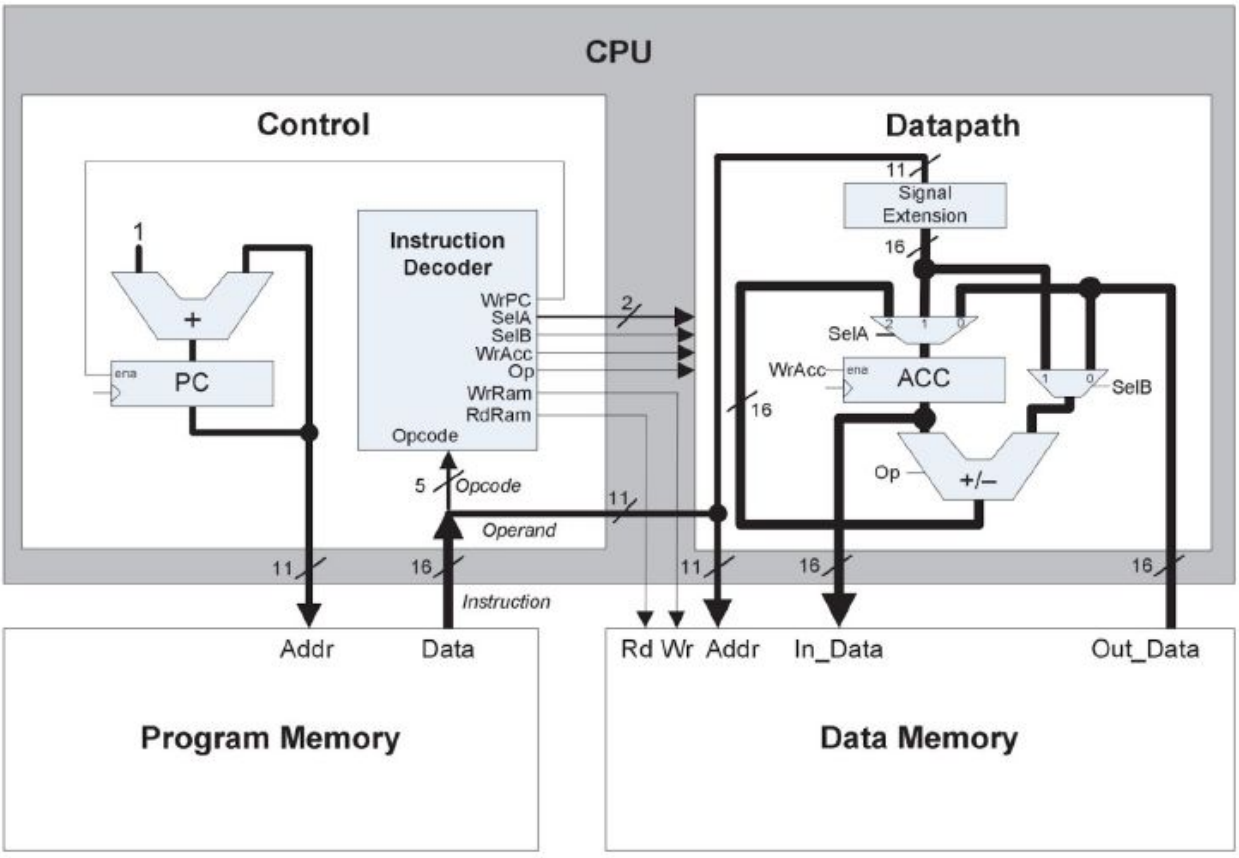
Alumnos:
Bosack Facundo Matias (facubosack@mi.unc.edu.ar)
Pichetti Augusto (augusto.pichetti@mi.unc.edu.ar)

Diciembre, 2020

Objetivos

- Describir en detalle la implementación de un procesador simplificado (BIP)
- Conectar el procesador con el módulo UART del trabajo anterior para la transmisión del valor que va a tener el acumulador.

Arquitectura del sistema a implementar



El BIT va a tener una unidad de Control, una unidad de Datapath, una memoria de programa y una memoria de datos. En la memoria de programa se va a codificar según el set de instrucciones del procesador un programa corto de 5 instrucciones.

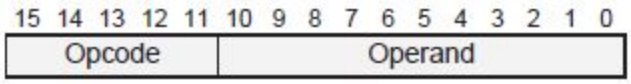
Cuando llegue la instrucción de Halt, se va a enviar por la Uart el resultado de lo que está en el registro ACC que es el acumulador. El procesador deberá ser monociclo ,es decir se tomará un ciclo de reloj para la ejecución de cualquier instrucción.

Set de instrucciones y formato de la instrucción

El Set de instrucciones que soporta el procesador es el siguiente:

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

El formato de cada instrucción es el siguiente:



Donde el **Opcode** corresponde a un campo de 5 bits de longitud que identifica la operación a ser realizada por la instrucción y el campo **Operand** es de 11 bits e identifica el operando de la instrucción. Este puede representar un dato inmediato (constante), una dirección en la memoria de datos o una dirección en la memoria de programa.

Desarrollo de módulos

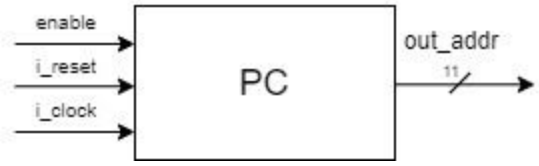
Para el desarrollo de este proyecto se implementaron los siguientes módulos:

- **PC:** Este módulo funciona como un sumador de 11 bits que va aumentando en uno en uno siempre y cuando esté habilitado su enable para ir obteniendo la siguiente instrucción de la memoria de programa.
- **Instruction Decoder:** Este módulo es el encargado de decodificar las instrucciones que llegan de la memoria del programa.
- **Signal Extension:** Módulo encargado de hacer la extensión de signo. Ingresa un operando de 11 bit y se lo extiende según el signo a 16 bits.
- **SelA:** Es un multiplexor que selecciona según corresponda, el dato salido del signal extension, la salida de la ALU o el dato de la memoria de datos. La salida que corresponda va a ser la entrada al acumulador
- **SelB**: Es un multiplexor encargado de seleccionar según corresponda, el dato salido del signal extension o el dato de la memoria de datos. La salida correspondiente va a ser la entrada del dato B de la ALU.

- **ACC**: Este módulo se encarga de ir acumulando en un registro los resultados de las operaciones que vaya realizando el procesador. La salida del mismo va a ser la entrada al dato A de la ALU.
- **ALU**: Es una Unidad Aritmética Lógica que va a realizar la operación que corresponda entre el dato A y el dato B.
- **Program Memory**: Va a ser la memoria de programa donde el PC va a ir seleccionando las instrucciones cargadas en la misma.
- **Data Memory**: Funciona como un registro donde se van a ir guardando los datos en algún espacio de la misma, es decir, en alguna dirección .
- **Tx:** Módulo utilizado para realizar la transmisión.
- **Baud_rate_gen:** Módulo que de acuerdo a la frecuencia del clock y el baud rate, genera a su salida una sucesión de ticks que van a ser utilizados por el transmisor y receptor.
- **Control**: Módulo utilizado para las interconexiones entre PC y el instruction decoder.
- **Datapath**: Módulo utilizado para las interconexiones entre Signal extension, SelA, SelB, ACC y ALU.
- **TOP**: Módulo utilizado para las interconexiones entre Control, Datapath, las memorias y el Tx.

Módulo PC

Diagrama de bloques



Lógica de implementación

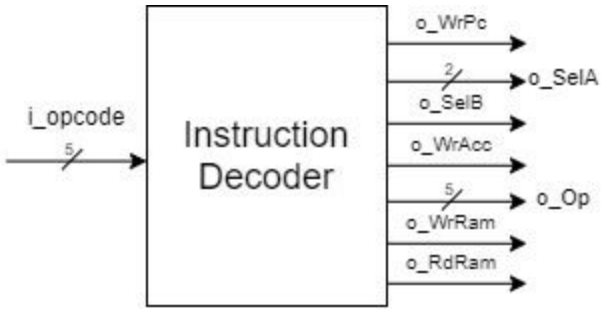
Para la implementación de este módulo, se creó un registro addr de 11 bits que si la señal de entrada enable está en alto va a incrementarse en 1 y de lo contrario no. Luego, se asigna a la salida del módulo el valor del registro addr para indicarle a la memoria de programa cuál instrucción va a ser la utilizada.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/PC.v

Módulo Instruction Decoder

Diagrama de bloques



Lógica de implementación

Este modulo tiene 1 entrada y 7 salidas. La entrada corresponde al código de operación que va a realizar el procesador y las salidas son distintas señales que se van a poner en alto o no según cual sea la operación. Por lo tanto, si la operación corresponde a un store variable, se van a poner en alto las salidas correspondientes para que esa operación se pueda realizar con éxito. Para ello se crea un case que según la operación va a realizar una u otra acción.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Instruction%20Decoder.v

Módulo Signal Extension

Diagrama de bloques

Lógica de implementación

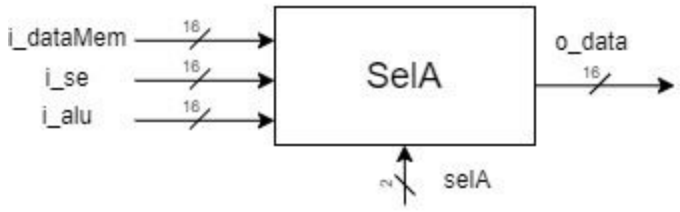
Este módulo tiene una entrada y una salida, cuando llega el dato de 11 bits, el signal extension a través de un always combinacional, concatena al valor ingresado con 5 ceros para convertirlo y extender el valor a 16 bits.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Signal_Extension.v

Módulo SelA

Diagrama de bloques



Lógica de implementación

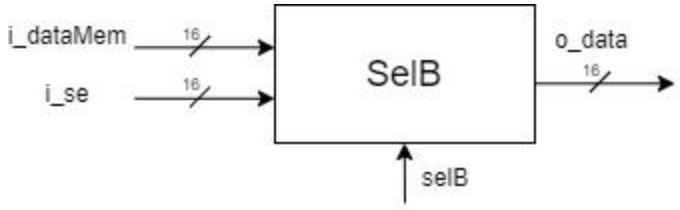
La implementación de este módulo se hizo a través de un case en un always combinacional. Se tiene una entrada que es selA y según sea 00, 01 ó 10, a la salida o_data del modulo SelA se le va a asignar la entrada del dato de la memoria de datos, la entrada del signal extensión o la entrada que viene de la salida de la ALU respectivamente. Este módulo va a funcionar como un multiplexor.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Sel%20A.v

Módulo SelB

Diagrama de bloques



Lógica de implementación

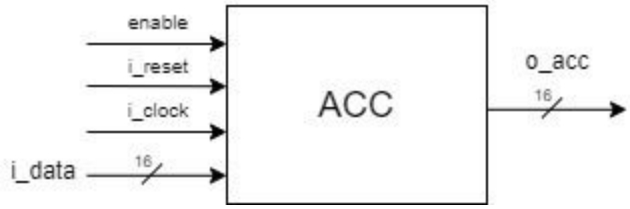
Al igual que SelA está formado por un case dentro de un always combinacional, donde según la entrada selB, a la salida o_data se le va a asignar la entrada que viene del signal extension (si selB = 1) o la que viene de la memoria de datos (si selB = 0). Este módulo va a funcionar como un multiplexor.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Sel%20B.v

Módulo ACC

Diagrama de bloques



Lógica de implementación

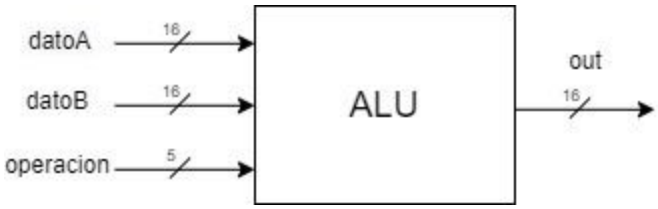
Para la implementación del módulo acumulador ACC, al ser secuencial, tiene la entrada del clock y de un reset. Luego tiene una entrada de habilitación (enable) que se encarga de habilitar o no el acumulador. Por lo tanto, el módulo está implementado con un always secuencial que segun cada flanco de subida del clock, se va a preguntar en primer lugar si el reset esta en alto, de ser asi, a la salida del registro del acumulador se le va a asignar un 0. En caso contrario, se vera si esta habilitada la señal de enable, de ser así, se le va a asignar a la salida del acumulador el valor de la entrada. Cabe destacar que la entrada del acumulador va a ser la salida del multiplexor SelA y la salida del acumulador va a ser la entrada del dato A de la ALU.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/ACC.v

Módulo ALU

Diagrama de bloques



Lógica de implementación

Módulo utilizado también en el trabajo práctico previo, utiliza una lógica combinacional implementada mediante un bloque always y una sentencia switch/case para el código de operación del módulo de la ALU. Se define el parámetro N_BITS para parametrizar el bus de datos. Las entradas que se definen son datoA, datoB y operación al igual que como se ve en el diagrama y se configura una salida que corresponde al resultado de datoA con datoB según el valor que tenga la entrada de operación.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/alu.v

Program Memory

Diagrama de bloques

Lógica de implementación

Módulo utilizado para la memoria de programa del procesador, contiene dos entradas, una para el clock y otra para especificar direcciones dentro del módulo, y una salida donde se pondrá el contenido del espacio de memoria referenciado. Esta memoria maneja bloques de 16 bits, organizados en un arreglo de registros, y se puede acceder a estos a través de una entrada de direcciones de 11 bits.

Este módulo utiliza una lógica secuencial, síncrona con el flanco positivo del clock, esto se realizó mediante un bloque always en el cual pondrá a la salida del módulo el contenido del bloque que se especifique con la entrada de direcciones.

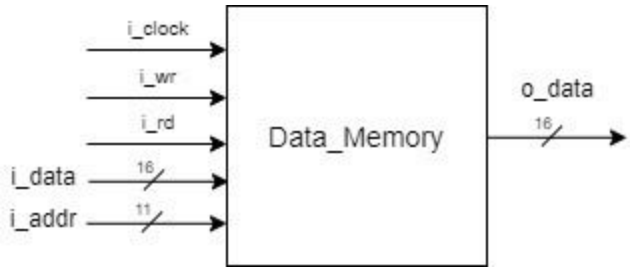
Además, utiliza un bloque generate para generar las instrucciones que estarán en esta memoria. El mismo utiliza la función readmem para leer desde un archivo .mem las estas instrucciones y colocarlas secuencialmente en los bloques de memoria. En el caso que no se contase con este archivo, la memoria iniciaría todos sus bloques con ceros.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Program%20Memory.v

Data Memory

Diagrama de bloques



Lógica de implementación

Módulo utilizado para la memoria de datos del procesador, contiene 6 entradas y una salida.

- Una entrada para ingresar datos a escribir en la memoria.
- Una entrada para especificar la dirección en la cual se realizará una escritura o lectura de esta memoria.
- Una entrada para habilitar la escritura de esta memoria
- Una entrada para habilitar la lectura de esta memoria
- Una entrada clock.
- Una salida donde se colocara el contenido del bloque especificado por la entrada de direcciones cuando esté habilitada la lectura de la memoria.

Este módulo utiliza una lógica secuencial síncrona con el flanco negativo del clock, en el cual se podrá leer o escribir un determinado bloque de acuerdo al estado de las entradas.

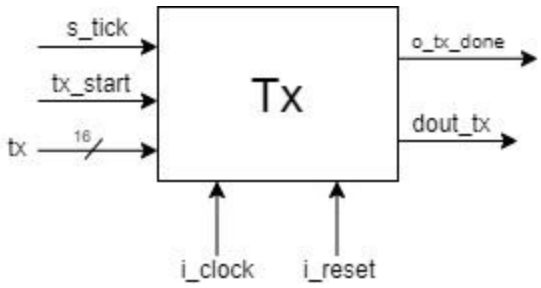
Además, cuenta con un bloque generate, que se utiliza para inicializar todos los bloques de esta memoria en cero.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Data%20Memory.v

Tx

Diagrama de bloques



Lógica de implementación

Este módulo se implementó utilizando una lógica de máquina de estado. Tiene como input el clock, el reset, el byte a transmitir, una entrada para los ticks y otra para indicar cuándo debe comenzar a transmitir. Como salida tiene una salida para la transmisión del dato en serie y otra para indicar cuando finaliza dicha transmisión.

Sus posibles estados son:

- IDLE: Estado de “espera” del transmisor. En este estado el transmisor tendrá su salida en bajo hasta que se le dé la señal para comenzar a transmitir, desde ese momento se comenzará a transmitir el bit de inicio (se pondrá un cero a la salida) el cual durará 15 ticks y se pasará al siguiente estado.
- DATA: Estado de “transmisión” del dato. En este estado se comienza con la transmisión en serie del dato de forma tal que la transmisión de cada bit durará 15 ticks, se utiliza un case acompañado de un registro interno para indicar que bit debe de transmitirse en cada momento. Al finalizar de transmitir el último bit, se pasará al siguiente estado.
- STOP: Estado de “finalización de la transmisión”. En el mismo, se procederá a enviar el bit de stop (la salida en alto), el cual durará 15 ticks y luego se pondrá como próximo estado el estado de IDLE.

Para la implementación se utilizaron dos always, uno secuencial para la transición de estado a estado y actualización de variables y salidas, y uno combinacional para la determinación del

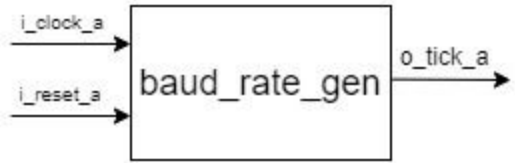
comportamiento del módulo de acuerdo a las entradas y variables internas, y la lógica del próximo estado.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Tx.v

Módulo Baud_rate_gen

Diagrama de bloques



Lógica de implementación

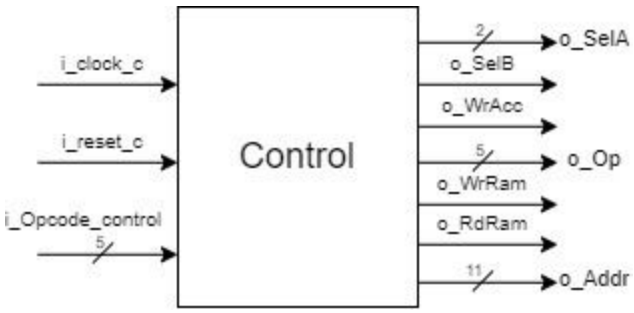
Se implementó utilizando una lógica secuencial mediante la utilización de un bloque always el cual se ejecuta cada vez que ocurre un flanco ascendente de reloj y pondrá la salida de este módulo en alto cada vez que un contador llegue a un valor determinado por la frecuencia y el baud rate utilizados.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/baud_rate_gen.v

Control

Diagrama de bloques



Lógica de implementación

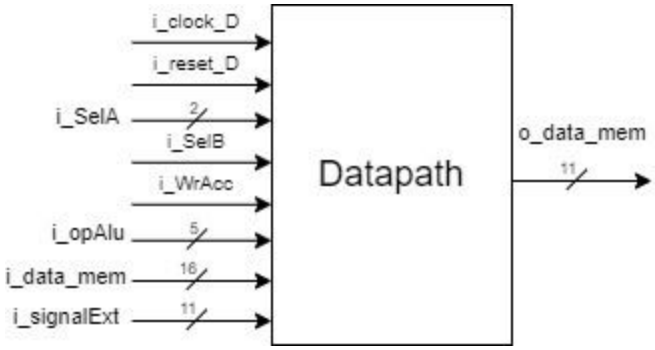
Este módulo de control, se utilizó para instanciar el módulo de Program Counter y el instruction decoder. Tiene como entrada el clock, el reset y el valor de Opcode que viene de cada instrucción y como salidas tiene las mismas que el módulo de instruction decoder.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Control.v

Datapath

Diagrama de bloques



Lógica de implementación

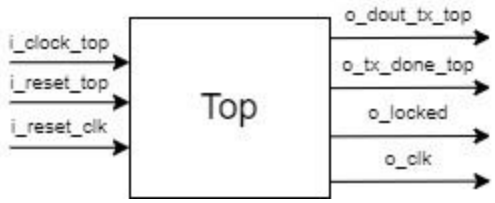
Este módulo se usa para instanciar los módulos antes mencionados. Tiene como entrada las salidas del módulo de control sumado a la la salida de la memoria de datos, que según si está en alto la señal Rd o Wr se va a leer o escribir en la misma. A su vez, además de instanciar los módulos, tiene un always combinacional donde a la salida o_data_mem que va a ser la entrada de la memoria de datos se le va a asignar el cable que tiene el valor de la salida del acumulador. Esto es para que cuando se escriba en memoria (con la señal Wr en alto), ese valor sea el que se guarde en la misma.

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Datapath.v

Módulo Top

Diagrama de bloques



Lógica de implementación

Este módulo se encarga de la instanciación de los módulos Control, Datapath, Program Memory, Data Memory y Tx y de las conexiones entre los mismos. A su vez tiene un always combinacional que va a poner en alto la señal de comienzo de transmisión cuando la operación sea HALT (es decir el Opcode 00000).

Codigo del modulo:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/TOP.v

Desarrollo de testbench

Para validar el funcionamiento del sistema, se implementó un testbench que testea el sistema en conjunto.

Para esto en el mismo se instancia por un lado un top con todos sus componentes internos, y por otro lado, un receptor y un generador de baud rate externos de forma tal que este receptor externo sea el encargado de recibir los datos que va a enviar el transmisor que tiene el TOP. Es decir, una vez que llegue la instrucción de HALT, el transmisor del top va a enviar el valor que se tiene en el acumulador y este receptor externo instanciado en el testbench va a ser el encargado de recibirlo.

Este test consta de un bloque initial y dos bloques always:

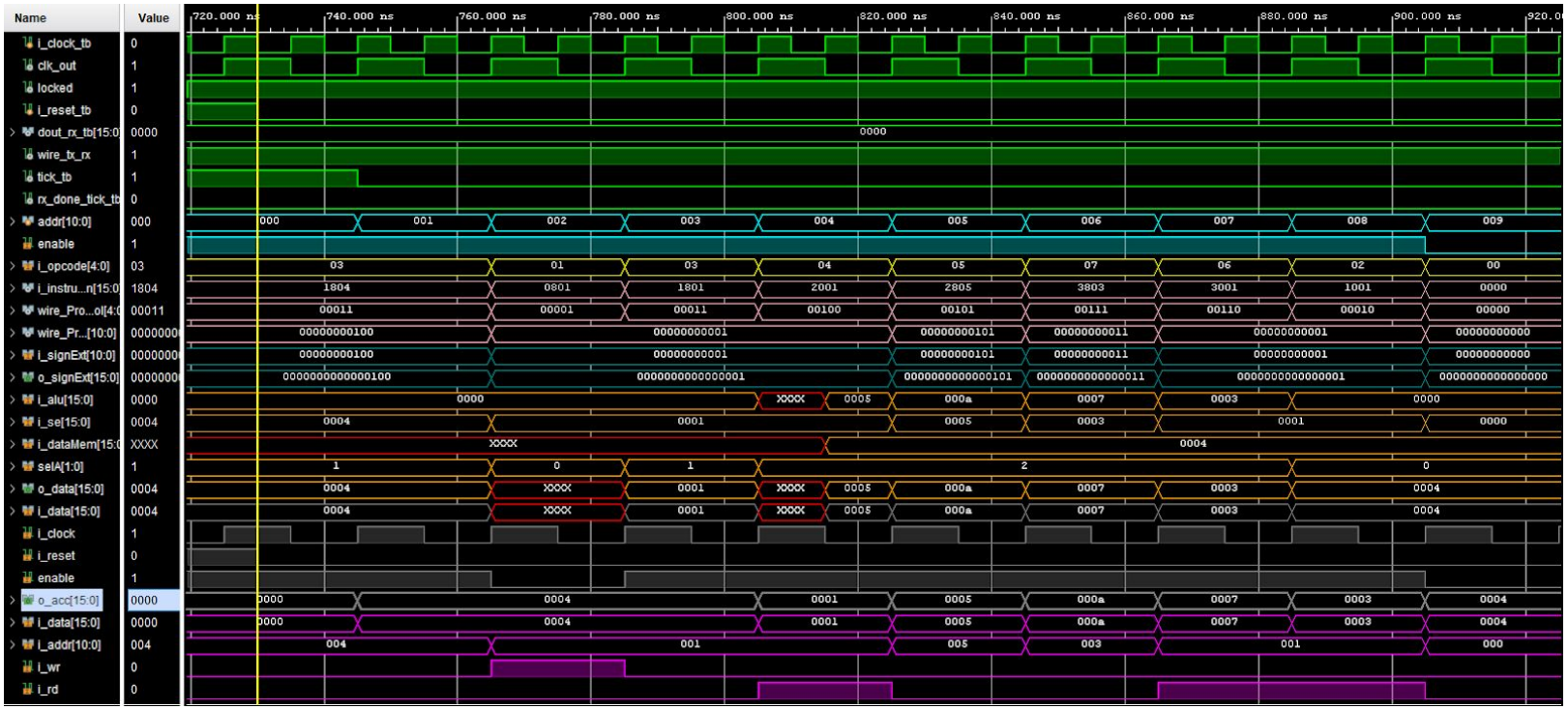
- En el bloque initial primero se resetea el clock wizard implementado y al mismo tiempo se pone en alto el reset que va a los módulos. Luego, en un while se especifica que hasta que la señal “locked” que llega del clock wizard indicando que ya está estable la señal de clock no se ponga en alto, el reset de los módulos siga estando en alto. Cuando esta señal de locked se ponga en alto, se va a poner en bajo el reset de los módulos y el procesador comenzará a incrementar el program counter según las instrucciones que vaya teniendo en la memoria del programa.
- En el primer bloque always se crea un clock externo para que sea la entrada del clock wizard. Cada 5 unidades de tiempo se va a negar esa señal.
- En el segundo bloque always, va a existir un if encargado de ver en todo momento si la señal de finalización de recepción está en alto. De ser así, indicaría que el receptor ya terminó de recibir el valor que estaba en el acumulador a través del transmisor. Por lo tanto, enviará por consola el valor recibido junto al mensaje “Test Finalizó” para indicar que la prueba pasó con éxito.

En el siguiente enlace se puede observar el código de este testbench:

https://github.com/Augustogp/tp3-arqui/blob/master/TP3/TP3.srcs/sources_1/new/Testbench.v

Resultados

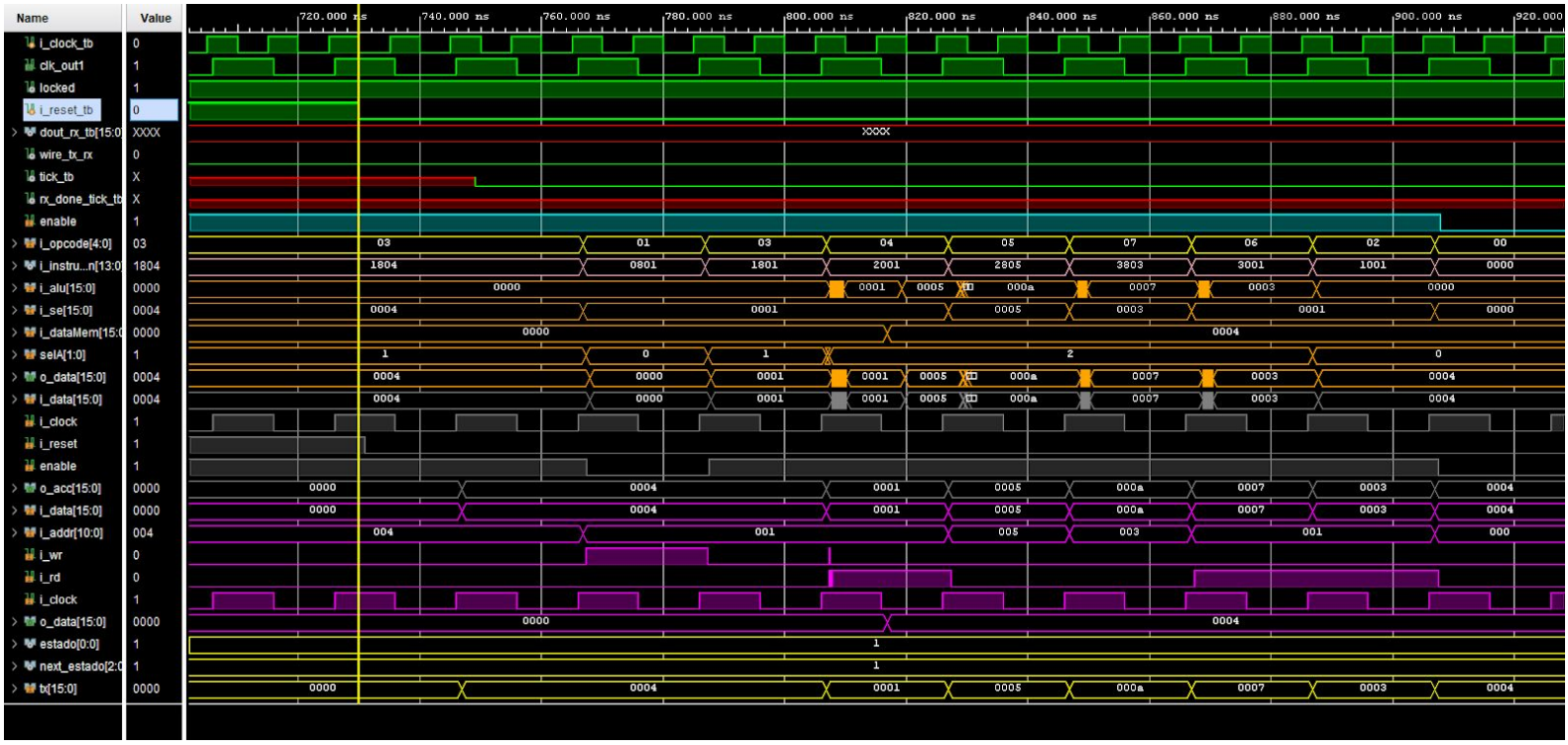
Behavioral Simulation



Al realizar la simulación de comportamiento de este test bench, puede observarse que el clock generado por el clock wizard se estabiliza a partir de aproximadamente 630 ns de forma tal que a partir de ahí se realizó la evaluación de dicho test. Pudo corroborarse que todas las señales se comportan de forma correcta, de forma tal que cada instrucción se logra ejecutar en un ciclo de clock.

Además, al finalizar, al ejecutar la instrucción halt, se puede verificar que se envió correctamente el contenido del acumulador, y que este fue recibido de forma precisa por un receptor externo.

Post-Synthesis Timing Simulation



En esta simulación post síntesis se puede observar un funcionamiento muy similar al visto en el anterior caso, con la suma de cierto jitter en algunas señales debido al retardo temporal que se da en las mismas. Sin embargo, esto no interfiere en el funcionamiento, pudiéndose observar que el proyecto se ejecuta de forma correcta.

Clock Wizard

En los trabajos anteriores, lo que se hizo fue generar el clock en el testbench. En este caso, se decidió hacerlo más realista utilizando el bloque clock wizard que usa el DCM que tiene la FPGA.

Este bloque va a tener la entrada de un clock que va a estar generado en el testbench y la señal de clock que sale del mismo va a ser la utilizada en todos los módulos instanciados en el TOP. Además, se utiliza la señal ”locked” del mismo para saber cuando el clock a la salida de este está estabilizado.

Esquemático del sistema