Universidad
Nacional
de Córdoba

UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Arquitectura de Computadoras

Trabajo Práctico Nro. 3: BIP

Fernández Oria, Luciano

luchoof212@gmail.com

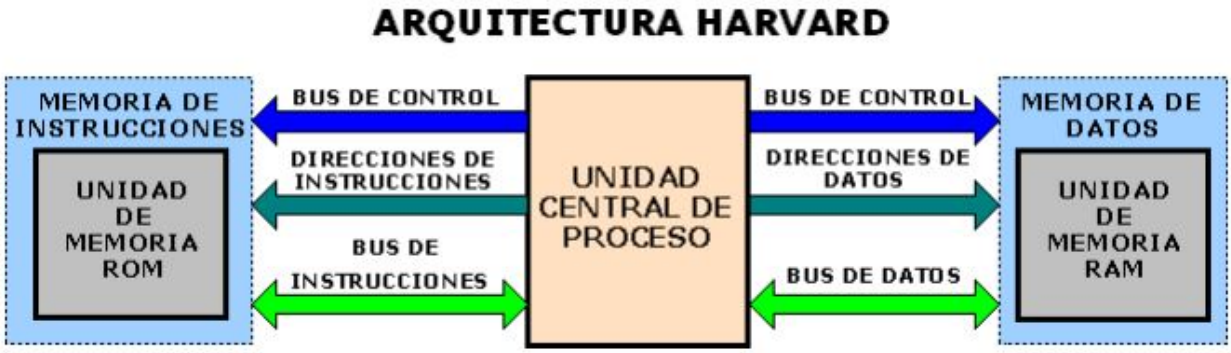
Sieber, Braian

braiansieber@gmail.com

26/3/2019

CONSIGNA:

En este trabajo práctico se nos solicitó desarrollar un procesador de instrucciones basico basado en una arquitectura de monociclo/Harvard (memorias de datos y de programa separadas). La unidad de control debe ser un circuito combinacional y tener solo los registros necesarios. Debe incluir una unidad aritmetica basica que soporte operaciones de suma y resta; y debe tener un unico registro de propposito general.



Atributos de la arquitectura:

- Accumulator-oriented architecture (un unico registro de proposito general).

- Instrucciones y datos de 16 bits.

- Un unico tipo de dato (entero).

- Un unico formato de instrucción.

- Dos modos de direccionamiento (directo e inmediato).

- Set de instrucciones reducido.

Aunque BIP presenta varias caracteristicas de maquinas RISC, no puede ser conciderado un procesador RISC porque no utiliza una arquitectura load-store, la cual solo accede a memoria a traves de estas instrucciones.

Formato de las instrucciones:

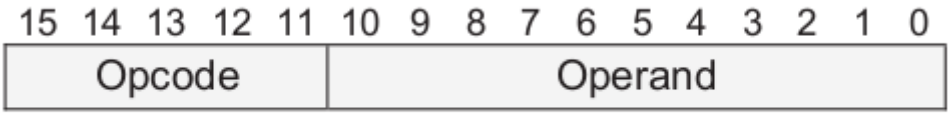


Figure 1. Instruction format

- Opcode: identificador de la operacion (5 bits).

- Operand: identificador del operando. Puede representar un valor inmediato o una direccion de memoria (11 bits).

El procesador contiene solo dos registros:

PC (Program Counter): Guarda la direccion de la instruccion actual.

ACC (Accumulator): trabaja como el operando implicito en muchas instrucciones.

INSTRUCCIONES:

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

La estructura del procesador esta separada en dos bloques:

- Control: decodifica las instrucciones de la memoria de programa y dirije las operaciones en el Datapath. Esta formada por el PC, un sumador de 11 bits y un decodificador de instrucciones combinacional.
- Datapath: procesa los datos. Incluye el registro ACC, la ALU, un bloque para extender la señal de 11 bits a 16 bits, y dos multiplexores.

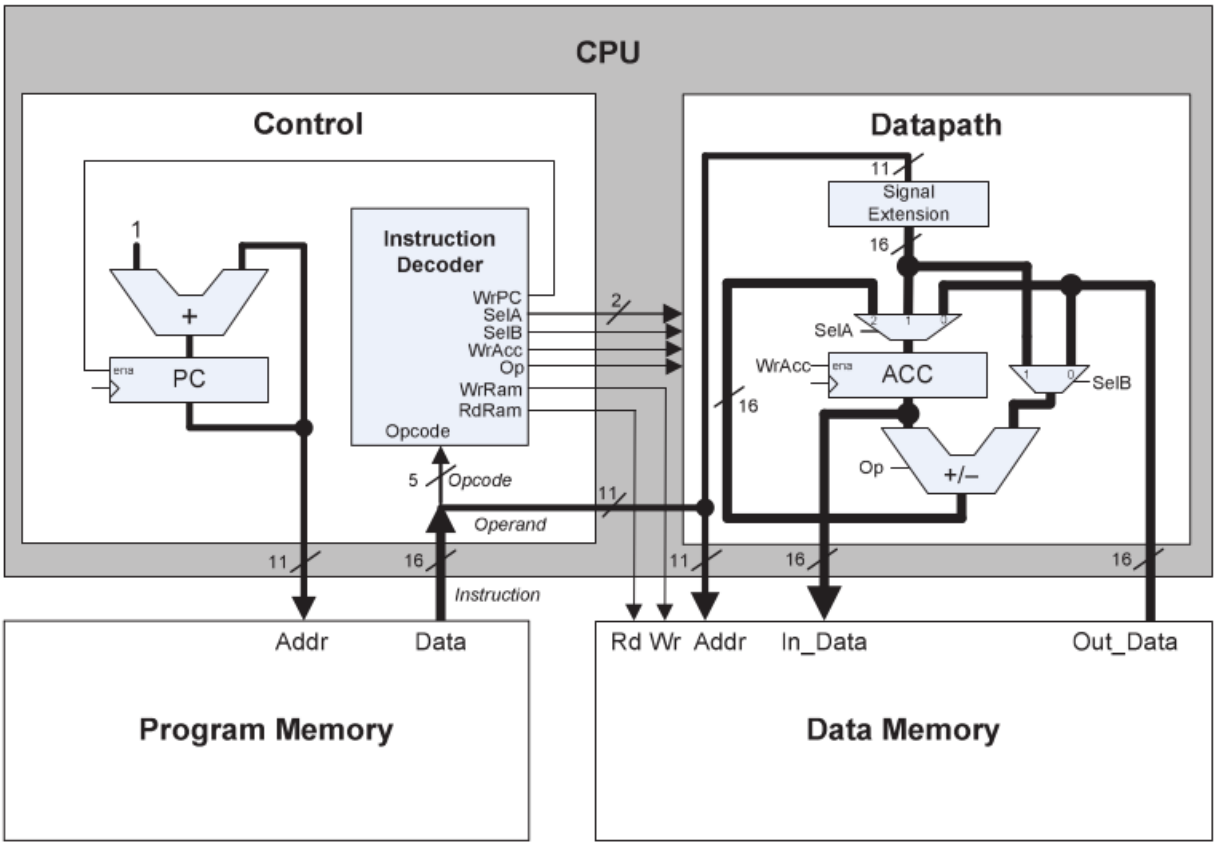


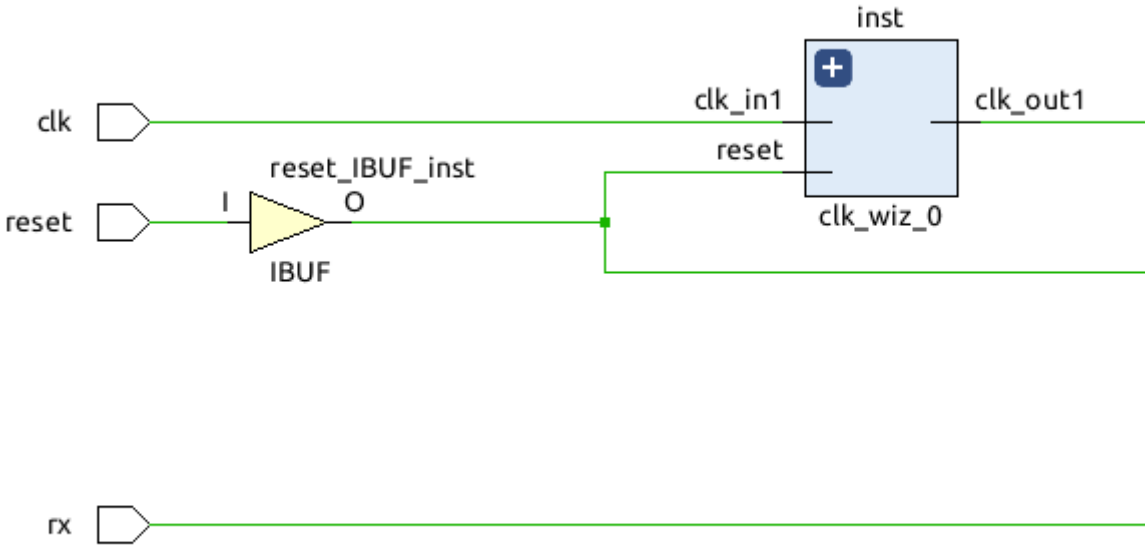
Figure 3. BIP I organization

Implementación

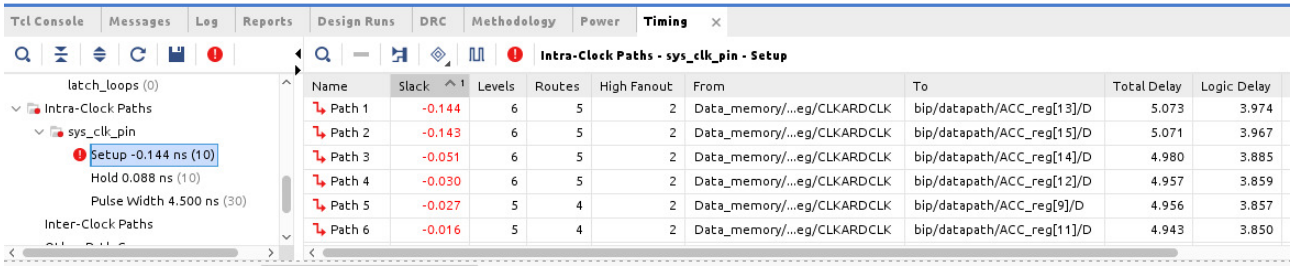
Se implementaron cuatro modulos principales:

- Data Memory: Es una memoria que contiene registros con determinados valores seteados por el usuario.
- Program Memory: Es una memoria donde se almacenan las intrucciones del programa a ejecutar. Ambas memorias funcionan de forma sincrona.
- UART: Es un modulo encargado de recibir las señal enviada desde la PC para habilitar el procesador y, al finalizar el programa, envia a la PC el valor del registro ACC seguido del valor del PC.
- BIP: es la unidad encargada de la decodificación y ejecución de las instrucciones, e incremento del valor del PC en cada ciclo del clock. La mayoria de sus componentes son circuitos combinacionales, excepto la actualización del registro del acumulador.

Para este trabajo debimos reducir la frecuencia del clock de 100 Mhz (el de la placa) a 50 Mhz utilizando el clock wizard:

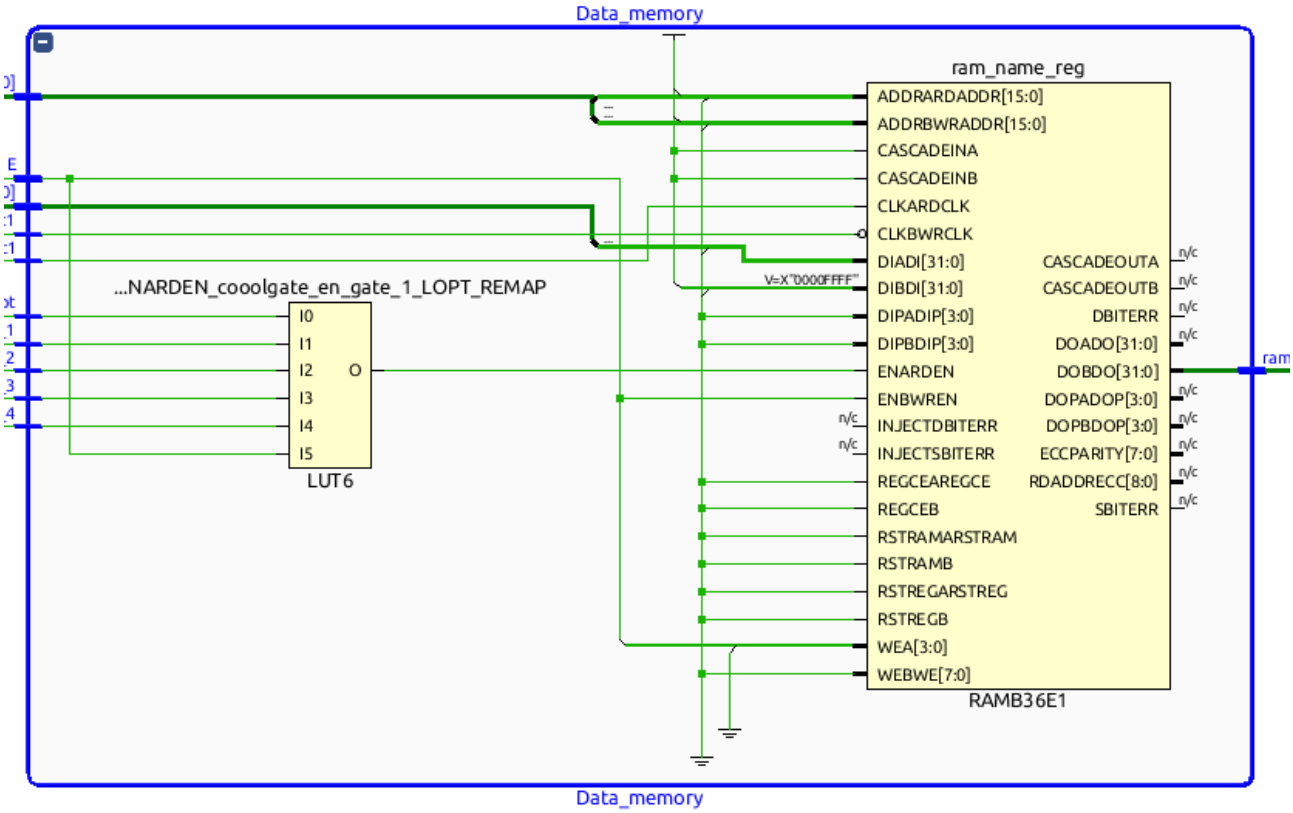


Esto se debio a problemas con el timing report, lo cual nos generaba una advertencia critica:



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay
Path 1	-0.144	6	5	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[13]/D	5.073	3.974
Path 2	-0.143	6	5	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[15]/D	5.071	3.967
Path 3	-0.051	6	5	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[14]/D	4.980	3.885
Path 4	-0.030	6	5	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[12]/D	4.957	3.859
Path 5	-0.027	5	4	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[9]/D	4.956	3.857
Path 6	-0.016	5	4	2	Data_memory/...eg/CLKARDCLK	bip/datapath/ACC_reg[11]/D	4.943	3.850

El delay maximo permitido era de 5 nseg, ya que el periodo del clock era de 10 nseg. La causa fue que utilizamos dos memorias, una para las instrucciones y otra para los datos:



Para poder ejecutar una instrucción por ciclo de clock pusimos la memoria de datos para que trabaje en el flanco descendete del clock, lo que genero que un delay mayor al permitido en la ruta hacia el procesador, ya que el mismo actualiza el valor del acumulador en flanco ascendete.

Este delay lo redujimos haciendo que la escritura en la memoria se ejecute en el flanco ascendete pero solo bajando la frecuencia del clock pudimos eliminarlo.

Clock Summary

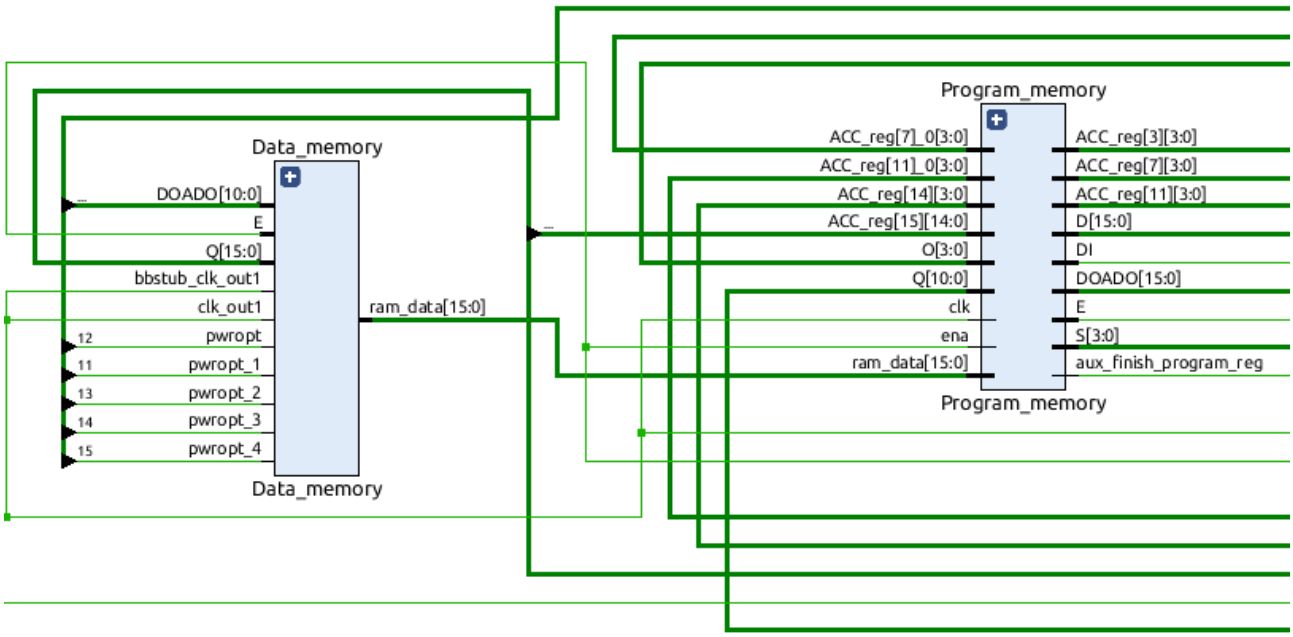
Name	Waveform	Period (ns)	Frequency (MHz)
clk	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 10.000}	20.000	50.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0_1	{0.000 10.000}	20.000	50.000
clkfbout_clk_wiz_0_1	{0.000 5.000}	10.000	100.000

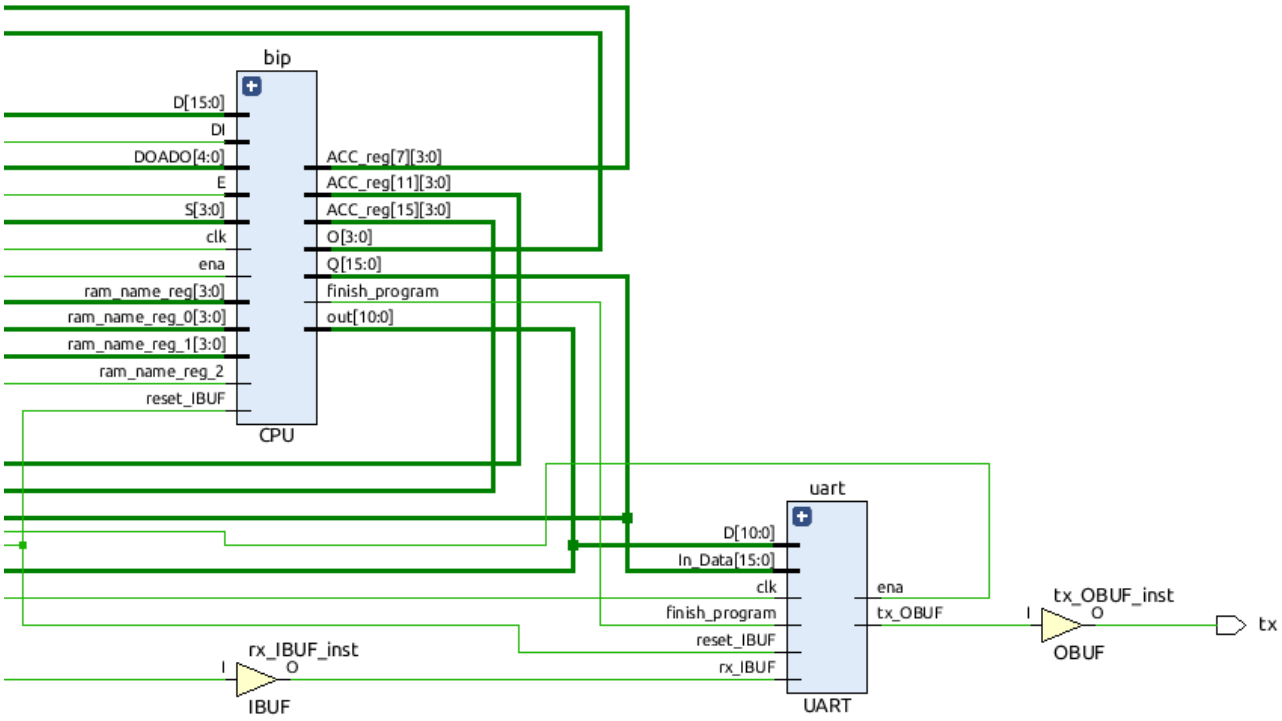
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.159 ns	Worst Hold Slack (WHS):	0.071 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	339	Total Number of Endpoints:	339	Total Number of Endpoints:	166

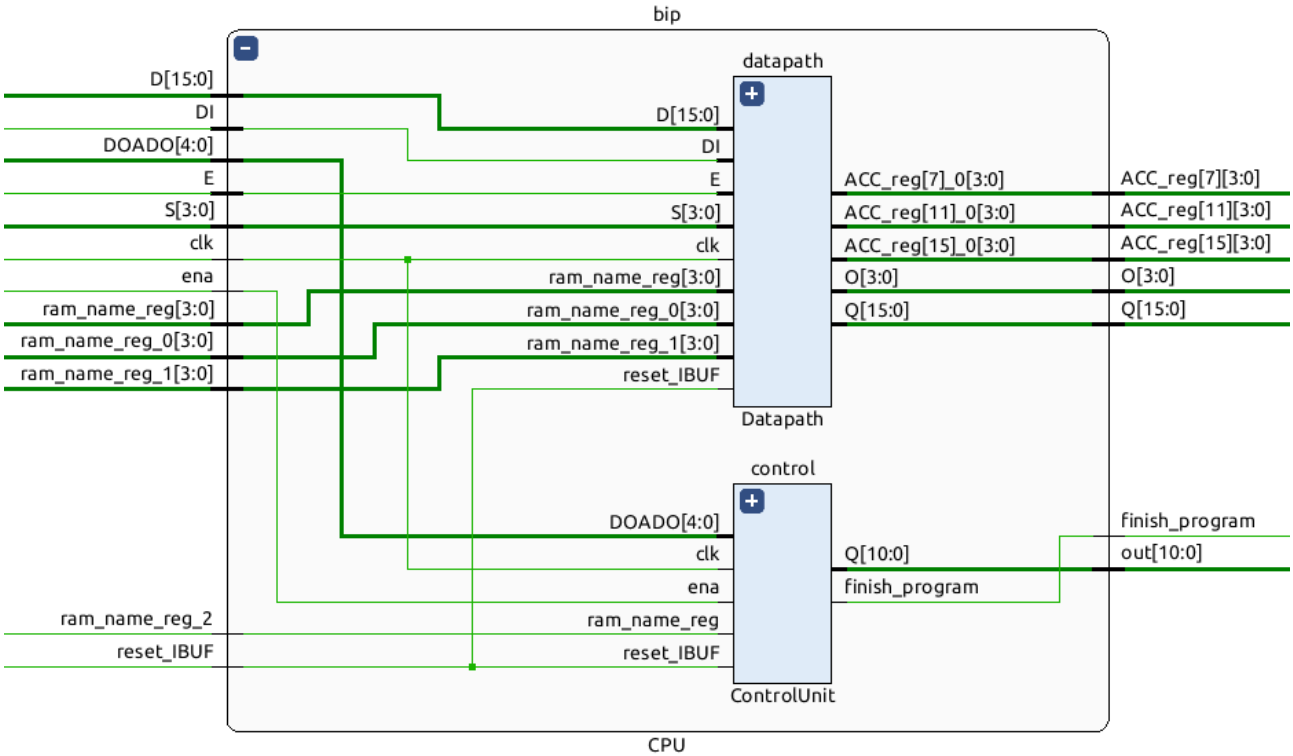
All user specified timing constraints are met.

El esquematico del sistema completo es el siguiente:

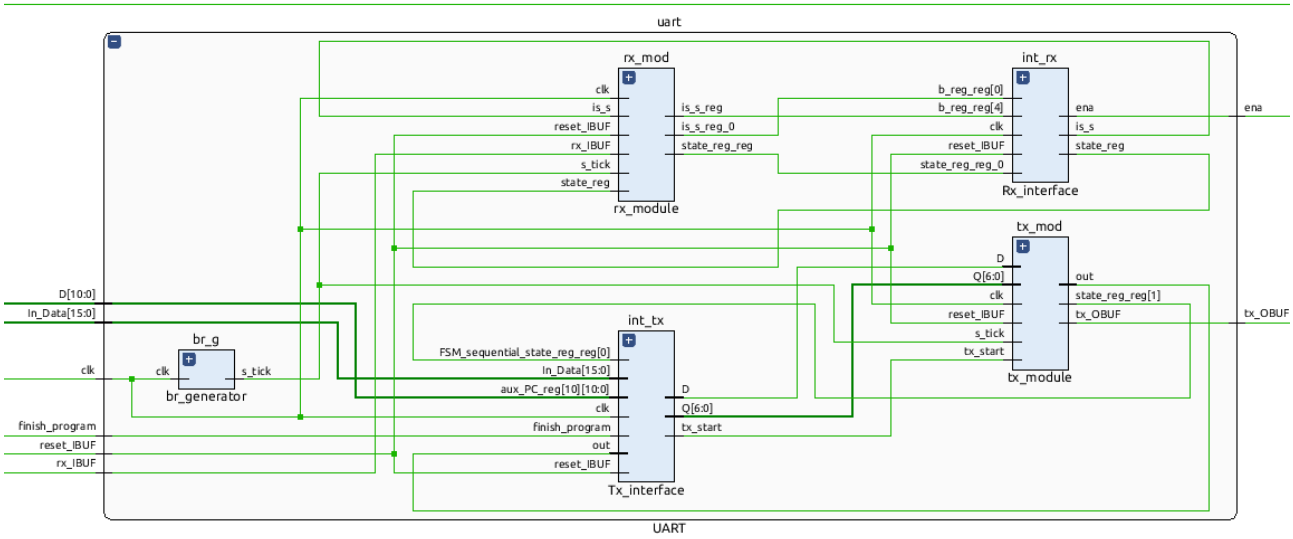




A su vez el modulo BIP contiende dos modulos, uno encargado de manipular las instrucciones (Control) y otro de ejecutarlas (Datapath).



Para iniciar el procesador debemos enviar dos simbolos por medio de interfaz UART, una “s” y luego la tecla Enter; luego de eso, el procesador ejecuta todas las instrucciones hasta encontrarse como una instruccion HALT, la cual hace que levante la bandera finish_program para que el modulo de UART envie los datos.



Debimos modificar la interfaz de transmision Tx_interface del TP 2, ya que tambien nos generaba problemas de timing, debido a la complejidad que genereba el utilizar la operacion de division para obtener los valores a enviar. A diferencia del TP 2, ahora debiamos enviar valores de hasta 16 bits por lo que ejecutabamos mayor cantidad de divisiones.

```
operate :
    begin
        dig = aux / div;     // divido para obtener el digito a transmitir (ej, 123/100 - obtengo 1 en it. 1)
        div = div / 10;      // Divido por 10 para en la sig iteración obtener el sig digito (100/10=10)
        if(dig || zflag == 1) state_reg = transmit; // Entro si dig != 0 ó zflag = 1 si ya transmiti un valor y
        salida = dig+48; // Sumo 48 al digito enviado para transmitir en ascii
    end
```

Reemplazamos el sistema de divisiones por otro que solo ejecuta sumas y restas. Esto redujo el tamaño del modulo y su complejidad. Por lo que pudimos investigar, ejecutar una division por hardware es algo complejo y su utilizacion no es recomendada.

```
operate:
  begin
      if (j<5) aux2= aux - div[j];
      if (j == 5) // Resetamos todos los parametros
         begin
             z_flag = 1'b0;
             j= 0;
             if (acc_sended==1) state_reg = idle_tx;
             else state_reg = transmit_reset;
             acc_sended = 1;
             aux = aux_Count;
                        out = array_char[i]; // salto de linea
         end // if (div == 0)
      else if (aux2>=0)
          begin
              aux = aux - div[j];
              out = out + 1;
          end
      else if ((out>0) || z_flag)
          begin
              state_prev = state_reg;
              state_reg = transmit_on;
              j = j+1;
              out= out+48;
              tx_start_aux = 1'b1;
              z_flag = 1'b1;
          end
      else j = j+1;
  end
```

Simulaciones por medio de testbench:

Tanto la memoria de datos como la de programa cargan los datos de archivos de texto guardados en la PC.

Datos.txt:

0x000C // 12
0x0008 // 8

Innstrucciones.txt:

0x1805
0x2000
0x0802
0x3001
0x3806
0x2818
0x0803
0x1002
0x2003
0x0000
// - OPERATION VAL / ACC
//0001 1000 0000 0101 - Load Immediate 5 / 5
//0010 0000 0000 0000 - Add Variable #0 (12) / 17
//0000 1000 0000 0010 - Store Variable #2 / 17
//0011 0000 0000 0001 - Subtract Variable #1 (8) / 9
//0011 1000 0000 0110 - Subtract Immediate 6 / 3
//0010 1000 0001 1000 - Add Immediate 24 / 27
//0000 1000 0000 0011 - Store Variable #3 / 27
//0001 0000 0000 0010 - Load Variable #2 / 17
//0010 0000 0000 0011 - Add Variable #3 / 44
//0000 0000 0000 0000 – halt

Como vemos en el archivo de instrucciones, el valor del acumulador debe resultar en 44. Este valor es luego transmitido por UART a la PC.

Ejecutamos la siguiente simulación:

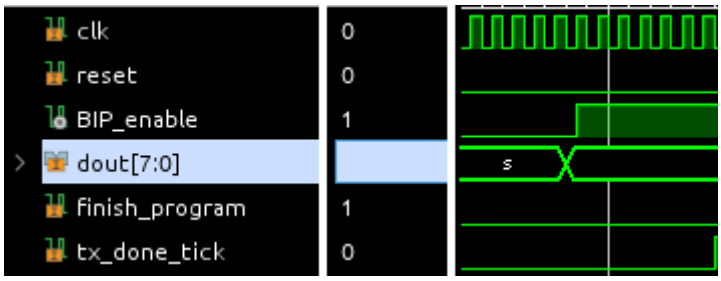
```
Data_memory #(.INIT_FILE("/home/vlad/Arquitectura2018/datos.txt"))
Data_memory
(.Wr(WrRAM),.clk(clk),.ena(BIP_enable), .Addr(Addr), .In_Data(In_Data), .Out_Dat
a(Out_Data));
   Program_memory
#(.INIT_FILE("/home/vlad/Arquitectura2018/instrucciones.txt")) Program_memory
(1'b0, clk, BIP_enable, PC, 0, Program_Data);
   CPU bip(BIP_enable, clk, reset, Program_Data, Out_Data, In_Data, PC, WrRAM,
RdRAM, finish_program);

   Rx_interface #(.DBIT(8)) int_rx (clk, reset,rx_done_tick, dout, BIP_enable);

   Tx_interface #(.DBIT(8)) int_tx (clk, reset, finish_program, tx_done_tick,
out_Acc_Counter, din, tx_start);


   assign out_Acc_Counter[31:27] = 0;
   assign out_Acc_Counter[26:16] = PC;
   assign out_Acc_Counter[15:0] = In_Data;
   assign Addr = Program_Data[10:0];

   always
      begin
         #5 clk = ~clk;
      end
    initial
      begin
         clk = 0;
         reset = 1;
         #10 reset = 0;
         #10  tx_done_tick = 0;
         #10  tx_done_tick = 1;
         #10  tx_done_tick = 0;
         #10  tx_done_tick = 1;
         #10  tx_done_tick = 0;
         #10  tx_done_tick = 1;
         #10  tx_done_tick = 0;
         #10  tx_done_tick = 1;
         #10  tx_done_tick = 0;
         #10  tx_done_tick = 1;
```

```
        #10  tx_done_tick = 0;
        #10
        rx_done_tick = 1;
        dout = 115;
        #50;
        dout = 13;
        #50;
        #10  tx_done_tick = 0;
        #10  tx_done_tick = 1;
```

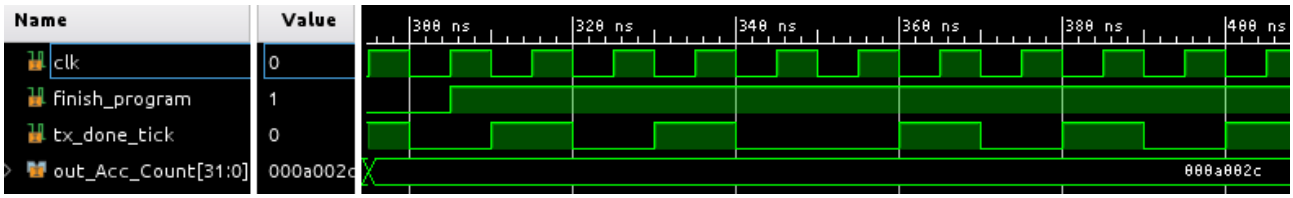
… Seguimos ejecutamos tx done hasta poder simular el envio de todos los caracteres.

```
    end
endmodule
```

Como podemos ver en la siguiente imagen, el procesador se activa una vez que el BIP_enable es puesto en alto:

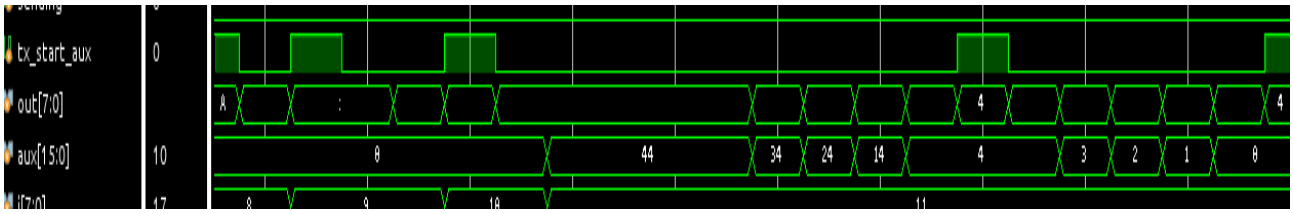


Cuando el procesador termina de ejecutar las instrucciones se levanta la bandera de finish_program y podemos ver el valor del PC y el ACC en el registro de 32 bits llamado out_Acc_Count:

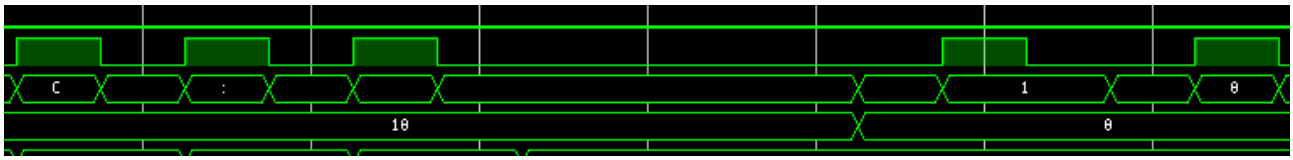


El valor esta en hexadecimal, el contador 000a (10) y 002c (44).

Luego debemos enviar los digitos por UART. Primero enviamos “A: 44”. Lo que podemos ver en la siguiente imagen:



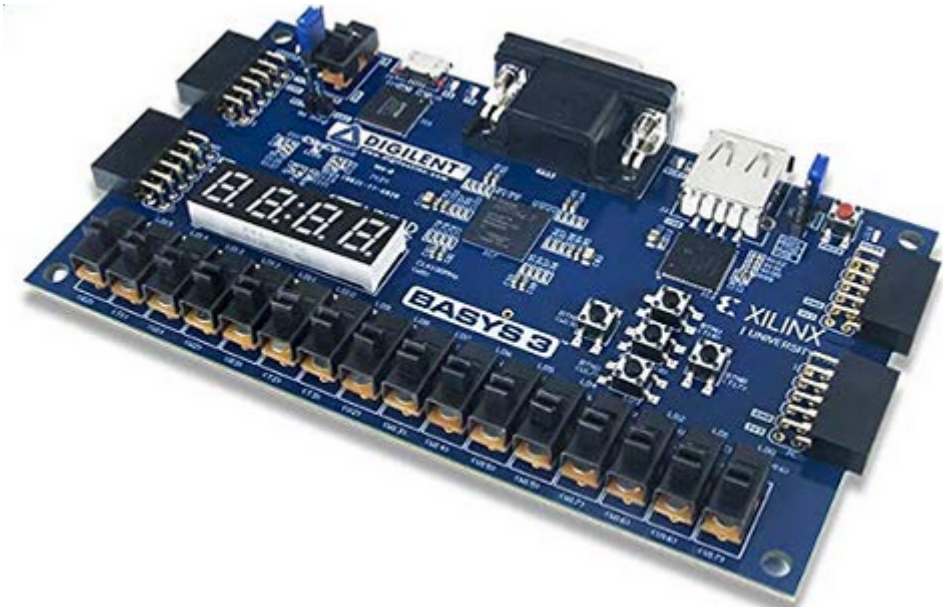
Vemos cuando se pone el valor en ASCII en la salida “out” se levanta la bandera tx_start para que el modulo envie el valor.



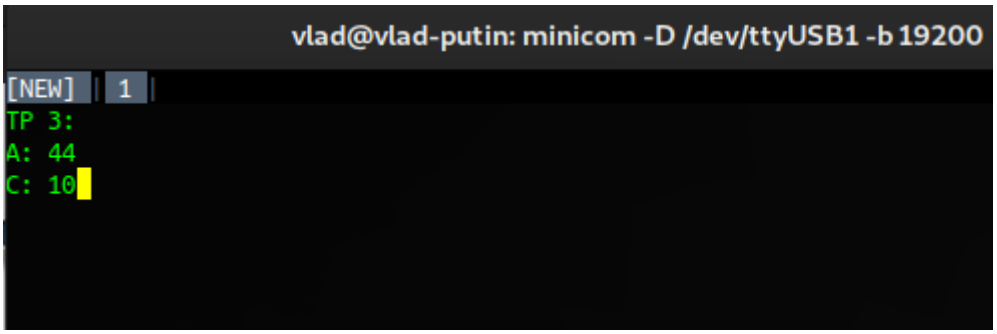
Podemos ver lo mismo con el valor del contador.

La placa que utilizamos fue la Basys3, que trabaja a una frecuencia de 100 Mhz.

-



Comprobamos el funcionamiento descargando el programa en la misma, y usamos minicom para comunicarnos a una frecuencia de 19200 baudios:



Conclusión:

Este trabajo nos permitio conocer el funcionamiento de un procesador basico, ademas de profundizar los conocimientos sobre lenguajes HDL y las herramientas que provee el entorno de desarrollo Vivado. Luego de enfrentarnos a problemas de timing pudimos solucionarlos y obterner los resultados esperados.

Bibliografía:

- a basic processor for teaching digital circuits and systems design with fpga (paper).
- FPGA prototyping by VHDL examples.