



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Arquitectura de Computadoras

“Implementación del Microprocesador de 4 bits”

Alumno:

Malagón Baeza Alan Adrian

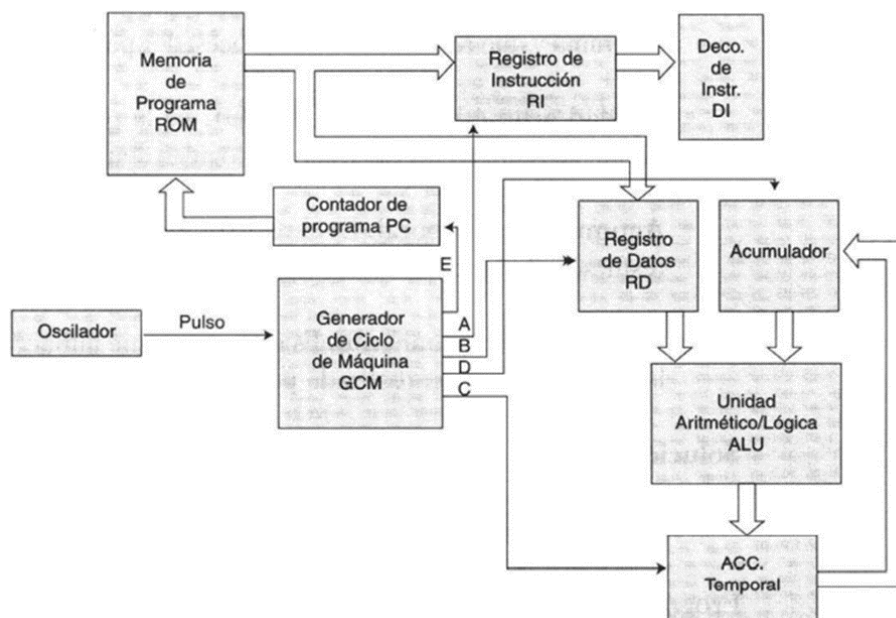
Profesor:

Alemán Arce Miguel Ángel

Grupo: 5CV1

Introducción

Objetivo: Implementar, en lenguaje HDL, el microprocesador de 4 bits de la figura siguiente:



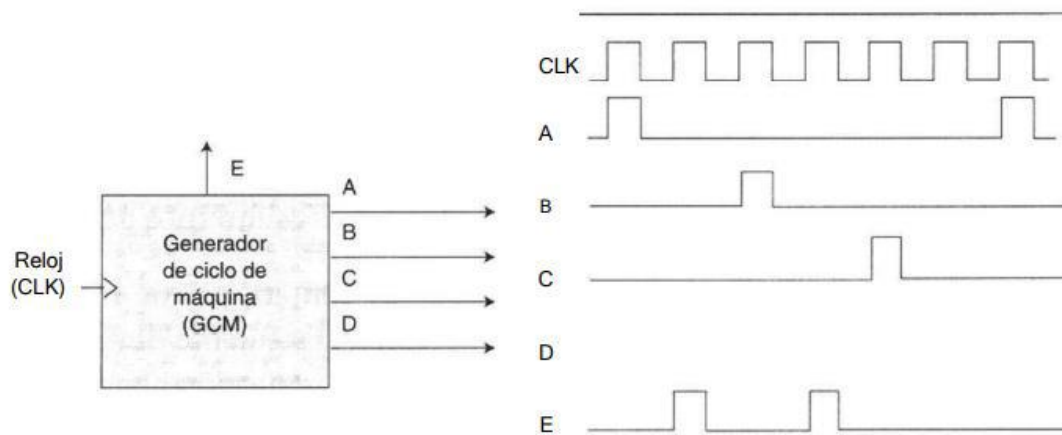
- **Memoria ROM.** Este módulo almacena una instrucción y un dato; así sucesivamente. El contador de programa accederá a cada una de sus direcciones donde la instrucción o el dato pasarán al registro de instrucción o al registro de datos según corresponda.
- **Generador de ciclo de máquina o unidad de control.** Este módulo es la unidad de control del microprocesador y su función es sincronizar y activar la participación de cada uno de los registros internos del microprocesador.
- **Contador de programa.** El contador es un registro interno del microprocesador que proporciona la siguiente dirección de memoria, sea para introducir un dato o una instrucción al microprocesador.
- **Registro de instrucción.** Almacena temporalmente la instrucción que se va a ejecutar en el microprocesador.
- **Decodificador de instrucción.** Es el elemento utilizado para interpretar y ejecutar la instrucción que se requiere realizar en la unidad aritmética y lógica.
- **Registro de datos.** Almacena los datos que provienen de la memoria de programa y que se requiere realizar en la unidad aritmética y lógica.

- **Unidad aritmética y lógica (ALU).** Es la parte del microprocesador la parte del microprocesador donde se realizan las operaciones lógicas y aritméticas.
- **Acumulador temporal.** Es el registro que almacena temporalmente el resultado de la última operación realizada dentro de la ALU.
- **Acumulador permanente.** Es el registro que almacena el resultado de última operación realizada por el microprocesador.

Desarrollo

- **Programación del generador de ciclo de máquina.**

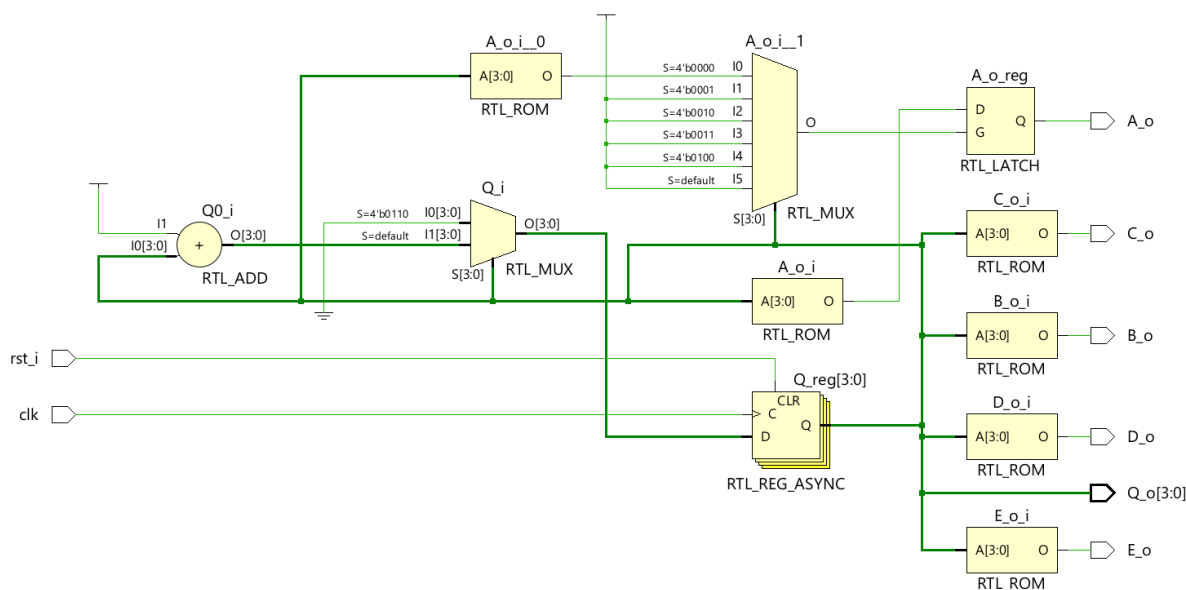
Este módulo coordina los procesos que realiza el microprocesador; utiliza cinco señales de control que activan en orden secuencial control que activan en orden secuencial los registros internos del microprocesador: registro de instrucción (A), o de instrucción (A), registro de datos (B), registro de datos (B), acumulador temporal (C), acumulador permanente (D) y contador de programa (E). y contador de programa (E). En la siguiente figura se muestra este módulo y el diagrama de tiempo de las señales de activación.



El programa quedará de la siguiente manera:

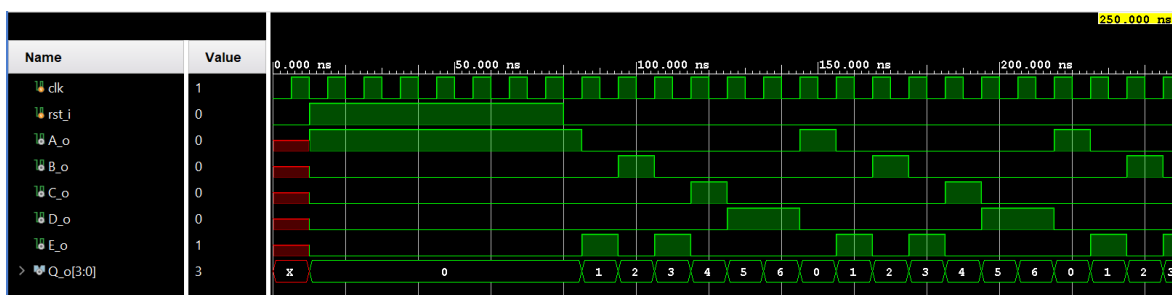
```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: GCM Generador de Ciclo de Máquina
9  // FPGA: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module gcm (
14     input clk,
15     input rst_i,
16     output reg A_o,
17     output reg B_o,
18     output reg C_o,
19     output reg D_o,
20     output reg E_o,
21     output [3:0] Q_o
22 );
23
24 reg [3:0] Q;
25
26 always @(posedge clk or posedge rst_i)
27 begin
28     if (rst_i)
29         Q <= 0;
30     else if (Q == 4'b0110)
31         Q <= 0;
32     else if (clk)
33         Q <= Q +1;
34     end
35
36 always @(Q)
37 case (Q)
38     4'b0000 : begin A_o = 1'b1; E_o <= 1'b0; B_o <= 1'b0; C_o <= 1'b0; D_o <= 1'b0; end
39     4'b0001 : begin A_o <= 1'b0; E_o <= 1'b1; B_o <= 1'b0; C_o <= 1'b0; D_o <= 1'b0; end
40     4'b0010 : begin A_o <= 1'b0; E_o <= 1'b0; B_o <= 1'b1; C_o <= 1'b0; D_o <= 1'b0; end
41     4'b0011 : begin A_o <= 1'b0; E_o <= 1'b1; B_o <= 1'b0; C_o <= 1'b0; D_o <= 1'b0; end
42     4'b0100 : begin A_o <= 1'b0; E_o <= 1'b0; B_o <= 1'b0; C_o <= 1'b1; D_o <= 1'b0; end
43     default : begin A_o <= 1'b0; E_o <= 1'b0; B_o <= 1'b0; C_o <= 1'b0; D_o <= 1'b1; end
44 endcase
45
46 assign Q_o = Q;
47 endmodule
```


Descripción RTL obtenida mediante Vivado 2022.2:



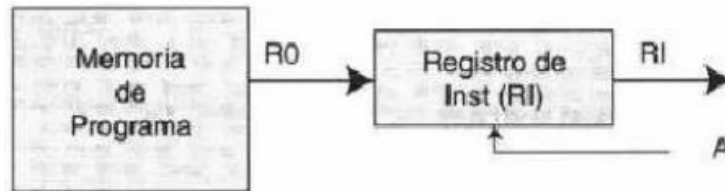
Podemos observar cómo utilizo FlipFlops, multiplexores, operador de suma y bloques de memoria ROM del FPGA seleccionado, y para el control de los datos de salida, se implementó mediante flipflops y bloques de memoria ROM del FPGA seleccionado.

Resultado de la simulación:



- Programación del registro de instrucción (RI).

Este módulo almacena temporalmente las instrucciones provenientes de la memoria de programa. Su función es guardar el código binario de la instrucción mediante la señal de habilitación (A) que envía el generador de ciclo de máquina.



El programa quedará de la siguiente manera:

```

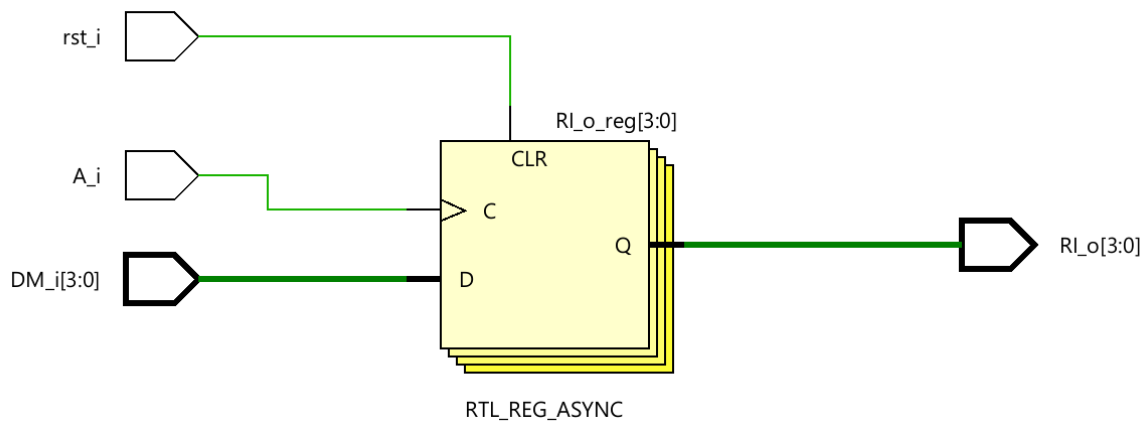
reg_ins.v x tb_reg_ins.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/reg_ins.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: RI Registro de instrucción
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 //Modulo Registo de instrucciones
14 module reg_ins (
15     input A_i,
16     input rst_i,
17     input [3:0] DM_i,
18     output reg [3:0] RI_o
19 );
20 always @(posedge A_i or posedge rst_i)
21 begin
22     if (rst_i)
23         RI_o <= 4'b0000;
24     else
25         RI_o <= DM_i;
26 end
27
28 endmodule
  
```

```
reg_ins.v x tb_reg_ins.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sim_1/new/tb_reg_ins.v

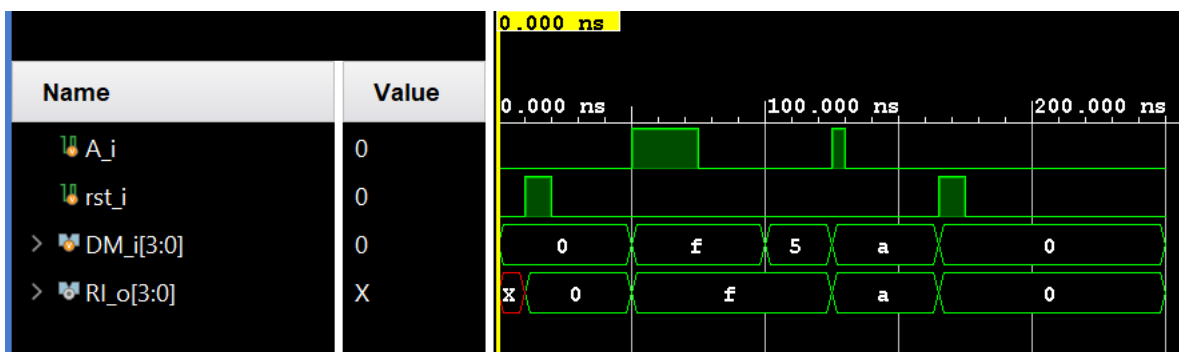
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: RI Registro de instruccion
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | //Módulo de Estimulo para el FlipFlop JK
14 |
15 | module tb_reg_ins ();
16 |     reg      A_i;
17 |     reg      rst_i;
18 |     reg [3:0] DM_i;
19 |     wire [3:0] RI_o;
20 |
21 |     reg_ins RI1(A_i, rst_i, DM_i, RI_o);
22 |
23 | // initial CLK = 1'b0; //Inicializamos el reloj
24 | // always #5 CLK = ~CLK; //El ciclo del Reloj (cambia cada 5 ns)
25 | //
26 | initial
27 |     begin
28 |
29 |         rst_i = 1'b0; A_i = 1'b0; DM_i = 4'b0000;
30 |         #10 rst_i = 1'b1; A_i = 1'b0; DM_i = 4'b0000;
31 |         #10 rst_i = 1'b0; A_i = 1'b0; DM_i = 4'b0000;
32 |         #25 A_i = 1'b0; DM_i = 4'b0000;
33 |         #5 A_i = 1'b1; DM_i = 4'b1111;
34 |         #25 A_i = 1'b0; DM_i = 4'b1111;
35 |         #25 A_i = 1'b0; DM_i = 4'b0101;
36 |         #25 A_i = 1'b1; DM_i = 4'b1010;
37 |         #5 A_i = 1'b0; DM_i = 4'b1010;
38 |         #25 A_i = 1'b0; DM_i = 4'b1010;
39 |         #10 rst_i = 1'b1; A_i = 1'b0; DM_i = 4'b0000;
40 |         #10 rst_i = 1'b0; A_i = 1'b0; DM_i = 4'b0000;
41 |         #25;
42 |     end
43 | initial
44 |     #250 $finish;
45 |
46 | endmodule
```


Descripción RTL obtenida mediante Vivado 2022.2:



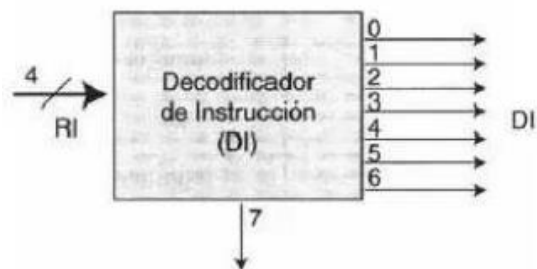
Podemos observar cómo utilizo FlipFlops con su terminal de clk para el control de salida.

Resultado de la simulación:



- Programación del decodificador de instrucción (DI).

La función de este bloque es convertir el código binario proveniente del registro de instrucción en una acción particular, la cual habilita una de varias operaciones lógicas o aritméticas dentro de la ALU.



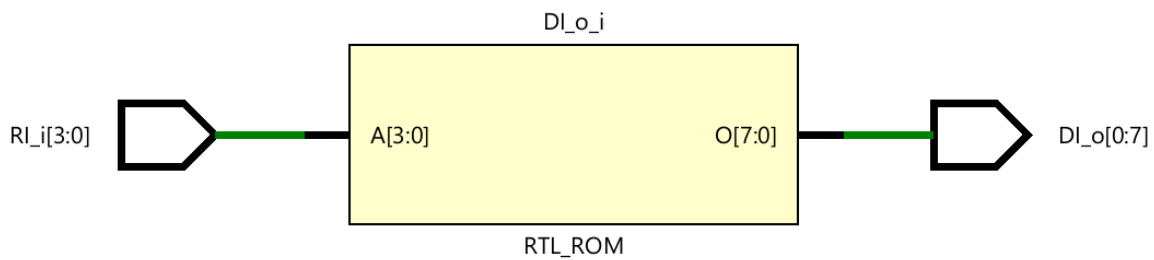
El programa quedará de la siguiente manera:

```
deco_ins.v

C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/deco_ins.v

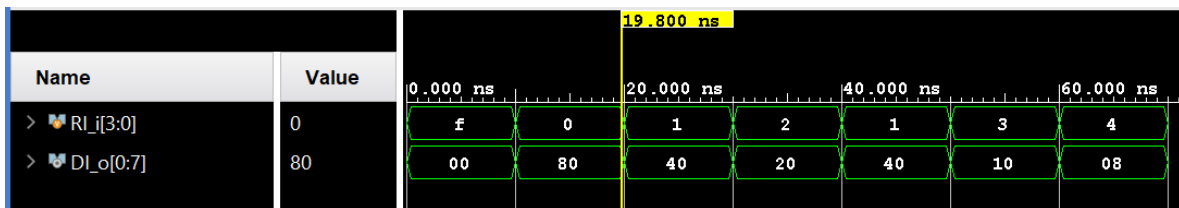
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: SCV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: DI Decodificador de instrucción (DI)
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | module deco_ins(
14 |     input [3:0] RI_i,
15 |     output reg [0:7] DI_o
16 | );
17 |
18 | parameter AND=4'b0000, OR=4'b0001, XOR=4'b0010, SUMA=4'b0011, INV=4'b0100, HOLD=4'b0101, LOAD=4'b0110, RST=4'b0111, UNAB=4'b1000;
19 |
20 | always @(RI_i)
21 |     case (RI_i)
22 |         AND : DI_o = 8'b10000000;
23 |         OR  : DI_o = 8'b01000000;
24 |         XOR : DI_o = 8'b00100000;
25 |         SUMA: DI_o = 8'b00010000;
26 |         INV : DI_o = 8'b00001000;
27 |         HOLD: DI_o = 8'b00000100;
28 |         LOAD: DI_o = 8'b00000010;
29 |         RST : DI_o = 8'b00000001;
30 |         default: DI_o = 8'b00000000; //Deshabilita DI
31 |     endcase
32 |
33 | endmodule
```


Descripción RTL obtenida mediante Vivado 2022.2:



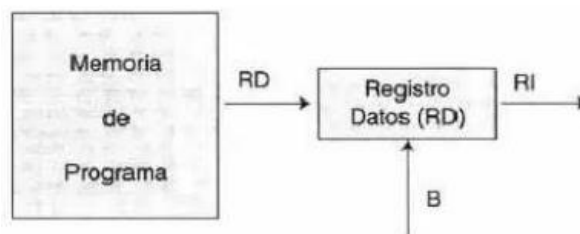
Podemos observar cómo utilizo un bloque de memoria ROM del FPGA seleccionado para el control de salida.

Resultado de la simulación:

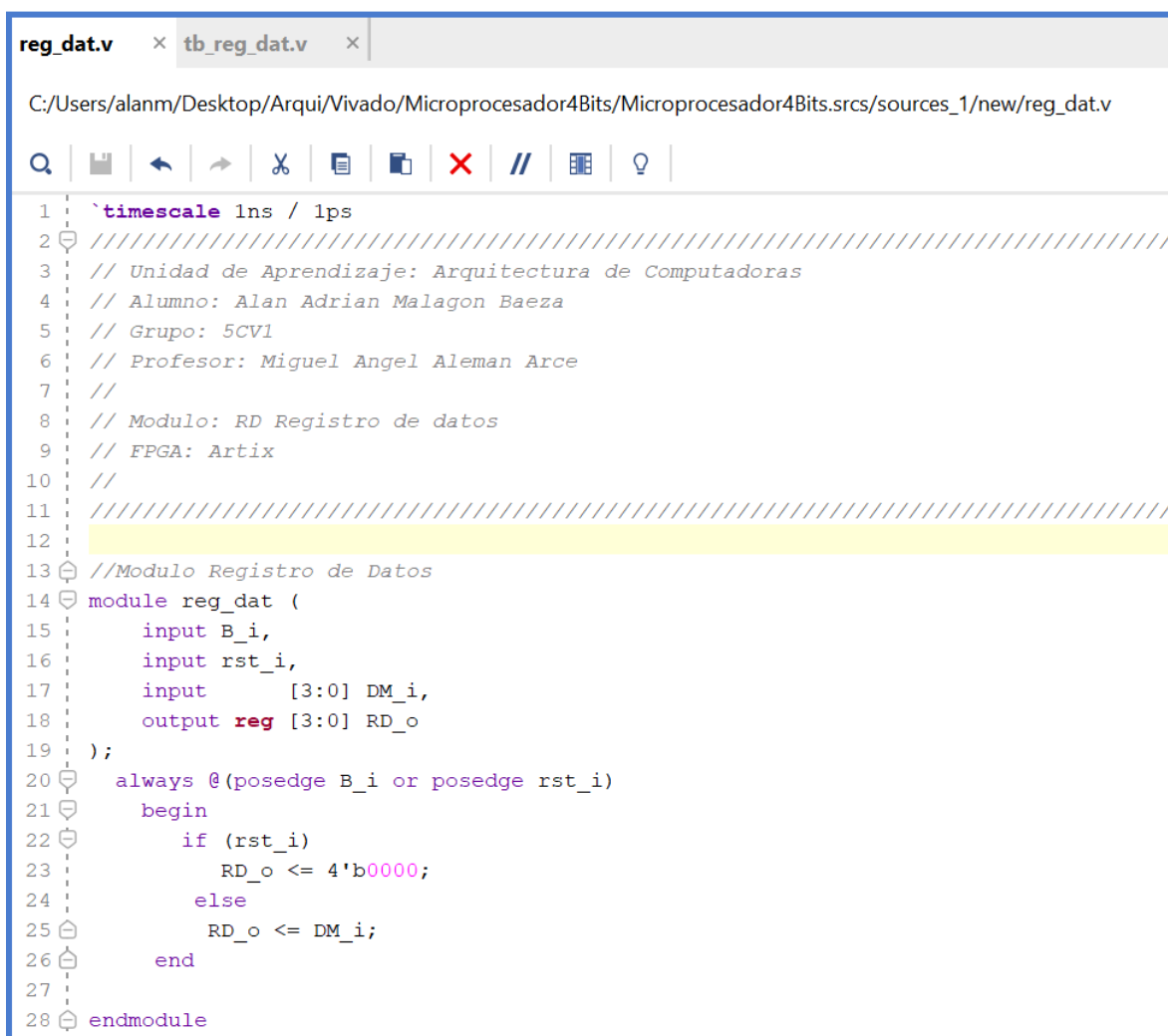


- Programación del registro de datos (RD).

Este módulo almacena temporalmente los datos provenientes de la memoria de programa. Su función es guardar el dato correspondiente mediante la señal de habilitación (B) que envía el generador de ciclo de máquina.



El programa quedará de la siguiente manera:



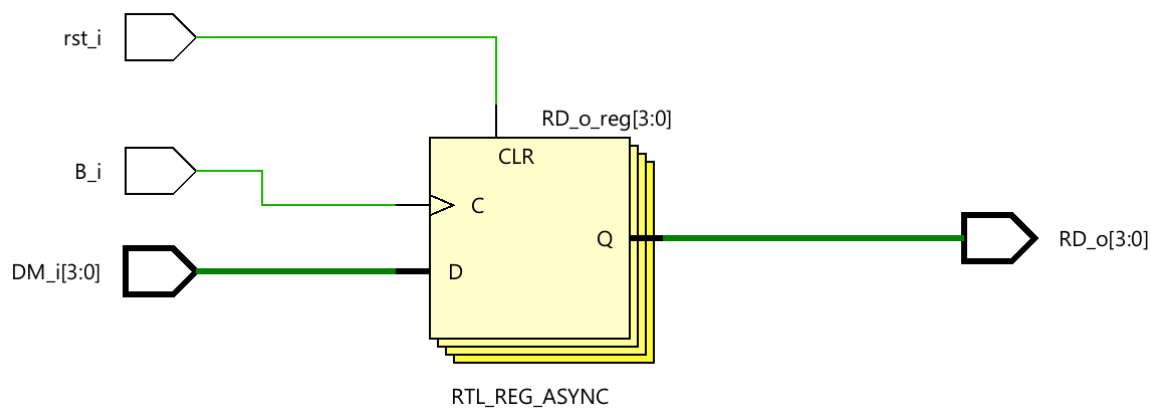
The screenshot shows a Verilog code editor with two tabs: 'reg_dat.v' and 'tb_reg_dat.v'. The file path is 'C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/reg_dat.v'. The code defines a module 'reg_dat' with inputs 'B_i', 'rst_i', and 'DM_i', and an output 'RD_o'. It includes a reset condition where 'RD_o' is set to '4'b0000' if 'rst_i' is active. Otherwise, 'RD_o' is assigned the value of 'DM_i'. The code is as follows:

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: RD Registro de datos
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////
12
13 //Modulo Registro de Datos
14 module reg_dat (
15     input B_i,
16     input rst_i,
17     input [3:0] DM_i,
18     output reg [3:0] RD_o
19 );
20 always @(posedge B_i or posedge rst_i)
21 begin
22     if (rst_i)
23         RD_o <= 4'b0000;
24     else
25         RD_o <= DM_i;
26     end
27
28 endmodule
```

```
reg_dat.v x tb_reg_dat.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sim_1/new/tb_reg_dat.v

1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: RD Registro de datos
9 // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 //Módulo de Estimulo para el registro de Datos
14
15 module tb_reg_dat ();
16     reg      B_i;
17     reg      rst_i;
18     reg [3:0] DM_i;
19     wire [3:0] RD_o;
20
21     reg_dat RD1(B_i, rst_i, DM_i, RD_o);
22
23 // initial CLK = 1'b0; //Inicializamos el reloj
24 // always #5 CLK = ~CLK; //El ciclo del Reloj (cambia cada 5 ns)
25 //
26 initial
27 begin
28
29     rst_i = 1'b0; B_i = 1'b0; DM_i = 4'b0000;
30     #10 rst_i = 1'b1; B_i = 1'b0; DM_i = 4'b0000;
31     #10 rst_i = 1'b0; B_i = 1'b0; DM_i = 4'b0000;
32     #25 B_i = 1'b0; DM_i = 4'b0000;
33     #5 B_i = 1'b1; DM_i = 4'b1111;
34     #25 B_i = 1'b0; DM_i = 4'b1111;
35     #25 B_i = 1'b0; DM_i = 4'b0101;
36     #25 B_i = 1'b1; DM_i = 4'b1010;
37     #5 B_i = 1'b0; DM_i = 4'b1010;
38     #25 B_i = 1'b0; DM_i = 4'b1010;
39     #10 rst_i = 1'b0; B_i = 1'b0; DM_i = 4'b1100;
40     #10 rst_i = 1'b1; B_i = 1'b0; DM_i = 4'b0110;
41     #10 rst_i = 1'b0; B_i = 1'b0; DM_i = 4'b0111;
42     #25;
43 end
44 initial
45     #250 $finish;
46 endmodule
```

Descripción RTL obtenida mediante Vivado 2022.2:



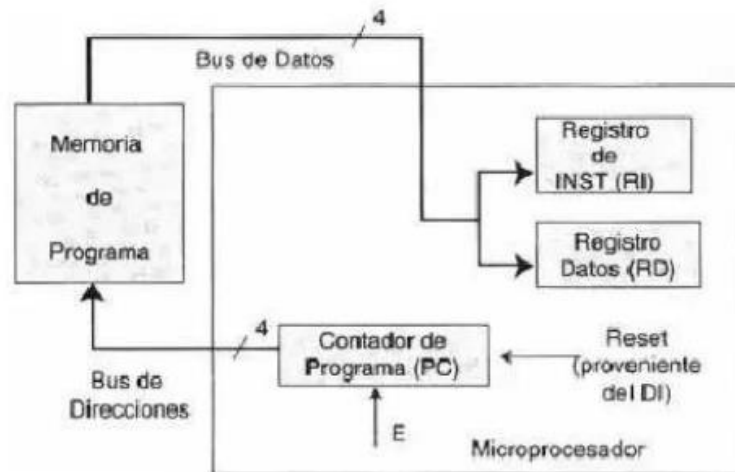
Podemos observar cómo utilizo FlipFlops con su terminal de clk para el control de salida.

Resultado de la simulación:



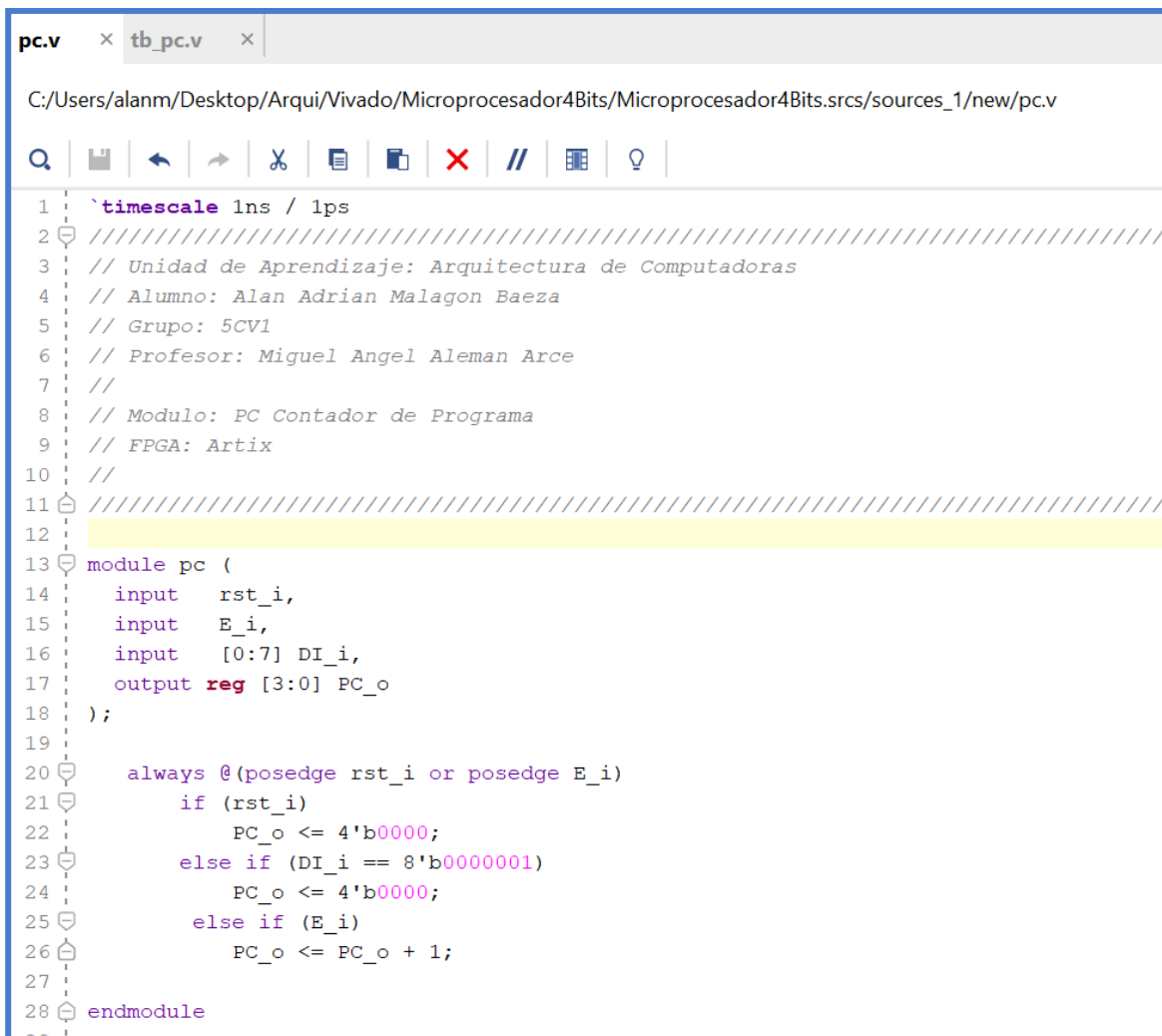
- Programación del contador de programa (PC).

Sin duda un microprocesador requiere periféricos externos que le auxilien en su funcionamiento. En nuestro ejemplo, el contador de miento. En nuestro ejemplo, el contador de programa es un elemento que genera el bus de direcciones, ya sea para direccionar una memoria de programa (ROM), una memoria de datos (RAM) o ambas.



Como puede observarse, el contador se incrementa cada que se genera la señal (E) proveniente del generador ciclo de máquina.

El programa quedará de la siguiente manera:



```
pc.v x tb_pc.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/pc.v

1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: PC Contador de Programa
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | module pc (
14 |     input  rst_i,
15 |     input  E_i,
16 |     input  [0:7] DI_i,
17 |     output reg [3:0] PC_o
18 | );
19 |
20 |     always @(posedge rst_i or posedge E_i)
21 |         if (rst_i)
22 |             PC_o <= 4'b0000;
23 |         else if (DI_i == 8'b0000001)
24 |             PC_o <= 4'b0000;
25 |         else if (E_i)
26 |             PC_o <= PC_o + 1;
27 |
28 | endmodule
```

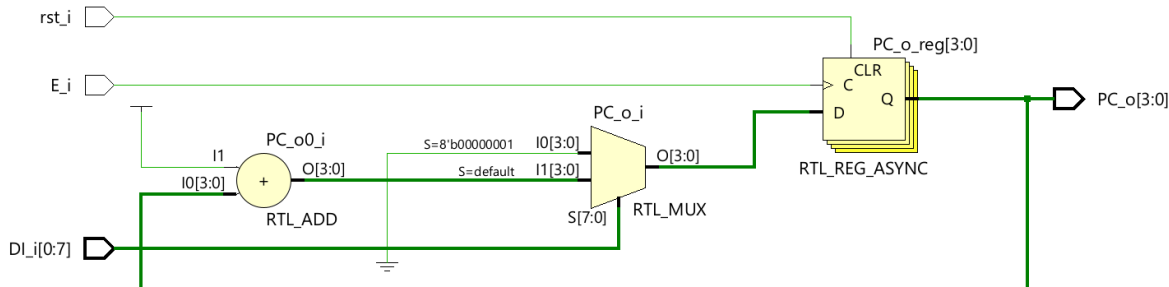
tb_pc.v

C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sim_1/new/tb_pc.v



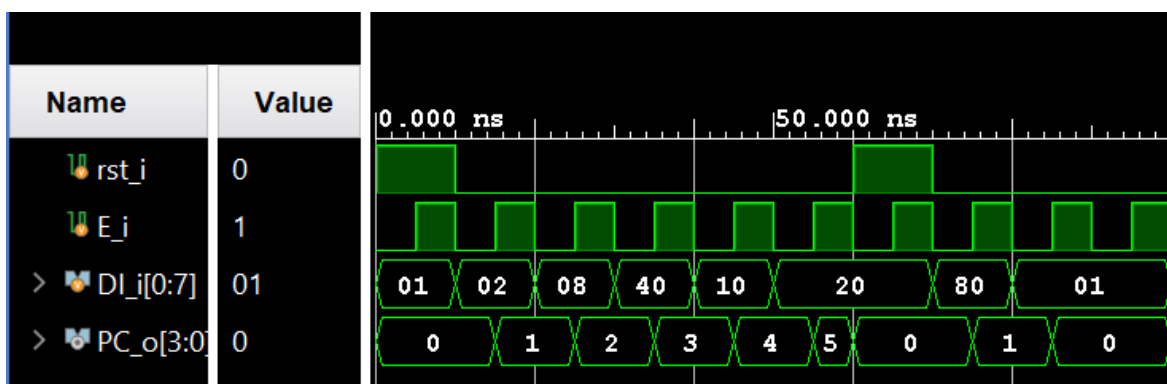
```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: PC Contador de Programa
9  // FPGA: Artix
10 //
11 //////////////////////////////////////
12
13 //Módulo de estímulo para program Counter
14 module tb_pc();
15     reg rst_i, E_i;
16     reg [0:7] DI_i;
17     wire [3:0] PC_o;
18
19     pc UUT(rst_i, E_i, DI_i, PC_o);
20
21     initial E_i = 0;
22
23     always #5 E_i = ~E_i;
24
25     initial
26     begin
27         rst_i=0; DI_i=8'b00000000;
28         rst_i=1; DI_i=8'b00000001; #10
29         rst_i=0; DI_i=8'b00000010; #10
30         rst_i=0; DI_i=8'b00001000; #10
31         rst_i=0; DI_i=8'b01000000; #10
32         rst_i=0; DI_i=8'b00010000; #10
33         rst_i=0; DI_i=8'b00100000; #10
34         rst_i=1; DI_i=8'b00100000; #10
35         rst_i=0; DI_i=8'b10000000; #10
36         rst_i=0; DI_i=8'b00000001; #10
37         #10
38         $finish;
39     end
40
41 endmodule
```

Descripción RTL obtenida mediante Vivado 2022.2:



Podemos observar cómo utilizó un operador de suma y un multiplexor, y para el control del dato a la salida, se implementó mediante flipflops con su terminal de `clk`.

Resultado de la simulación:

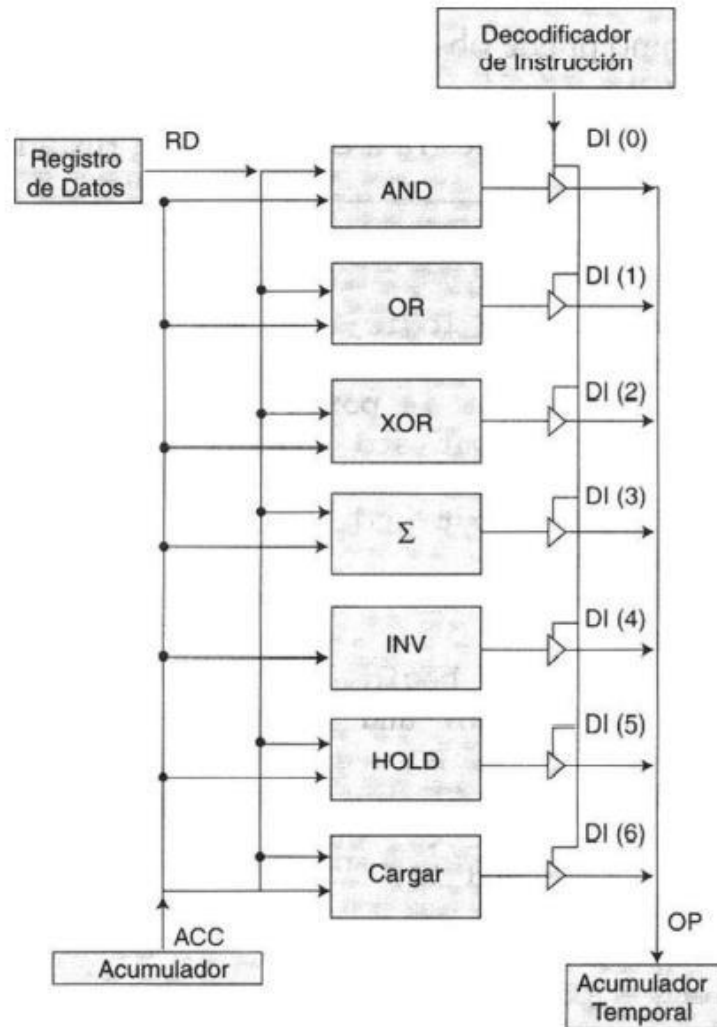


- Programación de la unidad aritmética y lógica (ALU).

Iniciaremos por la definición de las instrucciones que va a ejecutar. Este será nuestro set de instrucciones para el pequeño procesador de 4 bits. (ISA, Instruction Set Architecture)

Código de Operación	Instrucción
0000	AND, el acumulador con el dato inmediato
0001	OR, el acumulador con el dato inmediato
0010	XOR, el acumulador con el dato inmediato
0011	Suma aritmética del acumulador con el dato inmediato
0100	Invertir el acumulador
0101	Retener el dato (hold)
0110	Cargar un dato en el acumulador
0111	Brincar a cero

Tal como su nombre indica, la función nombre indica, la función de este bloque es realizar las operaciones aritméticas y lógicas del microprocesador. Según se aprecia en la figura, la ALU de nuestro ejemplo puede llevar a cabo ocho operaciones, las cuales se refieren por medio de un código de operación de cuatro bits (tabla).



Cabe mencionar que seis de las ocho operaciones que realiza la ALU requieren dos datos para funcionar: uno se almacena con anterioridad en el acumulador y el otro proviene del registro de datos. Observe que la única operación que no requiere dos datos es la función de "invertir", ya que se realiza invirtiendo el contenido del acumulador. Note también que a la salida de cada bloque de operación (and, or, xor, etc.) se encuentra un buffer triestado activo en alto que, con ayuda del decodificador de instrucción, habilita una de las siete salidas correspondientes a cada operación. Observe que el resultado de las operaciones se almacena de nuevo en el acumulador. En este caso utilizamos este caso utilizamos una señal auxiliar llamada OP (operación), la cual guardará temporalmente el resultado de dicha operación y luego lo canalizará al acumulador temporal.

El programa quedará de la siguiente manera:

```
alu.v x tb_alu.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sources_1/new/alu.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: ALU Unidad Logica y Aritmetica
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module alu (
14     input [0:7] DI_i,
15     input [3:0] RD_i,
16     input [3:0] ACC_i,
17     output reg [3:0] OP_o
18 );
19
20 parameter AND = 8'b10000000, OR = 8'b01000000, XOR = 8'b00100000, SUM = 8'b00010000;
21 parameter INV = 8'b00001000, HOLD = 8'b00000100;
22
23 //Opcion if
24 always @(DI_i, ACC_i, RD_i)
25 begin
26     if (DI_i == AND)
27         OP_o = ACC_i & RD_i; //AND
28     else if (DI_i == OR)
29         OP_o = ACC_i | RD_i; // OR
30     else if (DI_i == XOR)
31         OP_o = ACC_i ^ RD_i; //XOR
32     else if (DI_i == SUM)
33         OP_o = ACC_i + RD_i; // SUMA
34     else if (DI_i == INV)
35         OP_o = ~ACC_i; // INVIERTE
36     else if (DI_i == HOLD)
37         OP_o = ACC_i; // HOLD
38     else
39         OP_o = RD_i; // CARGA ACUMULADOR
40 end
41
42 endmodule
```

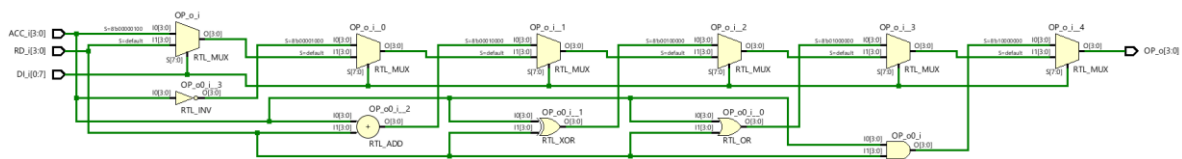
```

alu.v x tb_alu.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sim_1/new/tb_alu.v

1 `timescale 1ns / 1ps
2 //////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: ALU Unidad Logica y Aritmetica
9 // FPGA: Artix
10 //
11 //////////////////////////////////////
12
13 //Módulo de Estimulo para la alu
14
15 module tb_alu();
16     reg [0:7] DI_i;
17     reg [3:0] RD_i, ACC_i;
18     wire [3:0] OP_o;
19
20     alu alu1(DI_i, RD_i, ACC_i, OP_o);
21
22     initial
23     begin
24         DI_i = 8'b00000000; RD_i = 4'b0000; ACC_i = 4'b0000;
25         #25 DI_i = 8'b10000000; RD_i = 4'b1010; ACC_i = 4'b0101; //AND
26         #25 DI_i = 8'b01000000; RD_i = 4'b1010; ACC_i = 4'b0101; //OR
27         #25 DI_i = 8'b00100000; RD_i = 4'b1010; ACC_i = 4'b1010; //XOR
28         #25 DI_i = 8'b00010000; RD_i = 4'b0001; ACC_i = 4'b0001; //ADD
29         #25 DI_i = 8'b00001000; RD_i = 4'b0000; ACC_i = 4'b0000; //INV
30         #25 DI_i = 8'b00000100; RD_i = 4'b0000; ACC_i = 4'b0000; //HOLD
31         #25 DI_i = 8'b00000010; RD_i = 4'b1000; ACC_i = 4'b0000; //CARGA
32         #25 DI_i = 8'b00000001; RD_i = 4'b0011; ACC_i = 4'b0000; // CARGA
33     #25;
34     end
35     initial
36     #250 $finish;
37 endmodule

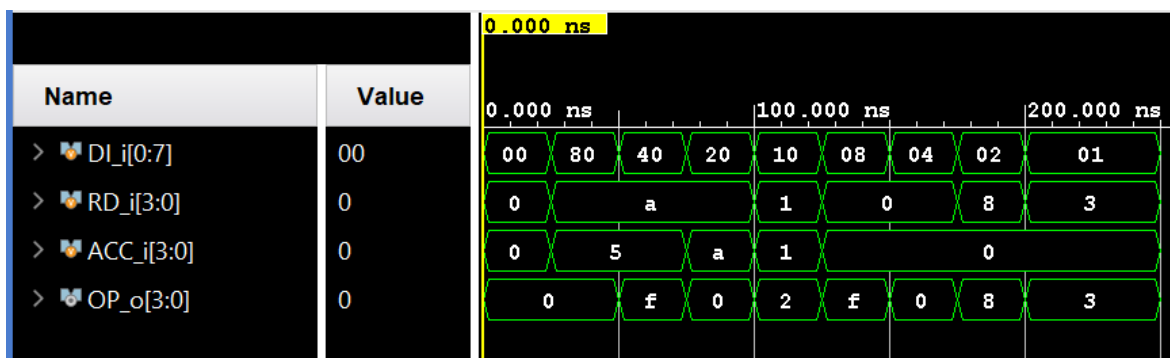
```

Descripción RTL obtenida mediante Vivado 2022.2:



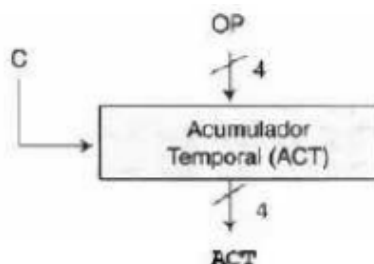
Podemos observar cómo utilizo multiplexores, operador inversor, de suma, xor, or y and, y para el control del dato a la salida, se implementó mediante un multiplexor.

Resultado de la simulación:

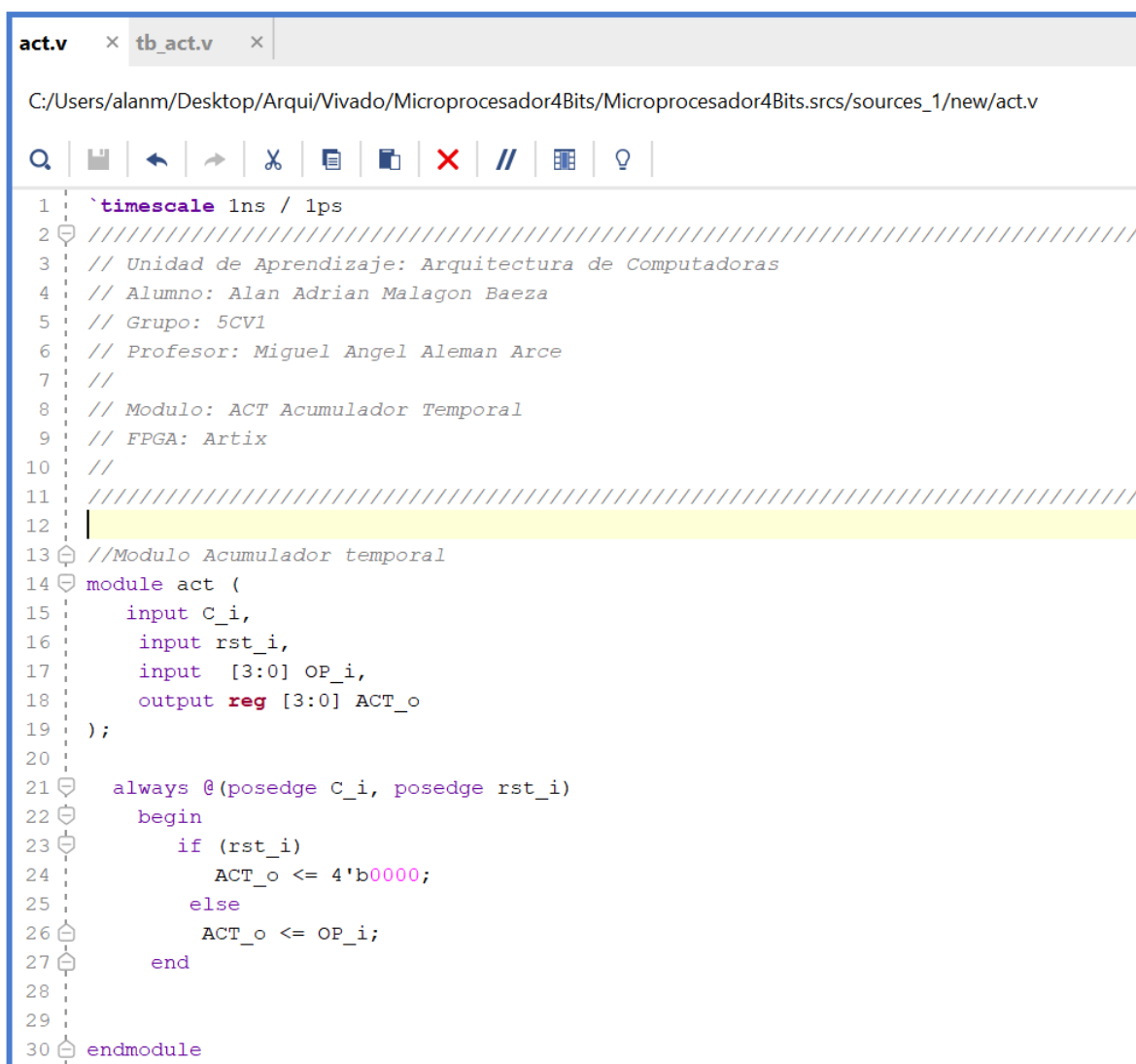


- **Programación del acumulador temporal (ACT).**

La función de este módulo es almacenar temporalmente el resultado proveniente de la ALU (OP) y después canalizarlo por medio de su salida (ACT) al acumulador permanente; este dato se almacena mediante su señal de habilitación (C) proveniente del generador ciclo de máquina.



El programa quedará de la siguiente manera:



```
act.v x tb_act.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/act.v

1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: ACT Acumulador Temporal
9  // FPGA: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 //Modulo Acumulador temporal
14 module act (
15     input C_i,
16     input rst_i,
17     input [3:0] OP_i,
18     output reg [3:0] ACT_o
19 );
20
21 always @(posedge C_i, posedge rst_i)
22 begin
23     if (rst_i)
24         ACT_o <= 4'b0000;
25     else
26         ACT_o <= OP_i;
27     end
28
29
30 endmodule
```

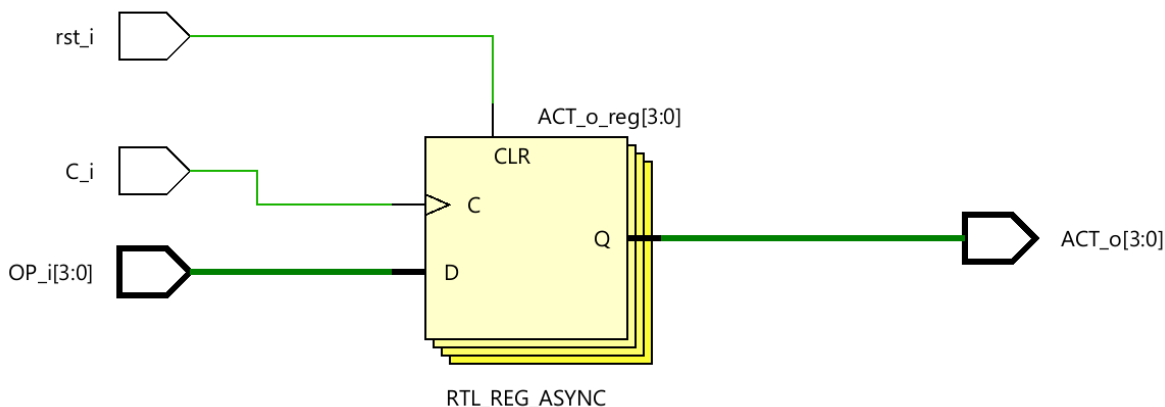
tb_act.v

C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sim_1/new/tb_act.v



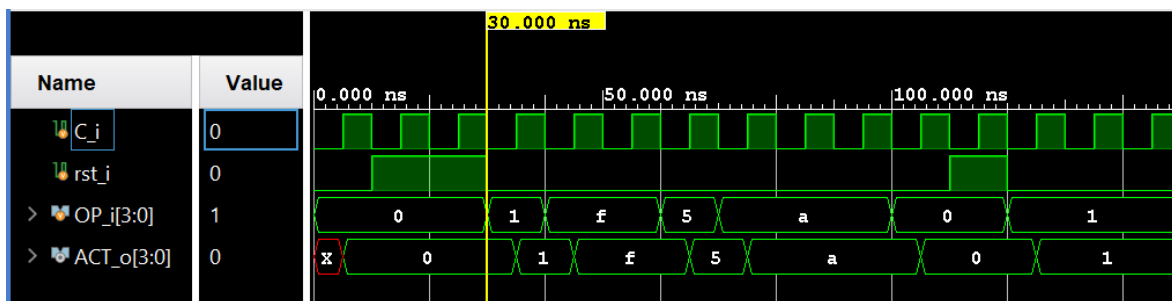
```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: ACT Acumulador Temporal
9  // FPGA: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12 //Módulo de Estimulo para el acumulador temporal
13 module tb_act ();
14     reg      C_i;
15     reg      rst_i;
16     reg [3:0] OP_i;
17     wire [3:0] ACT_o;
18
19     act UUT(C_i, rst_i, OP_i, ACT_o);
20     initial
21     begin
22         C_i = 1'b0; //Inicializamos el reloj
23         rst_i = 1'b0;
24         OP_i = 4'b0000;
25     end
26     always #5 C_i = ~C_i; //El ciclo del Reloj (cambia cada 5 nS)
27     initial
28     begin
29         rst_i = 1'b0; OP_i = 4'b0000;
30         #10 rst_i = 1'b1; OP_i = 4'b0000;
31         #10 rst_i = 1'b1; OP_i = 4'b0000;
32         #10 rst_i = 1'b0; OP_i = 4'b0001;
33         #10 rst_i = 1'b0; OP_i = 4'b1111;
34         #10 rst_i = 1'b0; OP_i = 4'b1111;
35         #10 rst_i = 1'b0; OP_i = 4'b0101;
36         #10 rst_i = 1'b0; OP_i = 4'b1010;
37         #10 rst_i = 1'b0; OP_i = 4'b1010;
38         #10 rst_i = 1'b0; OP_i = 4'b1010;
39         #10 rst_i = 1'b0; OP_i = 4'b0000;
40         #10 rst_i = 1'b1; OP_i = 4'b0000;
41         #10 rst_i = 1'b0; OP_i = 4'b0001;
42         #10 rst_i = 1'b0; OP_i = 4'b0001;
43         #10;
44     end
45     initial
46     #150 $finish;
47 endmodule
```

Descripción RTL obtenida mediante Vivado 2022.2:



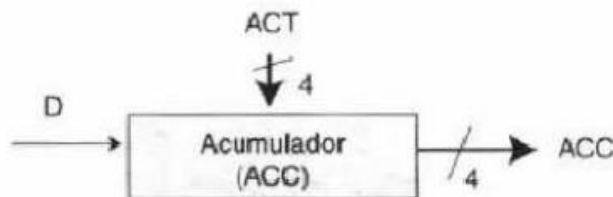
Podemos observar flipflops con su terminal de clk para el control del dato a la salida.

Resultado de la simulación:



- **Programación del acumulador permanente (ACC).**

La función de este módulo es almacenar el resultado final de la última operación realizada por la ALU, ya sea para enviarlo como aplicación externa o retroalimentar al microprocesador. Este dato se almacena mediante su señal de habilitación (D) proveniente del generador de ciclo de máquina.



El programa quedará de la siguiente manera:

```
acc.v

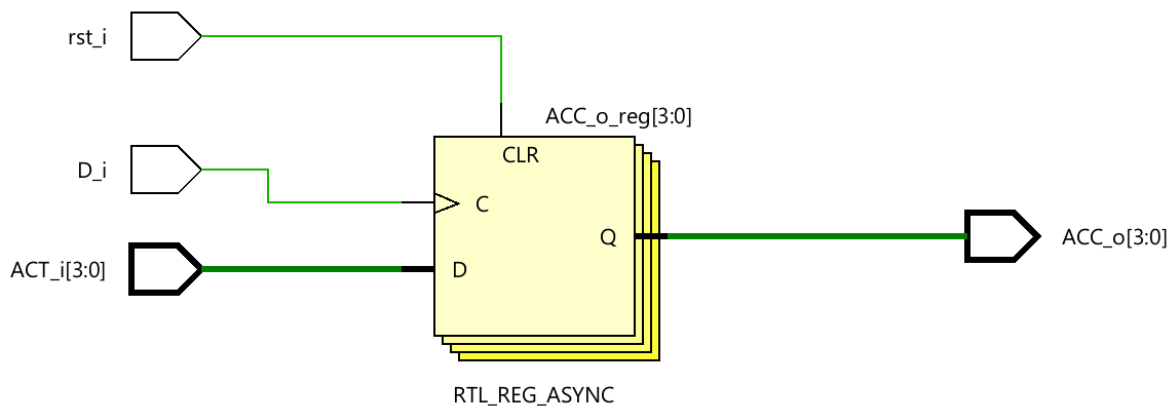
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/acc.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Modulo: ACC Acumulador Permanente
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 //Modulo Acumulador Final
14 module acc (
15     input D_i,
16     input rst_i,
17     input [3:0] ACT_i,
18     output reg [3:0] ACC_o
19 );
20
21 always @(posedge D_i or posedge rst_i)
22 begin
23     if (rst_i)
24         ACC_o <= 4'b0000;
25     else
26         ACC_o <= ACT_i;
27     end
28
29 endmodule
```

```
acc.v x tb_acc.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sim_1/new/tb_acc.v

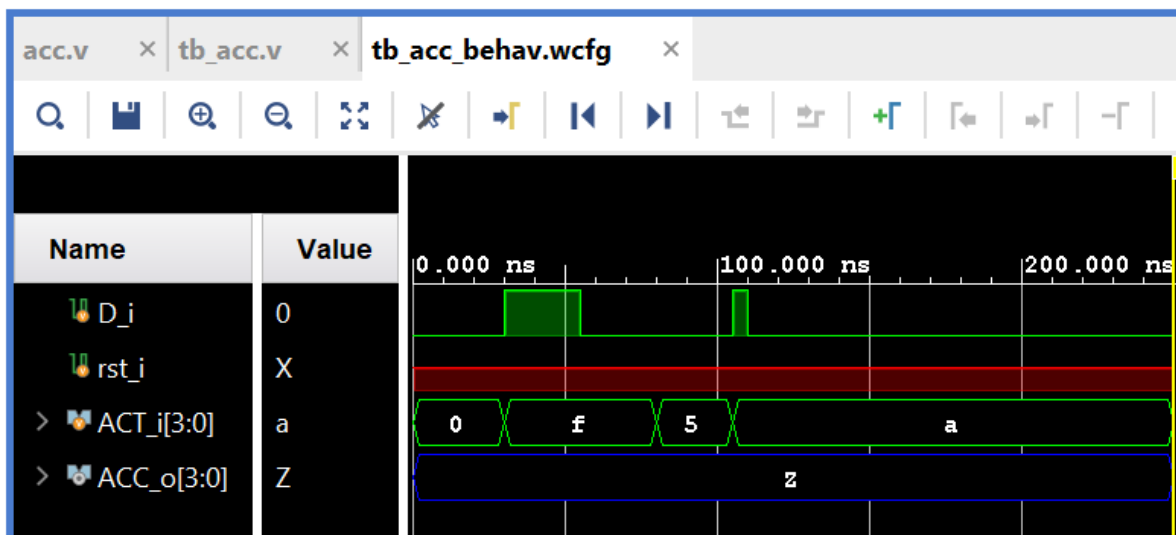
1 timescale 1ns / 1ps
2 //////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Modulo: ACC Acumulador Permanente
9 // FPGA: Artix
10 //
11 //////////////////////////////////////
12
13 //Módulo de Estimulo Acumulador Final
14
15 module tb_acc ();
16     reg      D_i;
17     reg      rst_i;
18     reg [3:0] ACT_i;
19     wire [3:0] ACC_o;
20
21     acc ACC1(D_i, ACT_i, ACC_o);
22
23 // initial CLK = 1'b0; //Inicializamos el reloj
24 // always #5 CLK = ~CLK; //El ciclo del Reloj (cambia cada 5 nS)
25 //
26 initial
27 begin
28
29     D_i = 1'b0; ACT_i = 4'b0000;
30     #25 D_i = 1'b0; ACT_i = 4'b0000;
31     #5 D_i = 1'b1; ACT_i = 4'b1111;
32     #25 D_i = 1'b0; ACT_i = 4'b1111;
33     #25 D_i = 1'b0; ACT_i = 4'b0101;
34     #25 D_i = 1'b1; ACT_i = 4'b1010;
35     #5 D_i = 1'b0; ACT_i = 4'b1010;
36     #25 D_i = 1'b0; ACT_i = 4'b1010;
37     #25;
38 end
39 initial
40 #250 $finish;
41 endmodule
```

Descripción RTL obtenida mediante Vivado 2022.2:



Podemos observar flipflops con su terminal de clk para el control del dato a la salida.

Resultado de la simulación:



- Programación de la memoria de programa ROM

El programa quedará de la siguiente manera:

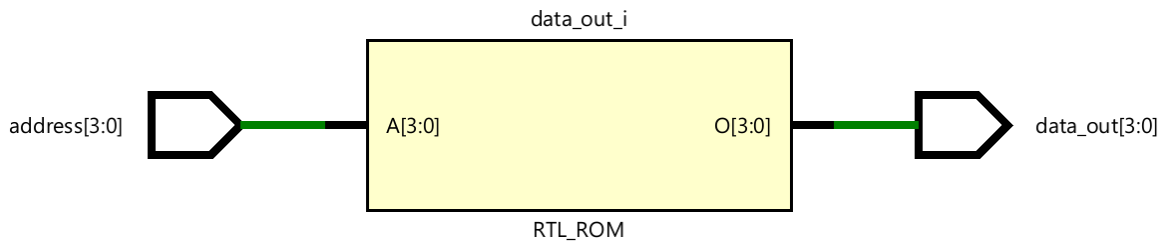
```
rom.v
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/rom.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Create Date: 25.03.2023 20:00:34
9  // Design Name: rom
10 // Module Name: rom
11 // Project Name: MemoriaROM
12 // Target Devices: Artix
13 // Description: Version 1
14 // Una declaración de caso para definir el contenido de cada ubicación en la
15 // memoria en función de la dirección entrante
16 //
17 //////////////////////////////////////////////////
18
19 // Example: Behavioral Model of a 4x4 Synchronous Read Only Memory in Verilog
20 // Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog,
21 // Springer, 1st Edition, USA, 2017. pp 353
22
23 module rom (
24     input  [3:0] address,
25     output reg  [3:0] data_out
26 );
27
28 always @(address)
29     case (address)
30         4'b0000 : data_out = 4'b0000;
31         4'b0001 : data_out = 4'b1110;
32         4'b0010 : data_out = 4'b0010;
33         4'b0011 : data_out = 4'b0100;
34         4'b0100 : data_out = 4'b1101;
35         4'b0101 : data_out = 4'b0110;
36         default : data_out = 4'bXXXX;
37     endcase
38
39 endmodule
```

```
rom.v x tb_rom.v x Untitled 27 x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sim_1/new/tb_rom.v

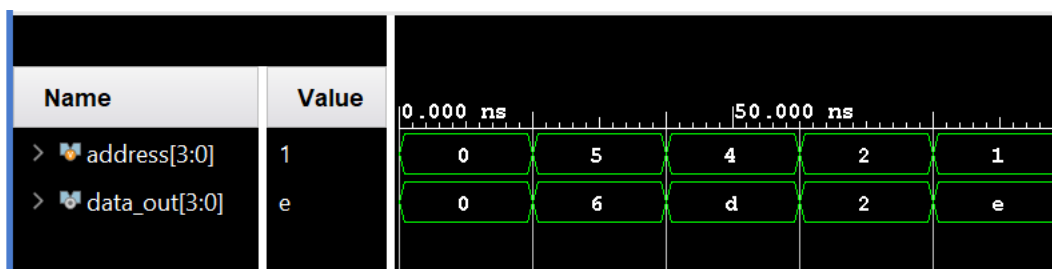
1 timescale 1ns / 1ps
2 //////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 6CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Create Date: 25.03.2023 20:00:34
9 // Design Name: rom
10 // Module Name: rom
11 // Project Name: MemoriaROM
12 // Target Devices: Artix
13 // Description: Version 1
14 // Una declaración de caso para definir el contenido de cada ubicación en la
15 // memoria en función de la dirección entrante
16 //
17 //////////////////////////////////////
18
19 //Modulo de Estimulo
20 module tb_rom ();
21
22 reg [3:0] address;
23
24 wire [3:0] data_out;
25
26 rom uut(address, data_out);
27
28
29 initial
30 begin
31     address = 4'b0000;
32     #20
33     address = 4'b0101;
34     #20
35     address = 4'b0100;
36     #20
37     address = 4'b0010;
38     #20
39     address = 4'b0001;
40     #20;
41 end
42
43 initial #100 $finish;
44
45 endmodule
```


Descripción RTL obtenida mediante Vivado 2022.2:



Como se puede observar se utilizó un bloque de memoria ROM del FPGA seleccionado.

Resultado de la simulación:



Como se observa en la simulación, los resultados son los almacenados en la memoria de acuerdo con la localidad.

Programación del microprocesador de 4 bits

El programa quedará de la siguiente manera:

```
mp4b.v

C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sources_1/new/mp4b.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Microprocesador de 4 bits
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module mp4b (
14
15     input rst,
16     input clk,
17
18     input [3:0] DM,
19
20     output A_out,
21     output B_out,
22     output C_out,
23     output D_out,
24     output E_out,
25
26     output [0:7] DI_out,
27     output [3:0] RI_out,
28     output [3:0] RD_out,
29     output [3:0] ACC_out,
30     output [3:0] ACT_out,
31     output [3:0] OP_out,
32     output [3:0] Q_out,
33     output [3:0] PC_o
34 );
35
36 wire [3:0] RD, RI, ACC, ACT, OP, Q, PC;
37 wire A, B, C, D, E;
38 wire [0:7] DI;
```

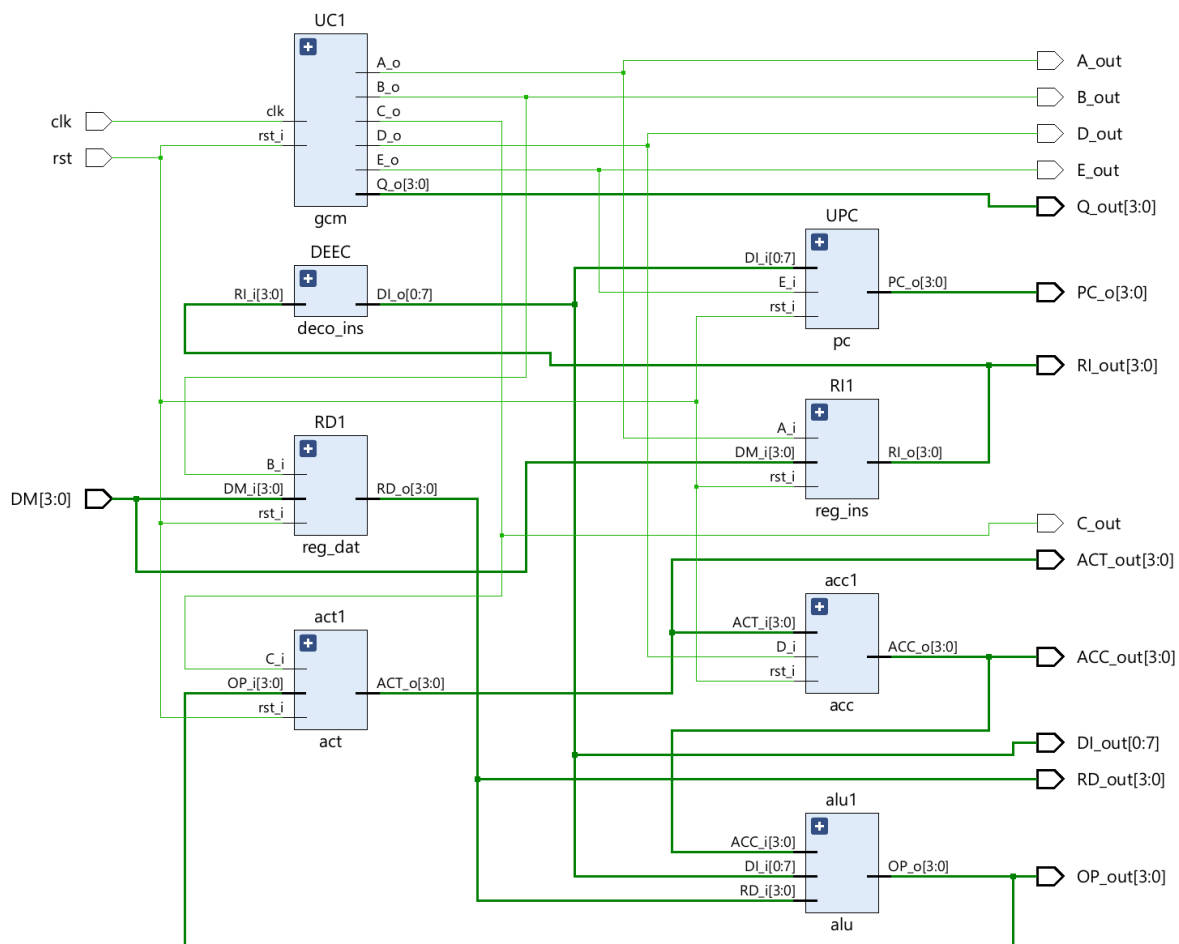


```

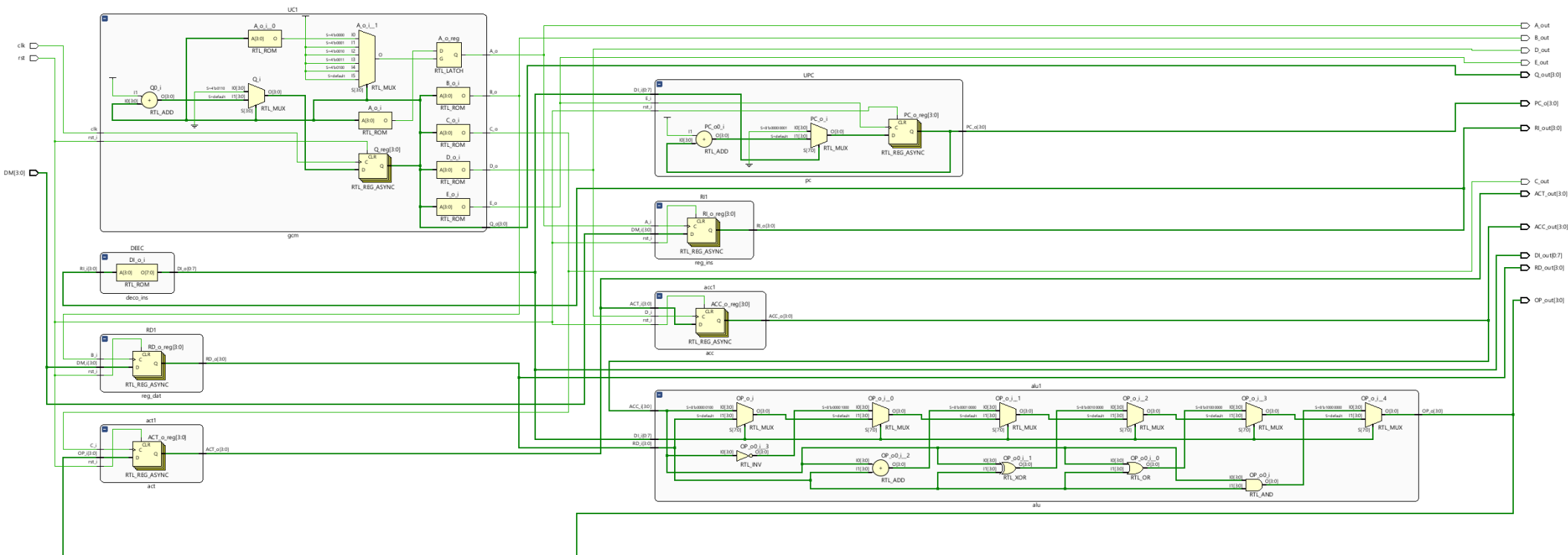
35 |
36 |     initial clk = 1'b0; //Inicializamos el reloj
37 |     always #5 clk = ~clk; //El ciclo del Reloj (cambia cada 5 nS)
38 | //
39 | initial
40 |     begin
41 |
42 |         rst=1'b0; DM=4'b0000;
43 |         #10 rst=1'b1; DM=4'b0001;
44 |         #10 rst=1'b0; DM=4'b0010;
45 |         #10 rst=1'b0; DM=4'b0100;
46 |         #10 rst=1'b0; DM=4'b0000;
47 |         #10 rst=1'b0; DM=4'b0100;
48 |         #10 rst=1'b0; DM=4'b0010;
49 |         #10 rst=1'b0; DM=4'b0100;
50 |         #10 rst=1'b0; DM=4'b0010;
51 |         #10 rst=1'b0; DM=4'b0100;
52 |         #10 rst=1'b0; DM=4'b0010;
53 |         #10 rst=1'b0; DM=4'b0100;
54 |         #10 rst=1'b0; DM=4'b0010;
55 |         #10 rst=1'b0; DM=4'b0100;
56 |         #10 rst=1'b0; DM=4'b0010;
57 |         #25;
58 |     end
59 |     initial
60 |         #250 $finish;
61 | endmodule

```

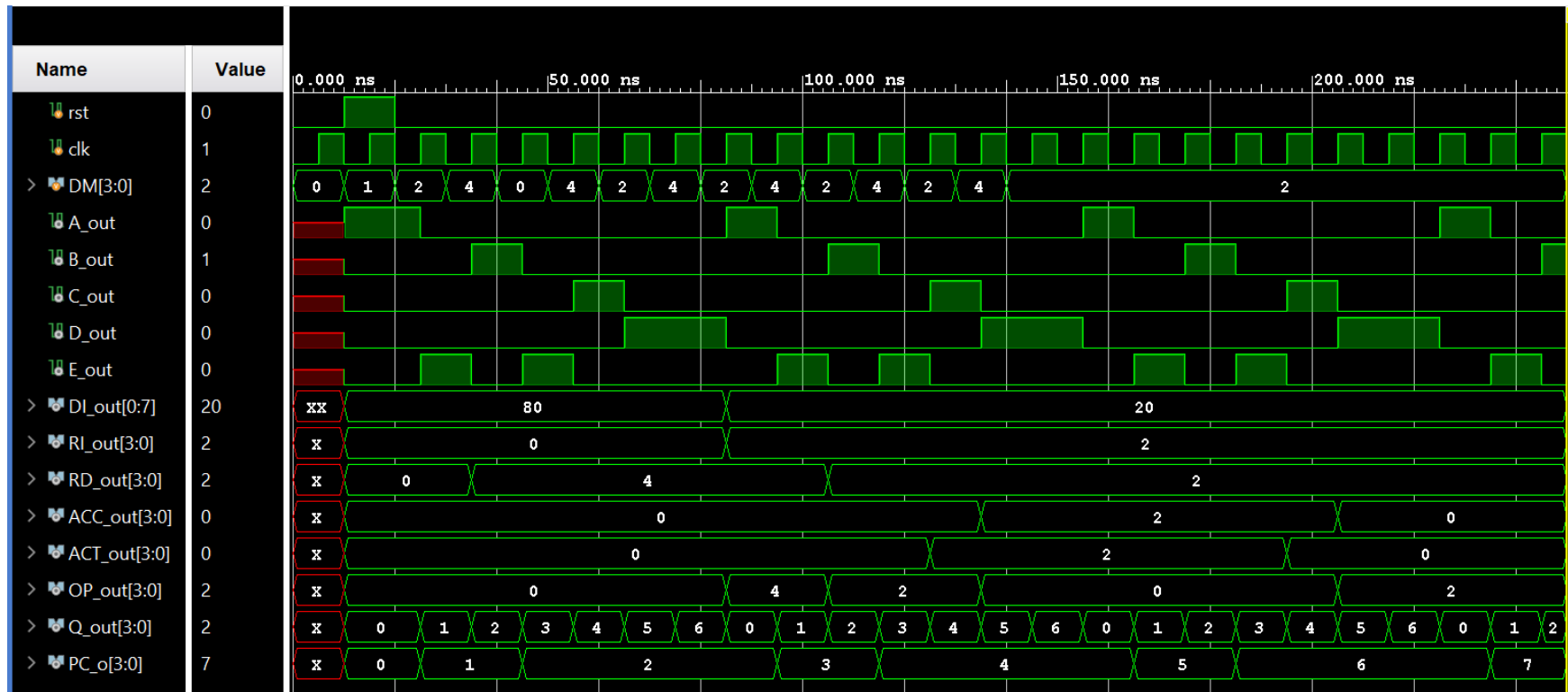
Descripción RTL obtenida mediante Vivado 2022.2:



Podemos como utilizó todos los módulos anteriormente descritos (GCM, DI, PC, RD, RI, ACT, ACC y ALU) con su terminal de `clk` y `rst`.



Resultado de la simulación:



Programación del microprocesador de 4 bits con ROM

El programa quedará de la siguiente manera:

```
mp4b.v
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/mp4b.v

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Microprocesador de 4 bits
9  // FPGA: Artix
10 //
11 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12
13 module mp4b (
14
15     input rst,
16     input clk,
17
18     output [3:0] DM_out,
19
20     output A_out,
21     output B_out,
22     output C_out,
23     output D_out,
24     output E_out,
25
26     output [0:7] DI_out,
27     output [3:0] RI_out,
28     output [3:0] RD_out,
29     output [3:0] ACC_out,
30     output [3:0] ACT_out,
31     output [3:0] OP_out,
32     output [3:0] Q_out,
33     output [3:0] PC_o
34 );
35
36 wire [3:0] RD, RI, ACC, ACT, OP, Q, PC, DM;
37 wire A, B, C, D, E;
38 wire [0:7] DI;
39
```



```

40 : acc acc1 (.D_i(D), .rst_i(rst), .ACT_i(ACT), .ACC_o(ACC));
41 : act act1 (.C_i(C), .rst_i(rst), .OP_i(OP), .ACT_o(ACT));
42 : alu alu1 (.DI_i(DI), .RD_i(RD), .ACC_i(ACC), .OP_o(OP));
43 : reg_dat RD1 (.B_i(B), .rst_i(rst), .DM_i(DM), .RD_o(RD));
44 : reg_ins RI1 (.A_i(A), .rst_i(rst), .DM_i(DM), .RI_o(RI));
45 : deco_ins DEEC (.RI_i(RI), .DI_o(DI));
46 : gcm UC1 (.clk(clk), .rst_i(rst), .A_o(A), .B_o(B), .C_o(C), .D_o(D), .E_o(E), .Q_o(Q));
47 : pc UPC (.rst_i(rst), .E_i(E), .DI_i(DI), .PC_o(PC));
48 : rom MEM (.address(PC), .data_out(DM));
49 :
50 : assign DM_out = DM;
51 : assign DI_out = DI;
52 : assign RI_out = RI;
53 : assign RD_out = RD;
54 : assign ACC_out = ACC;
55 : assign ACT_out = ACT;
56 : assign OP_out = OP;
57 : assign A_out = A;
58 : assign B_out = B;
59 : assign C_out = C;
60 : assign D_out = D;
61 : assign E_out = E;
62 : assign Q_out = Q;
63 : assign PC_o = PC;
64 :
65 : endmodule

```

```
mp4b.v x tb_mp4b.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sim_1/new/tb_mp4b.v

1 `timescale 1ns / 1ps
2 //////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Microprocesador de 4 bits
9 // FPGA: Artix
10 //
11 //////////////////////////////////////////////////
12
13 //Módulo de Estimulo para m4b (Microprocesador de 4 bits)
14
15 module tb_m4b ();
16     reg rst;
17     reg clk;
18
19     wire [3:0] DM;
20
21     wire A_out;
22     wire B_out;
23     wire C_out;
24     wire D_out;
25     wire E_out;
26
27     wire [0:7] DI_out;
28     wire [3:0] RI_out;
29     wire [3:0] RD_out;
30     wire [3:0] ACC_out;
31     wire [3:0] ACT_out;
32     wire [3:0] OP_out;
33     wire [3:0] Q_out;
34     wire [3:0] PC_o;
35
36     mp4b m4b1(rst, clk, DM, A_out, B_out, C_out, D_out, E_out, DI_out, RI_out, RD_out, ACC_out, ACT_out, OP_out, Q_out, PC_o);
37
38     initial clk = 1'b0; //Inicializamos el reloj
39     always #5 clk = ~clk; //El ciclo del Reloj (cambia cada 5 nS)
40
41     initial
42     begin
43
44         rst=1'b0; //DM=4'b0000;
45         #10 rst=1'b1; //DM=4'b0001;
46         #10 rst=1'b0; //DM=4'b0010;
47         #10 rst=1'b0; //DM=4'b0100;
48         #10 rst=1'b0; //DM=4'b0000;
49         #10 rst=1'b0; //DM=4'b0100;
50         #10 rst=1'b0; //DM=4'b0010;
51         #10 rst=1'b0; //DM=4'b0100;
52         #10 rst=1'b1; //DM=4'b0010;
53         #10 rst=1'b0; //DM=4'b0100;
54         #10 rst=1'b0; //DM=4'b0010;
55         #10 rst=1'b0; //DM=4'b0100;
56         #10 rst=1'b0; //DM=4'b0010;
57         #10 rst=1'b0; //DM=4'b0100;
58         #10 rst=1'b0; //DM=4'b0010;
59         #25;
60     end
61     initial
62     #250 $finish;
63 endmodule
```

Se agregaron nuevas operaciones para la ALU y el DI : NOR, NAND XNOR, MAYOR QUE, MENOR QUE, IGUAL, Y RESTA.

```

deco_ins.v x alu.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srscs/sources_1/new/alu.v

1      `timescale 1ns / 1ps
2      //////////////////////////////////////////////////
3      // Unidad de Aprendizaje: Arquitectura de Computadoras
4      // Alumno: Alan Adrian Malagon Baeza
5      // Grupo: 5CV1
6      // Profesor: Miguel Angel Aleman Arce
7      //
8      // Modulo: ALU Unidad Logica y Aritmetica
9      // FPGA: Artix
10     //
11     //////////////////////////////////
12
13     module alu (
14         input [0:7] DI_i,
15         input [3:0] RD_i,
16         input [3:0] ACC_i,
17         output reg [3:0] OP_o
18     );
19
20     parameter AND = 8'b1000000, OR = 8'b0100000, XOR = 8'b0010000;
21     parameter NOR = 8'b0001000, NAND = 8'b0000100, XNOR = 8'b00000100;
22     parameter MAYOR = 8'b00000010, MENOR = 8'b00000011, IGUAL = 8'b1100000;
23     parameter SUM = 8'b10100000, RES = 8'b10010000;
24     parameter INV = 8'b11100000, HOLD = 8'b11010000;
25
26     //Opcion if
27     always @(DI_i, ACC_i, RD_i)
28     begin
29         if (DI_i == AND)
30             OP_o = ACC_i & RD_i; //AND
31         else if (DI_i == OR)
32             OP_o = ACC_i | RD_i; // OR
33         else if (DI_i == XOR)
34             OP_o = ACC_i ^ RD_i; //XOR
35         else if (DI_i == NOR)
36             OP_o = ~(ACC_i|RD_i); //NOR
37         else if (DI_i == NAND)
38             OP_o = ~(ACC_i&RD_i); //NAND
39         else if (DI_i == XNOR)
40             OP_o = ~(ACC_i ^ RD_i); //XNOR
41         else if (DI_i == MAYOR)
42             OP_o = (ACC_i>RD_i)?8'd1:8'd0; //MAYOR
43         else if (DI_i == MENOR)
44             OP_o = (ACC_i<RD_i)?8'd1:8'd0; //MENOR
45         else if (DI_i == IGUAL)
46             OP_o = (ACC_i==RD_i)?8'd1:8'd0; //IGUAL

```

```

47 |         else if (DI_i == SUM)
48 |             OP_o = ACC_i + RD_i; // SUMA
49 |         else if (DI_i == RES)
50 |             OP_o = ACC_i - RD_i; // RESTA
51 |         else if (DI_i == INV)
52 |             OP_o = ~ACC_i; // INVIERTE
53 |         else if (DI_i == HOLD)
54 |             OP_o = ACC_i; // HOLD
55 |         else
56 |             OP_o = RD_i; // CARGA ACUMULADOR
57 |     end
58 | endmodule

```

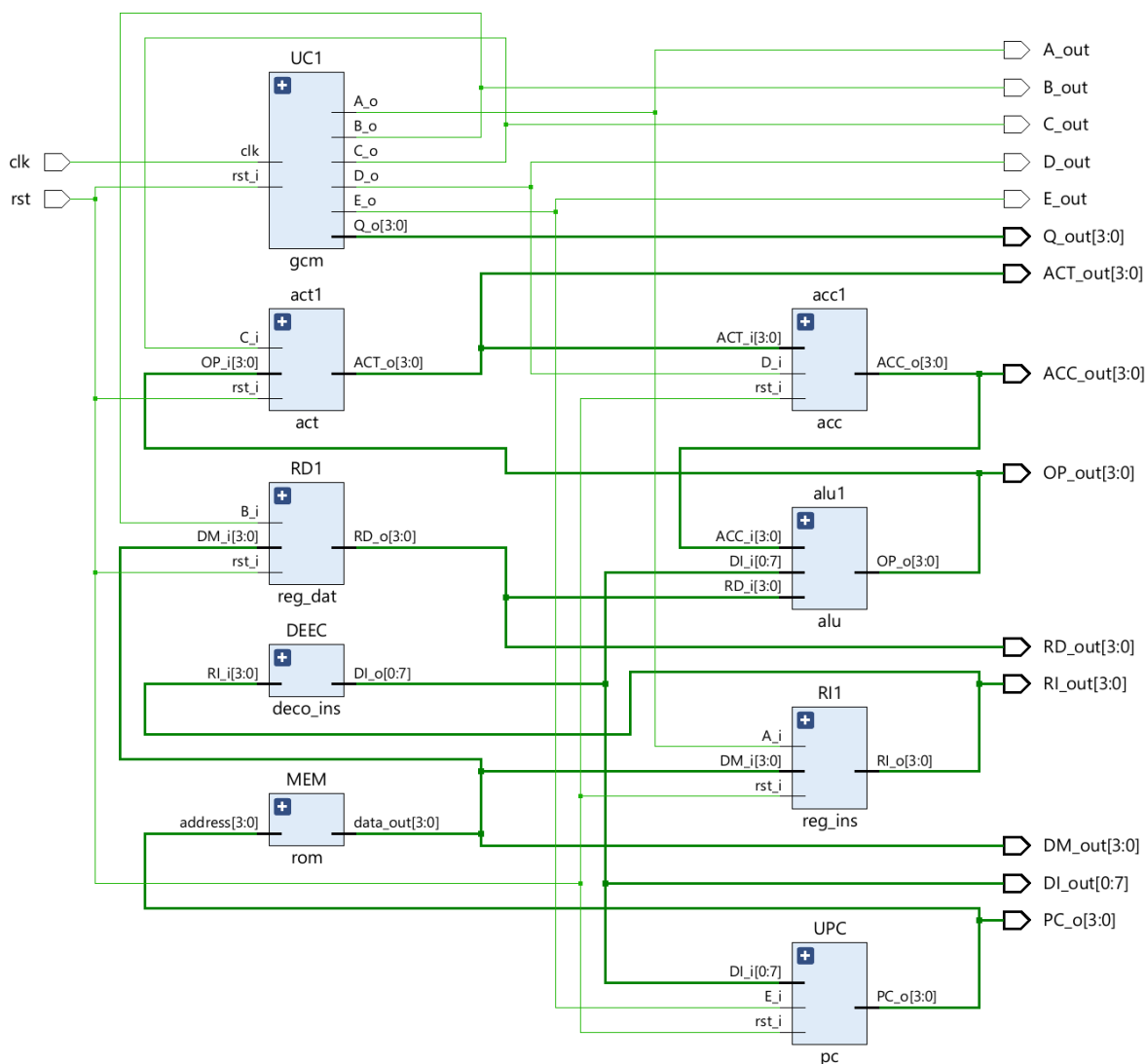
```

deco_ins.v x alu.v x
C:/Users/alanm/Desktop/Arqui/Vivado/Microprocesador4Bits/Microprocesador4Bits.srcs/sources_1/new/deco_ins.v

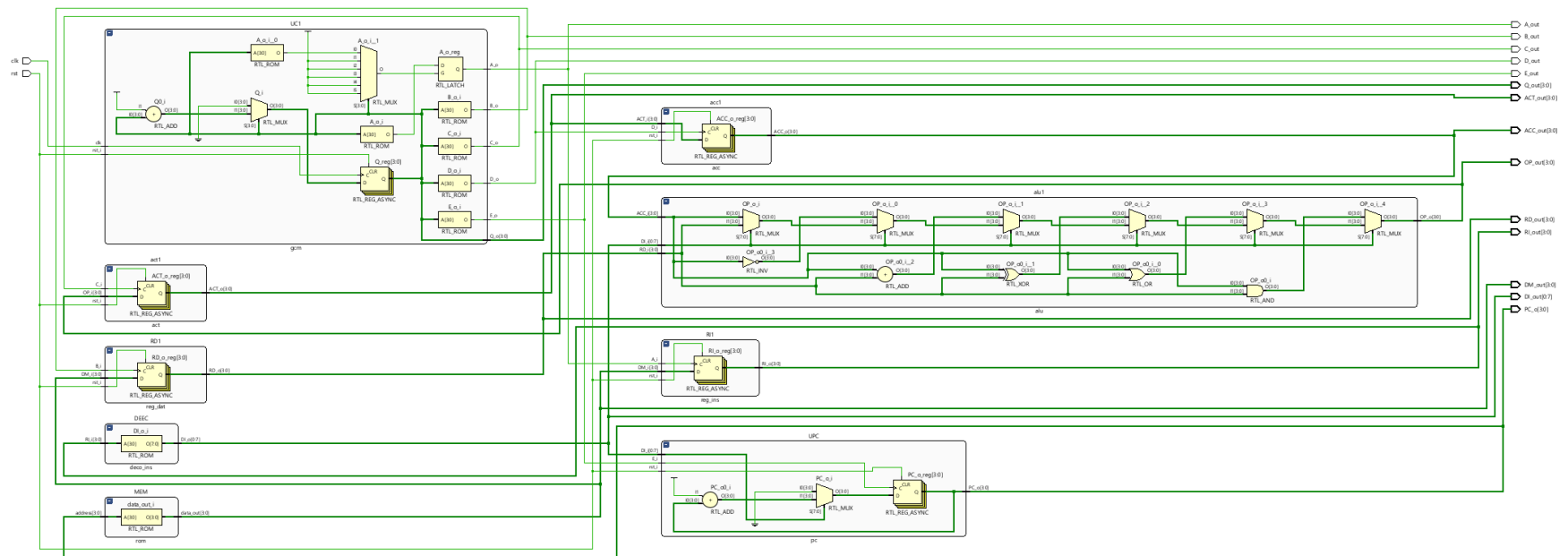
1 | timescale 1ns / 1ps
2 | //////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Modulo: DI Decodificador de instrucción (DI)
9 | // FPGA: Artix
10 | //
11 | //////////////////////////////////////////////////
12 |
13 | module deco_ins(
14 |     input [3:0] RI_i,
15 |     output reg [0:7] DI_o
16 | );
17 |
18 | parameter AND=4'b0000, OR=4'b0001, XOR=4'b0010, SUMA=4'b0011, INV=4'b0100, HOLD=4'b0101, LOAD=4'b0110, RST=4'b0111, UNAB=4'b1000;
19 | parameter NOR=4'b1001, NAND=4'b1010, XNOR=4'b1011, MAYOR=4'b1100, MENOR=4'b1101, IGUAL=4'b1110, RES=4'b1111;
20 | always @(RI_i)
21 |     case (RI_i)
22 |         AND : DI_o = 8'b10000000;
23 |         OR  : DI_o = 8'b01000000;
24 |         XOR : DI_o = 8'b00100000;
25 |         NOR : DI_o = 8'b00010000;
26 |         NAND : DI_o = 8'b00001000;
27 |         XNOR : DI_o = 8'b00000100;
28 |         MAYOR : DI_o = 8'b00000010;
29 |         MENOR : DI_o = 8'b00000011;
30 |         IGUAL : DI_o = 8'b11000000;
31 |         SUMA : DI_o = 8'b10100000;
32 |         RES : DI_o = 8'b00100000;
33 |         INV : DI_o = 8'b11100000;
34 |         HOLD : DI_o = 8'b11010000;
35 |         LOAD : DI_o = 8'b11111111;
36 |         RST : DI_o = 8'b00000001;
37 |         default: DI_o = 8'b00000000; //Deshabilita DI
38 |     endcase
39 |
40 | endmodule

```

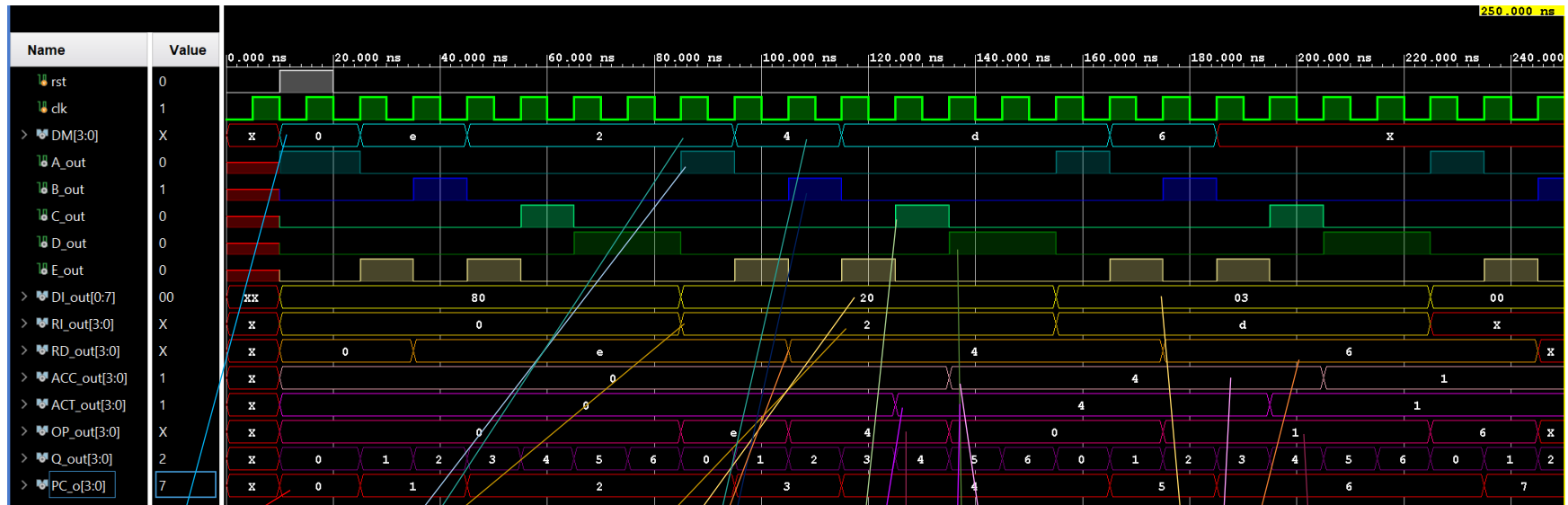
Descripción RTL obtenida mediante Vivado 2022.2:



Podemos como utilizó todos los módulos anteriormente descritos (GCM, DI, PC, ROM, RD, RI, ACT, ACC y ALU) con su terminal de clk y rst.



Resultado de la simulación:



Cuando
PC=0 -> DM=0
PC=1 -> DM=e
PC=2 -> DM=2
PC=3 -> DM=4
PC=4 -> DM=d
PC=5 -> DM=6
PC=6 y 7 -> DM=X
porque no están
asignados en rom

Cuando A está en
flanco de subida
RI=DM

Cuando
Suma: RI=0 -> DI=80
XOR: RI=2 -> DI=20
MENOR: RI=d -> DI= 3

Cuando B esta en
flanco de subida
RD=DM

Cuando C está
en flanco de
subida ACT=OP

Cuando D está
en flanco de
subida ACT=ACC

DI=03 -> MENOR QUE
ACC=4
RD=6
OP = (4<6)? 1:0=1

Conclusión

En conclusión, un microprocesador de 4 bits es un tipo de procesador que se utiliza para realizar tareas relativamente simples y que requieren menos recursos de procesamiento en comparación con los procesadores más modernos y avanzados. Aunque puede no ser tan potente como los microprocesadores de 8 bits o más, todavía tiene su lugar en aplicaciones específicas donde el costo, el tamaño y el consumo de energía son factores críticos.

La arquitectura de un microprocesador de 4 bits generalmente consiste en un conjunto limitado de instrucciones y registros, lo que lo hace más fácil de diseñar y fabricar. Además, la arquitectura de 4 bits generalmente requiere menos energía y espacio físico en comparación con procesadores más grandes.

En resumen, la elección de un microprocesador de 4 bits dependerá de las necesidades específicas de una aplicación determinada, teniendo en cuenta factores como la eficiencia energética, el tamaño y el costo.

Referencia

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.