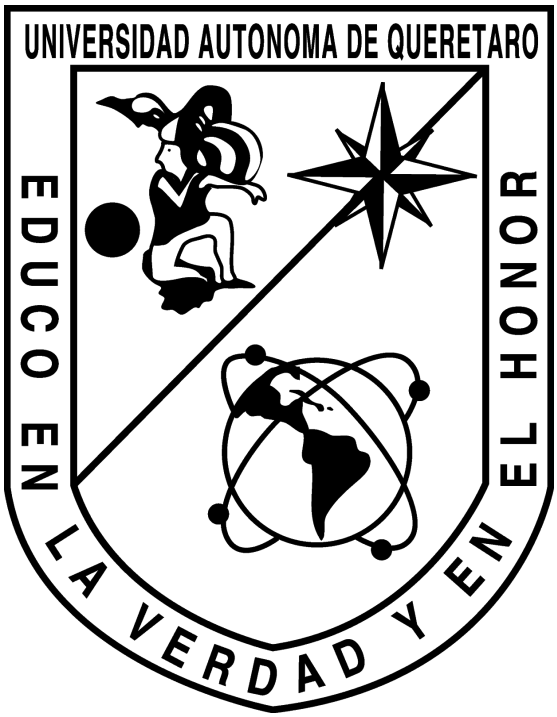


Documentación

Mares Martínez Sergio

Universidad Autónoma de Querétaro, facultad de informática

Dispositivos programables

Romero González Julio Alejandro

Repositorio de los códigos vhdl
https://github.com/SergioMares/VHDL.git

Asignaciones condicionales

Usadas para describir señales que tienen asociadas más de un estado.

```
<señal> <= <valor> when <condición> else
           <valor> when >condición> else
           <valor>;
```

Asignaciones de selección

Las instrucciones de asignación de señal seleccionadas sólo tienen un operador de asignación.

```
with <selección> select
<señal> <= <valor> when <condición>,
           <valor> when <condición>;
```

Instrucción process

sentencia concurrente identificada por su etiqueta, su lista de sensibilidad, un área de declaración y un área de comienzo-final que contiene instrucciones ejecutadas secuencialmente

```
etiqueta:process(lista de sensibilidad) is
<area de declaración>
begin
<instrucciones secuenciales>
end process etiqueta;
```

Instrucción if

Usada para crear una derivación en la ejecución de sentencias secuenciales.

```
if (condicion) then
        <declaracion>
elsif (condicion) then
        <declaracion>
else
        <declaracion>
end if;
```

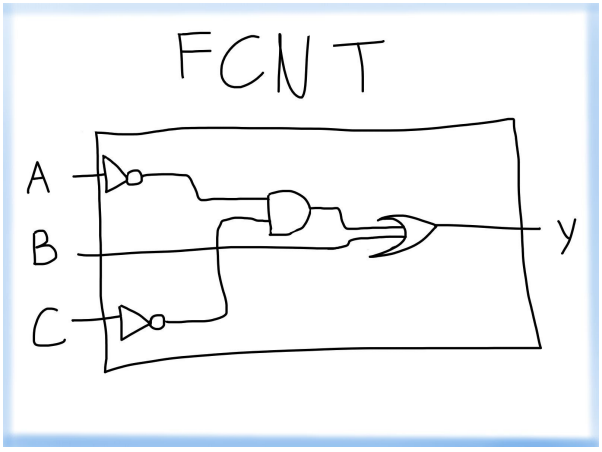
FCNT

Descripción creada al inicio del curso para comprender la arquitectura simple y un poco de la sintaxis de vhdl.

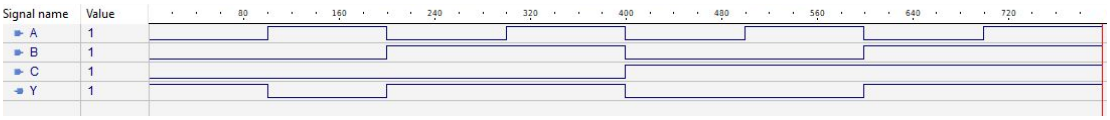
Código

```
library ieee;
use ieee.std_logic_1164.all;

entity FCNT is
        port(
        A, B, C         :       in std_logic := '0';
        Y               :       out std_logic
        );
end entity;

architecture simpleFlujo of FCNT is
begin
        Y <= B or ((not C) and (not A));

end architecture;
```

Diagrama



Simulación



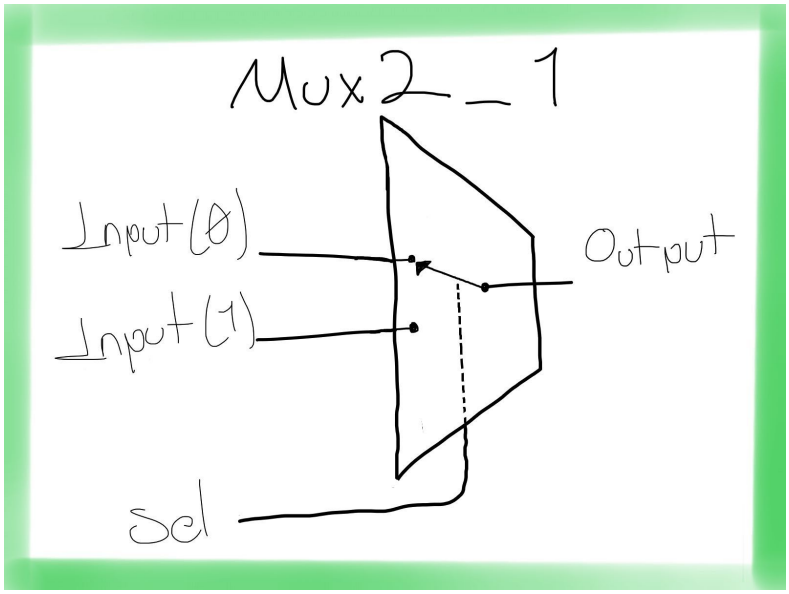
Mux2_1

Un multiplexor tiene la característica de recibir varias entradas y dirigirlas a una específica salida. En este caso, tenemos un multiplexor 2 a 1, es decir, dos entradas a una salida.

Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Mux2_1 is
        port(
        Input   :       in      std_logic_vector(1 downt to 0);
        Sel     :       in      std_logic;
        Output:         out     std_logic
        );
end entity;+
architecture simple of Mux2_1 is
begin
end architecture;
```

Diagrama



Simulación

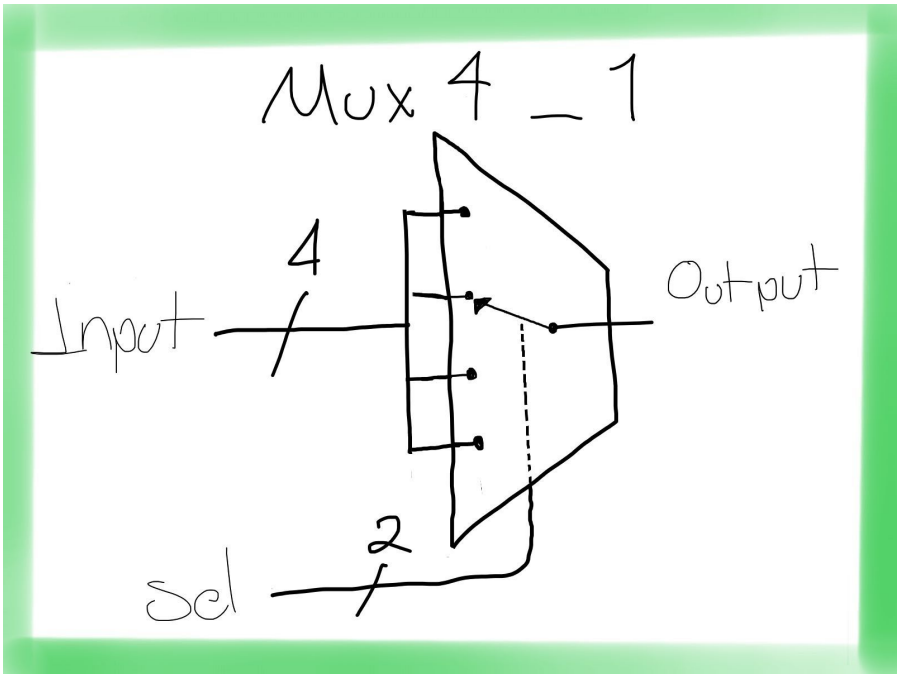
Mux4_1Map

Este es un multiplexor 4 a 1 pero tiene una diferencia interesante en el código, implementa mapeo. El mapeo es prácticamente mandar llamar a otros códigos que ya hemos hecho. Esto para tener un código más ordenado y reutilizar código. El multiplexor 4 a 1 presentado se crea usando tres multiplexores 2 a 1.

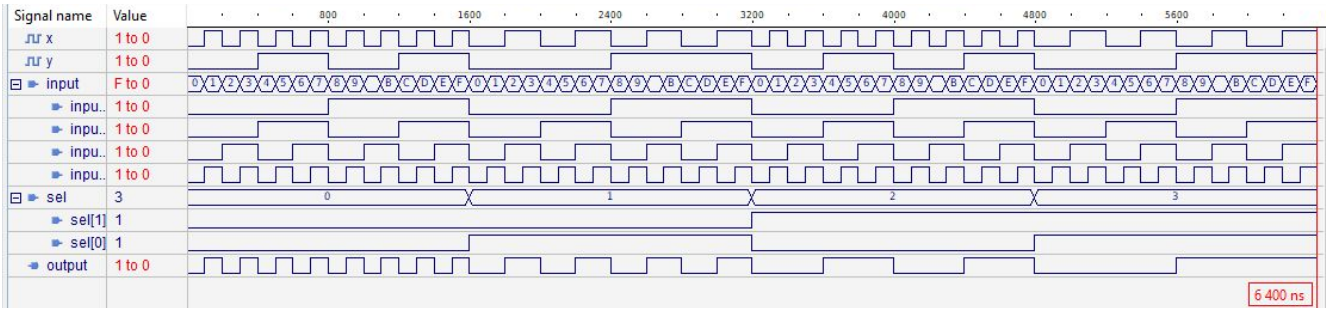
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Mux4_1Map is
	port(
	input:  in	std_logic_vector(3 downto 0);
	sel:	in	std_logic_vector(1 downto 0);
	output:		out	std_logic
	);
end entity;
architecture simple of Mux4_1Map is
component Mux2_1 is
	port(
		I0, I1:  in		std_logic;
		sel:		in		std_logic;
		Output:		out		std_logic
	);
end component;
signal x, y: std_logic;
begin
	bloque0: Mux2_1 port map(input(0),input(1),sel(0),x);
	bloque1: Mux2_1 port map(input(2),input(3),sel(0),y);
	bloque3: Mux2_1 port map(x,y,sel(1),output);
end architecture;
```

Diagrama



Simulación



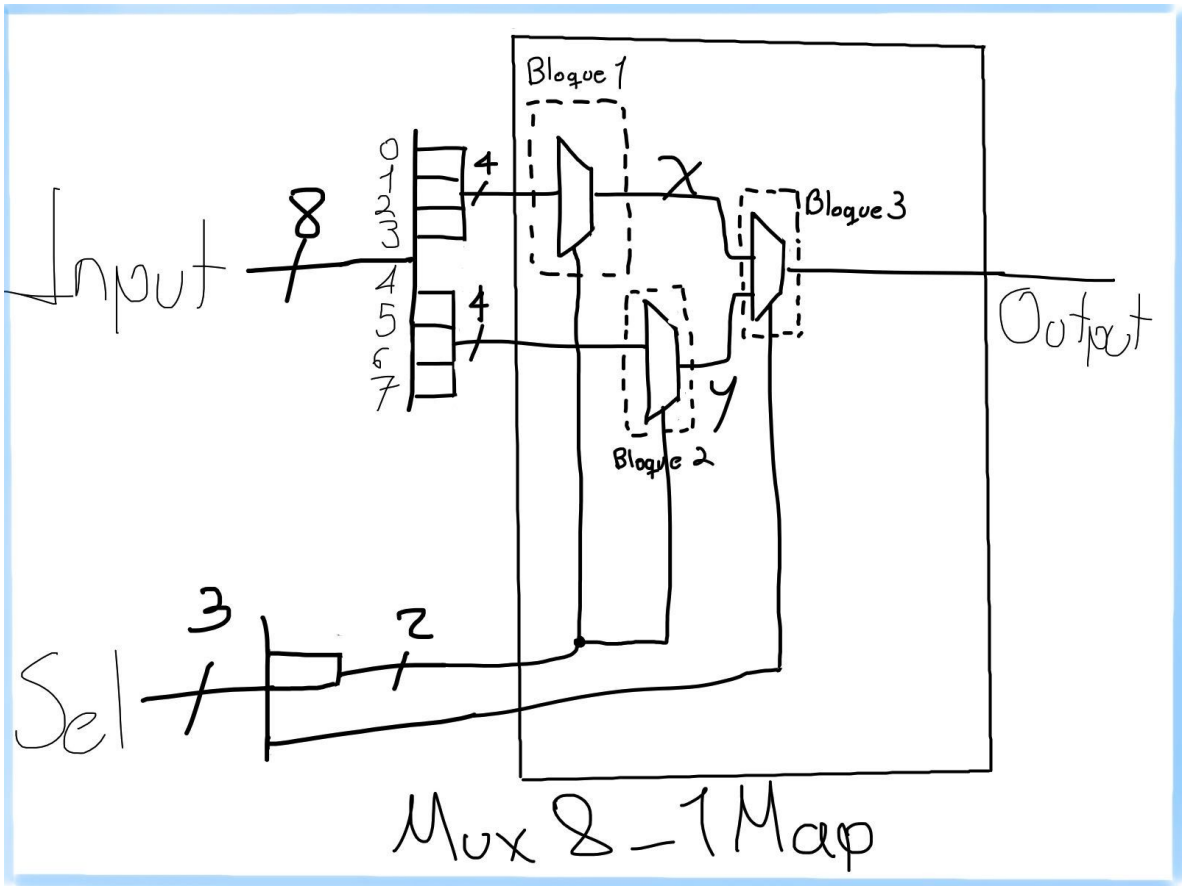
Mux8_1Map

Este multiplexor es similar al anterior, es creado a partir de otros por medio de mapeo(mux2_1 mux4_1).

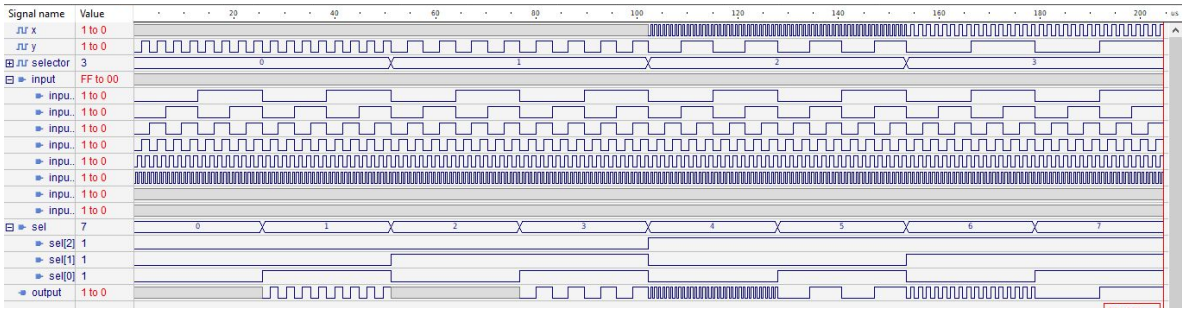
Código

```
library IEEE;
use ieee.std_logic_1164.all;
entity Mux8_1Map is
	port(
	input:  in	std_logic_vector(7 downto 0);
	sel:	in	std_logic_vector(2 downto 0);
	output: out std_logic
	);
end entity;
architecture simple of Mux8_1Map is
component Mux2_1 is
	port(
		I0, I1:  in		std_logic;
		sel:		in		std_logic;
		Output:		out		std_logic
	);
	end component;
component Mux4_1 is
	port(
	A, B, C, D:	in	std_logic;
	Sel:			in	std_logic_vector(1 downto 0);
	Output:				out std_logic
	);
end component;
signal x, y:	std_logic;
signal selector: std_logic_vector(1 downto 0);
begin
		selector <= sel(2)&sel(1);
		bloque0: Mux4_1 port map(input(0),input(1),input(2),input(3),sel(2
downto 1),x);
		bloque1: Mux4_1 port
map(input(4),input(5),input(6),input(7),selector,y);
		bolque3: Mux2_1 port map(x,y,sel(0),output);
end architecture;
```

Diagrama



Simulación

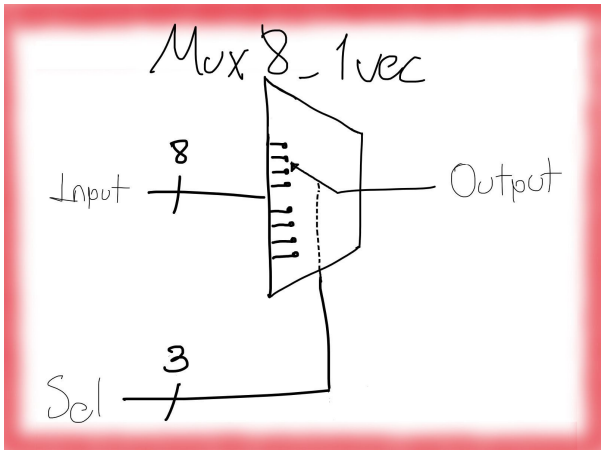


Mux8_1vec

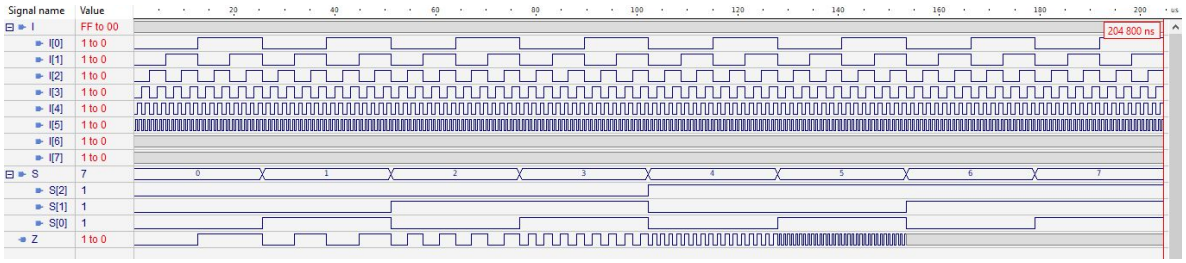
Este también es un multiplexor 8 a 1, pero ahora sin mapeo. Utilizamos un vector de entrada de 8 bits y un bit de salida. Se implementa la asignación condicional para elegir la señal de salida.

```
library IEEE;
use ieee.std_logic_1164.all;
entity Mux8_1vec is
	port(
	--I:	in	std_logic_verctor(7 downto 0);
	I:	in	std_logic_vector(0 to 7);
	S:	in	std_logic_vector(2 downto 0);
	Z:	out std_logic
	);
end entity;
architecture simple of Mux8_1vec is
begin
z <= I(0) when (S="000") else
	I(1) when (S="001") else
	I(2) when (S="010") else
	I(3) when (S="011") else
	I(4) when (S="100") else
	I(5) when (S="101") else
	I(6) when (S="110") else
	I(7) when (S="111") else;
end architecture;
```

Diagrama



Simulación

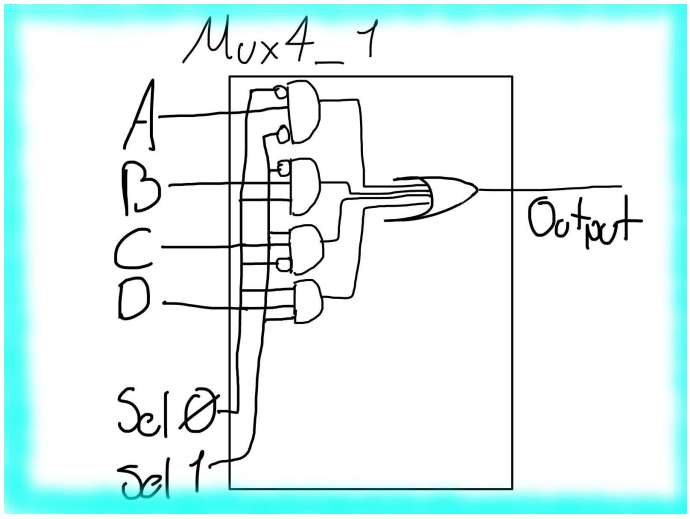


Mux4_1

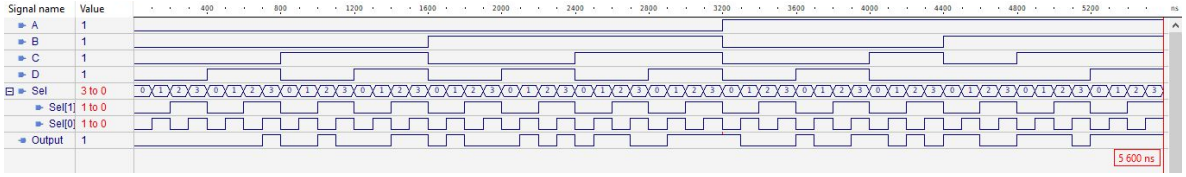
Un multiplexor 4 a 1, pero ahora sin mapeo y usando concurrencia para la asignación de la salida.

```
library IEEE;
use ieee.std_logic_1164.all;
entity Mux4_1 is
        port(
        A, B, C, D:     in      std_logic;
        Sel:                    in      std_logic_vector(1 downto 0);
        Output:                         out std_logic
        );
end entity;
architecture flujo of Mux4_1 is
begin
Output <= (A and not Sel(1) and not Sel(0)) or
                (B and not Sel(1) and Sel(0)) or
                (C and Sel(1) and not Sel(0)) or
                (D and Sel(1) and Sel(0));
end architecture;
```

Diagrama



Simulación



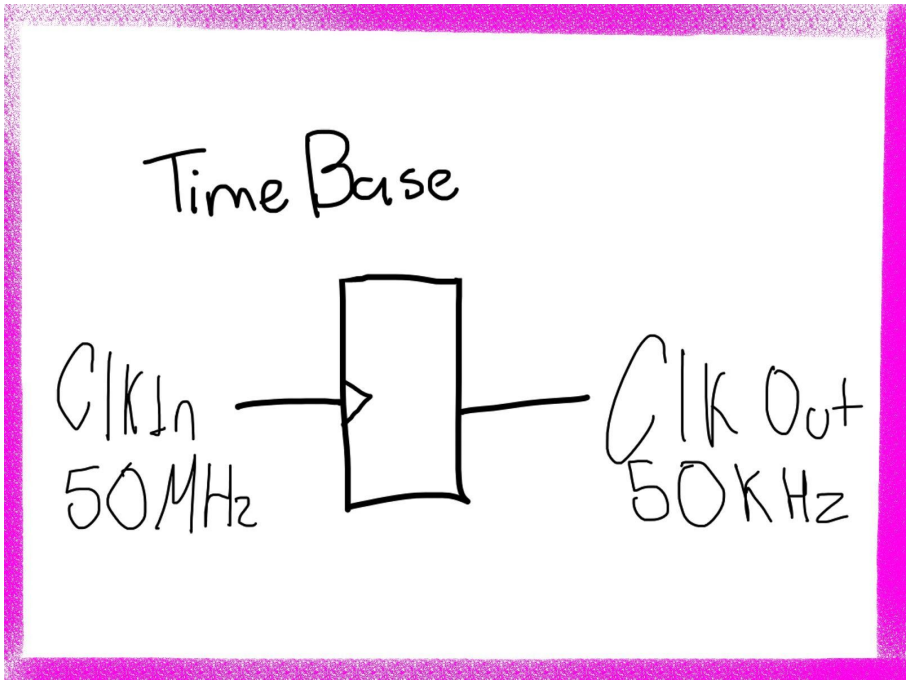
TB50KHZ

El siguiente código describe una base de tiempo de 50 kilohertz. Por primera vez se hace uso de la librería arith y unsigned. Como entrada tenemos un reloj a 50MHZ y la salida de 50KHZ. Esto lo logramos al usar un contador, el cual al llegar a cierto número dará un alto lógico. El propósito de las bases de tiempo es reducir el reloj de entrada, dando como salida una frecuencia menor.

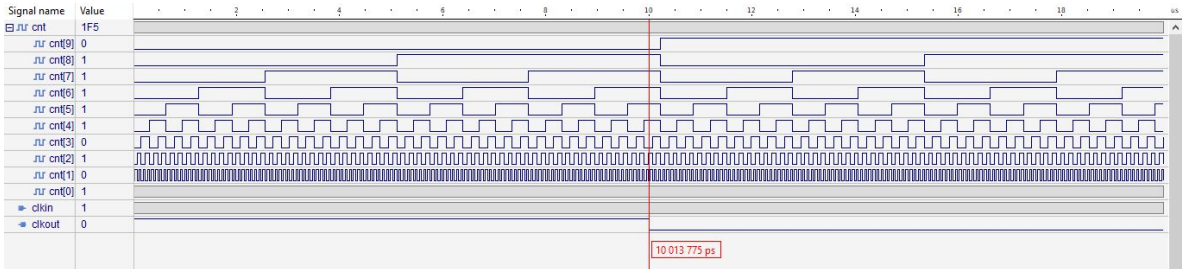
Código

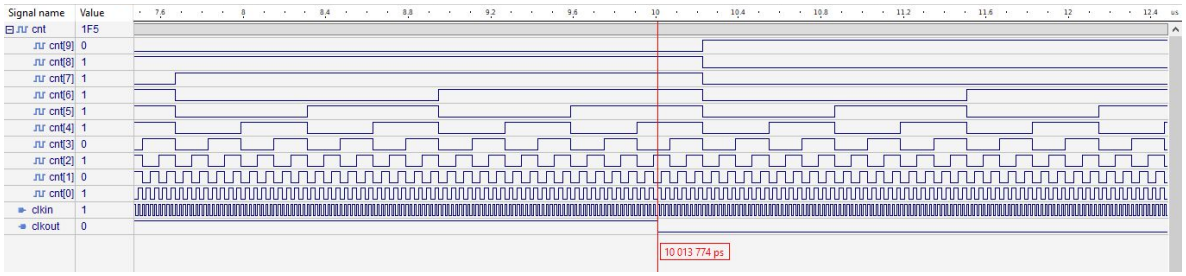
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity TB50KHZ is
	port(
	clkin:  in	std_logic;
	clkout: out	std_logic
	);
end entity;
architecture behavioural of TB50KHZ is
signal cnt: std_logic_vector(9 downto 0) := "0000000000";
begin
	process(clkIn)
		begin
		if (clkin 'event and clkin='1') then --Verificar si ha sucesido algún cambio, rising_edge(clkIn)
			--contamos pulsos
			cnt<=cnt+"0000000001";	--si no tienen la librería unsigned dará error, porque no sabe si está o no manejando signos
			if(cnt<"0111110100") then --[0,500]
				clkout <='1';
			elsif(cnt<"1111101000") then --[500, 1000]
				clkout <= '0';
			else
				cnt <= "0000000000"; --reiniciamos el counter
			end if;
		end if;
	end process;
end architecture;
```

Diagrama



Simulación



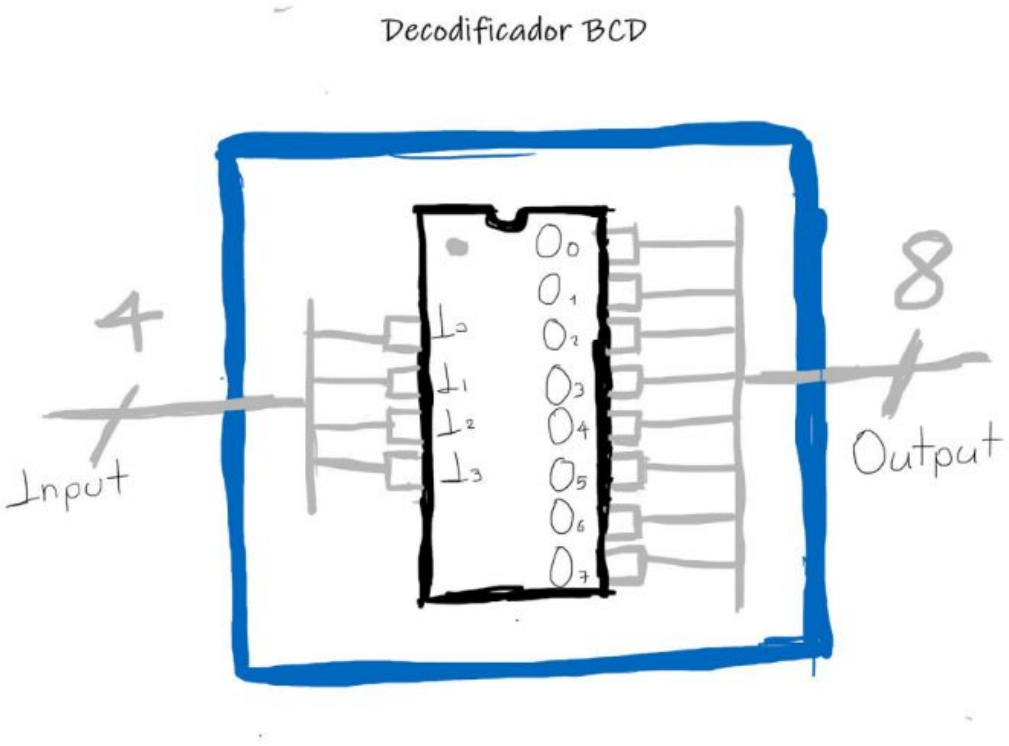


DecoBCD

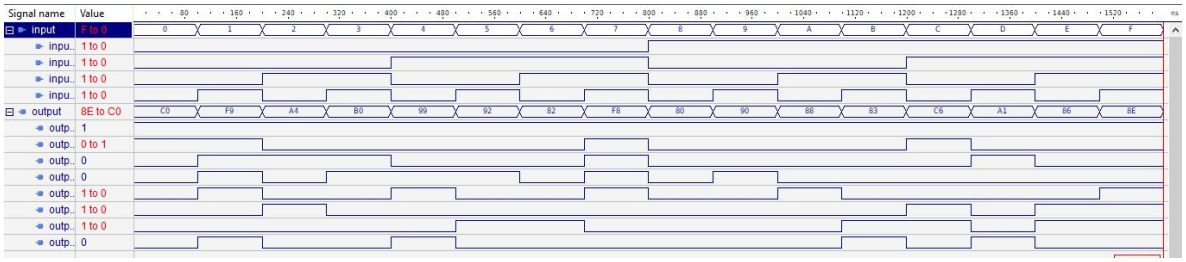
Un decodificador es un circuito que recibe un número de entradas determinado y aumenta el número de salidas. En este caso se describe un decodificador con cuatro bits de entrada y 8 de salida. El uso está pensado para mostrar valores hexadecimales en un display de 7 segmentos.

```
library ieee;
use ieee.std_logic_1164.all;
entity DecoBCD is
        port(
        input:  in      std_logic_vector(3 downto 0);
        output:         out     std_logic_vector(7 downto 0)
        );
end entity;
architecture simple of DecoBCD is
begin
output<= X"C0" when(input=X"0") else
                X"F9" when(input=X"1") else
                X"A4" when(input=X"2") else
                X"B0" when(input=X"3") else
                X"99" when(input=X"4") else
                X"92" when(input=X"5") else
                X"82" when(input=X"6") else
                X"F8" when(input=X"7") else
                X"80" when(input=X"8") else
                X"90" when(input=X"9") else
                X"88" when(input=X"A") else
                X"83" when(input=X"B") else
                X"C6" when(input=X"C") else
                X"A1" when(input=X"D") else
                X"86" when(input=X"E") else
            X"8E";
end architecture;
```

Diagrama



Simulación



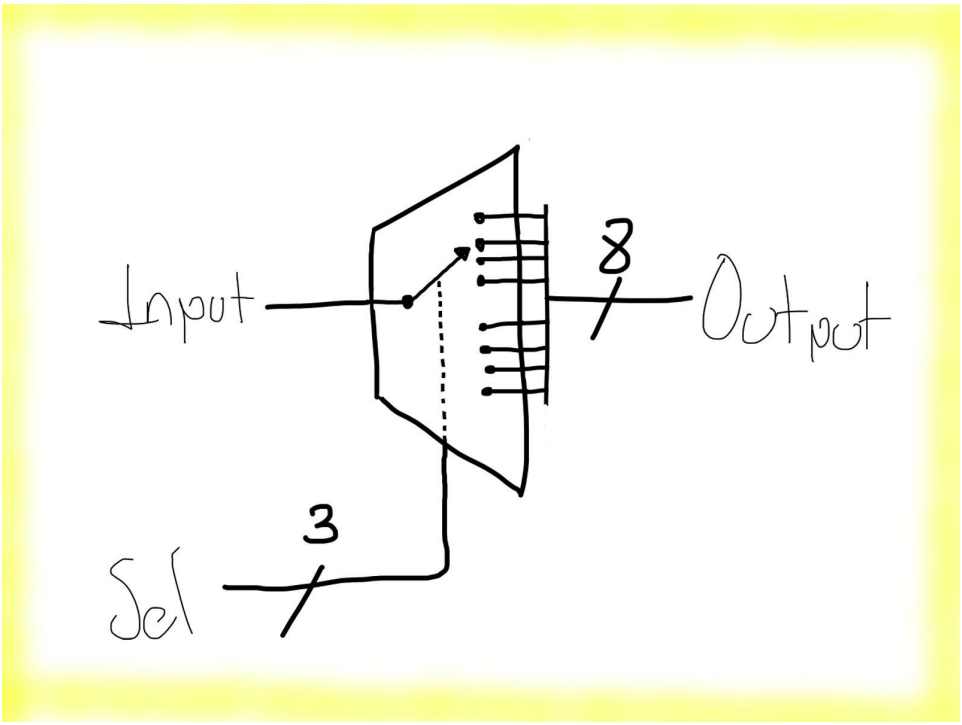
Demux3_8

El Demultiplexor tiene una entrada y múltiples salidas. Por medio de un selector decide una de las posibles salidas.

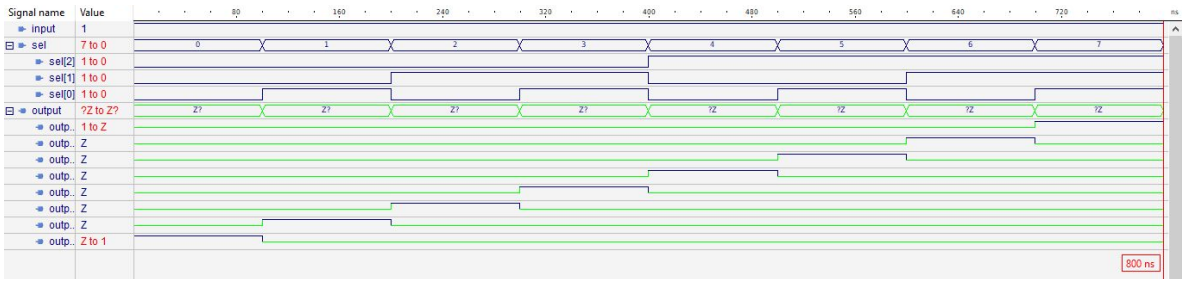
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Demux3_8 is
	port(
	input:  in	std_logic;
	sel:	in	std_logic_vector(2 downto 0);
	output: out	std_logic_vector(7 downto 0)
	);
end entity;
architecture simple of Demux3_8 is
begin
with sel select
output(0) <= input when ("000"),
'Z' when others;
with sel select
output(1) <= input when ("001"),
'Z' when others;
with sel select
output(2) <= input when ("010"),
'Z' when others;
with sel select
output(3) <= input when ("011"),
'Z' when others;
with sel select
output(4) <= input when ("100"),
'Z' when others;
with sel select
output(5) <= input when ("101"),
'Z' when others;
with sel select
output(6) <= input when ("110"),
'Z' when others;
with sel select
output(7) <= input when ("111"),
'Z' when others;
end architecture;
```

Diagrama



Simulación



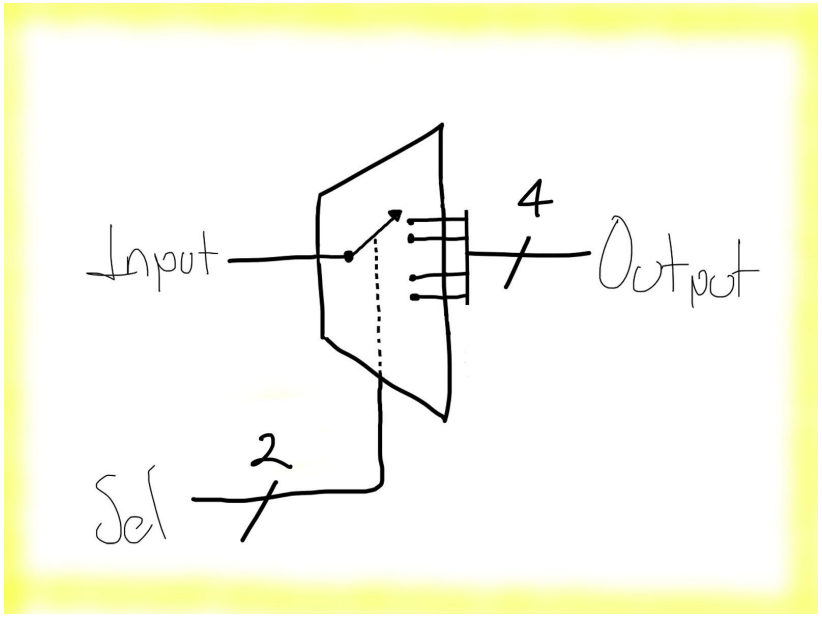
Demux2_4

Demultiplexor con una salida de 4 bits con arquitectura de comportamiento.

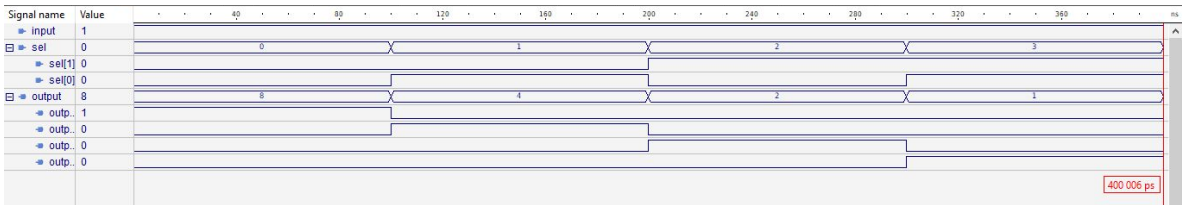
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Demux2_4 is
	port(
	input:  in	std_logic;
	sel:	in	std_logic_vector(1 downto 0);
	output:		out	std_logic_vector(3 downto 0)
	);
end entity;
architecture comportamiento of Demux2_4 is
begin
	--label:process( list )
	process(sel) is
begin
	if sel="00" then
		--output(0) <= input;
		output <= input&"000";
	elsif sel ="01" then
		output <= '0'&input&"00";
	elsif sel ="10" then
		output <= "00"&input&'0';
	else
		output <= "000"&input;
	end if;
end process;
end architecture;
```

Diagrama



Simulación



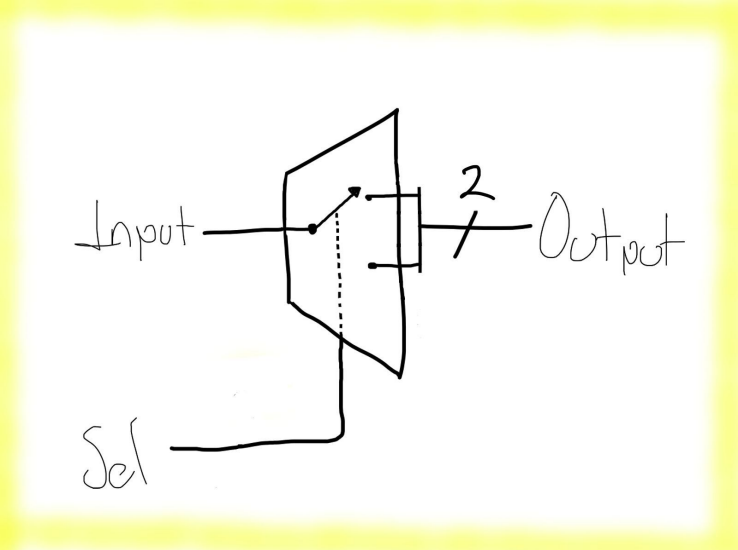
Demux2_1

Pequeño demultiplexor con solo dos salidas.

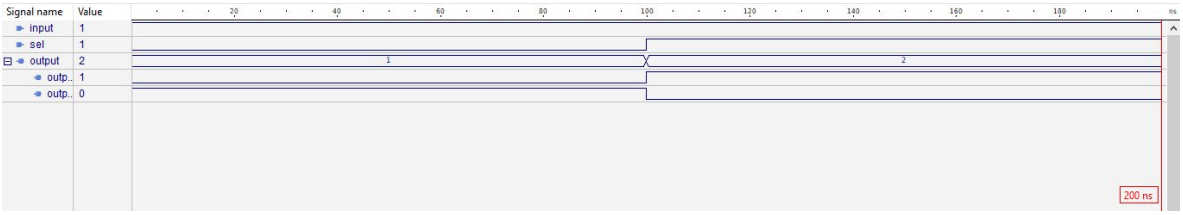
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Demux2_1 is
	port(
		input:  in	std_logic;
		sel:	in	std_logic;
		output:		out	std_logic_vector(1 downto 0)
	);
end entity;
architecture simple of Demux2_1 is
begin
output(0)<=input and (not(sel));
output(1)<=input and sel;
end architecture;
```

Diagrama



Simulación

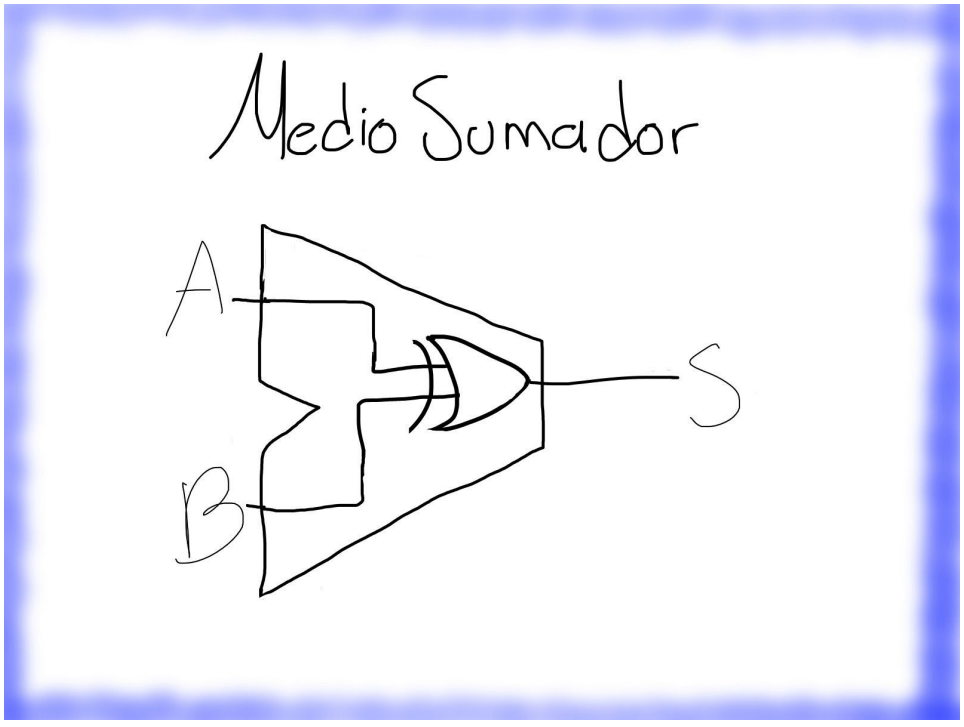


MedioSumador

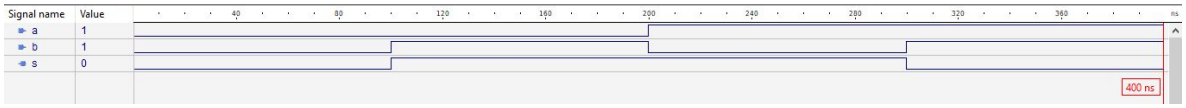
Un medio sumador realiza una suma de 1 bit únicamente. Recibe como entradas los dos números a sumar y salida de un resultado de 1 bit. al ser el resultado de 1 solo bit, se le llama medio sumador, pues no contempla el desbordamiento.

```
library ieee;
use ieee.std_logic_1164.all;
entity MedioSumador is
        port(
        a, b:   in      std_logic;
        s       :       out std_logic
        );
end entity;
architecture simple of MedioSumador is
begin
        s <= a xor b;
end architecture;
```

Diagrama



Simulación



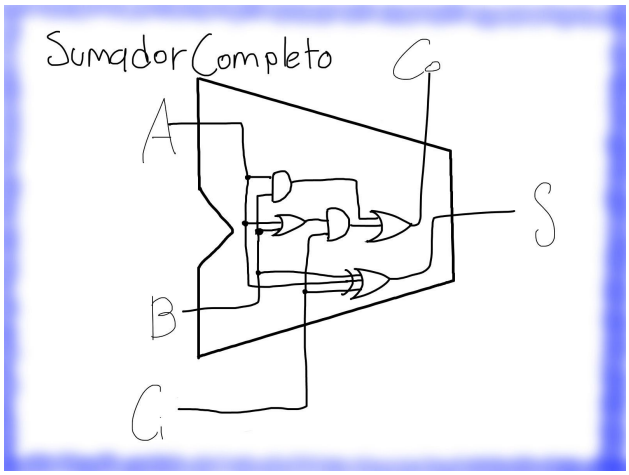
SumadorCompleto

A diferencia del sumador anterior, este si contempla el desbordamiento, mejor llamado como acarreo. El acarreo se da cuando la palabra del resultado es mayor a la salida. Un sumador completo toma en cuenta el posible acarreo de una suma anterior y el acarreo actual como una salida.

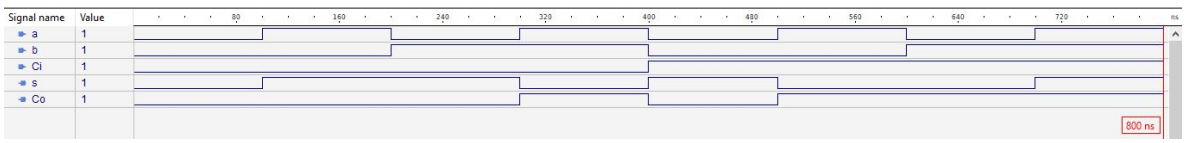
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity SumadorCompleto is
	port(
	a, b	:	in	std_logic;--palabras a sumar
	Ci		:	in	std_logic;--Acarreos anteriores
	s		:	out std_logic;--Resultado de la suma
	Co		:	out std_logic--Acarreo actual
	);
end entity;
architecture simple of SumadorCompleto is
begin
	s <= Ci xor a xor b;
	Co <=(Ci and (a or b)) or (a and b);
end architecture;
```

Diagrama



Simulación



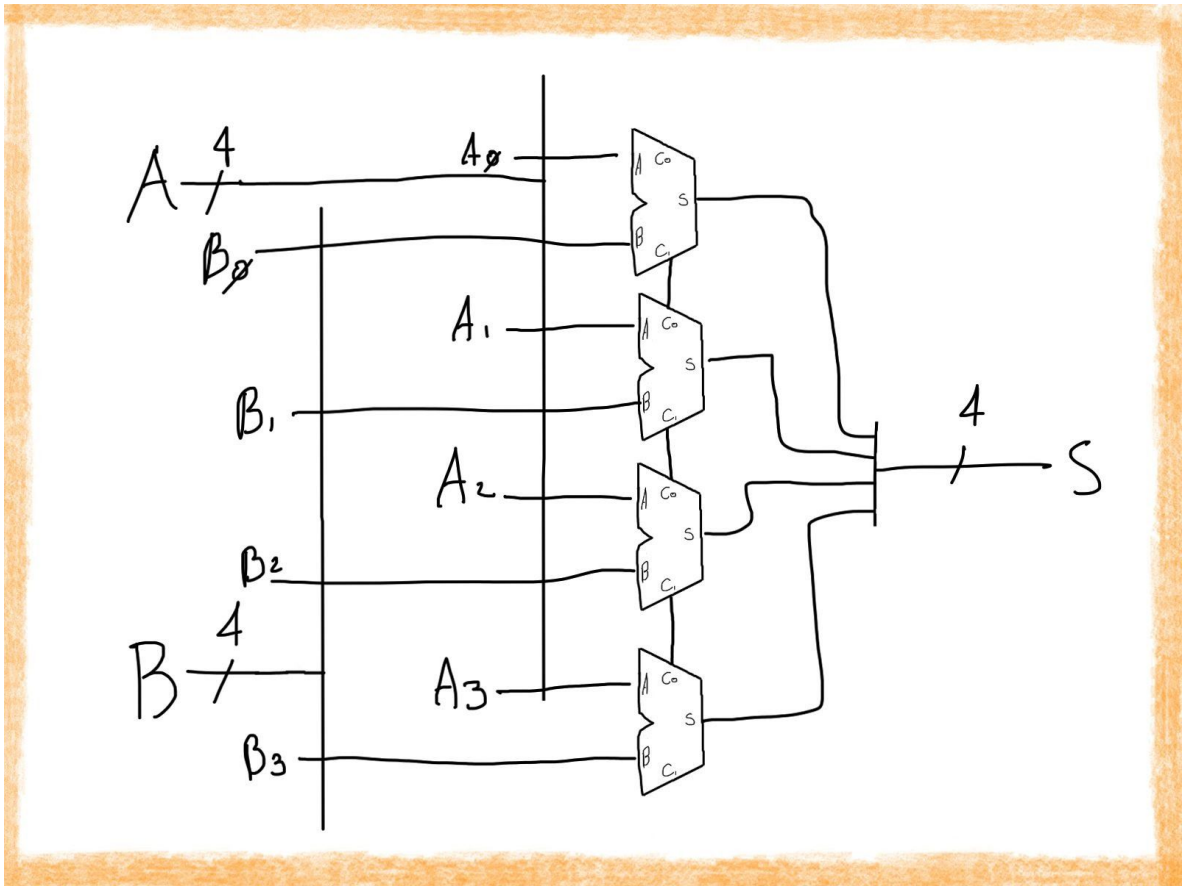
Sumador4bits

El sumador presente está creado a partir de sumadores completos de 1 bit mapeados.

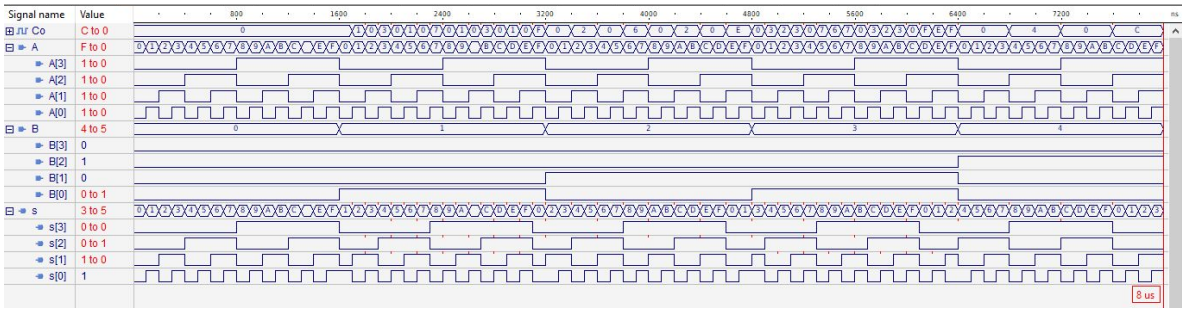
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Sumador4bits is
	port(
	A, B	:	in std_logic_vector(3 downto 0);
	s		:	out std_logic_vector(3 downto 0)
	);
end entity;
architecture simple of Sumador4Bits is
component SumadorCompleto is
	port(
	a, b	:	in	std_logic;--palabras a sumar
	Ci		:	in	std_logic;--Acarreos anteriores
	s		:	out std_logic;--Resultado de la suma
	Co		:	out std_logic--Acarreo actual
	);
end component;
signal Co: std_logic_vector(3 downto 0);
begin
	bloque0: SumadorCompleto port map (A(0),B(0),'0',s(0),Co(0));
	bloque1: SumadorCompleto port map (A(1),B(1),Co(0),s(1),Co(1));
	bloque2: SumadorCompleto port map (A(2),B(2),Co(1),s(2),Co(2));
	bloque3: SumadorCompleto port map (A(3),B(3),Co(2),s(3),Co(3));
end architecture;
```

Diagrama



Simulación



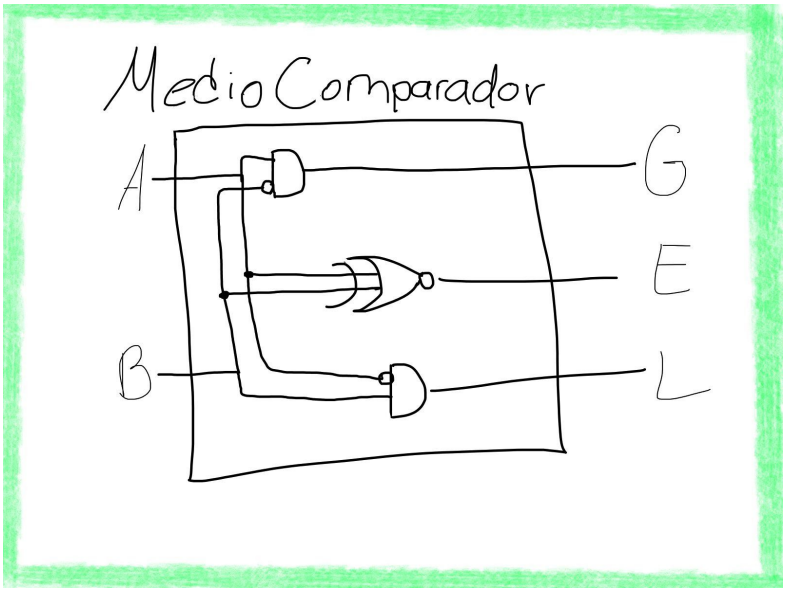
MedioComparador

Un comparador nos dirá si una trama contra otra es menor, igual o mayor que esta.

Código

```
library ieee;
use ieee.std_logic_1164.all;
entity MedioComparador is
	port(
	A, B	:	in	std_logic;
	G, E, L	:	out std_logic
	);
end entity;
architecture simple of MedioComparador is
begin
	G <= A and not(B);
	E <= A xnor B;
	L <= not(A) and B;
end architecture;
```

Diagrama



Simulación

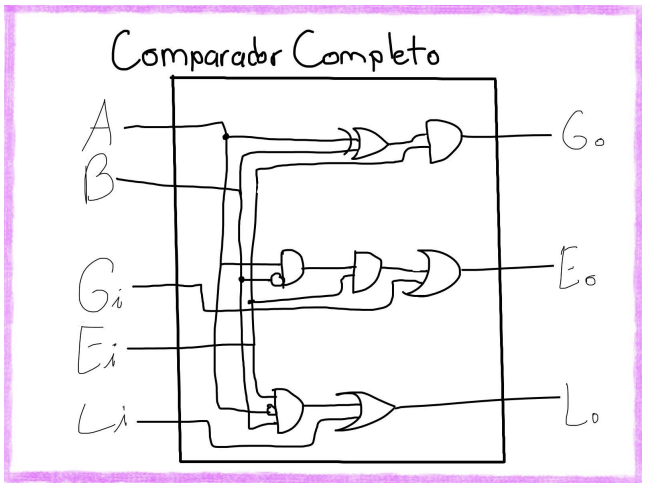
ComparadorCompleto

Un comparador completo contempla como entradas una posible comparación anterior, donde sabe si la comparación anterior dio resultado como mayor, igual o menor.

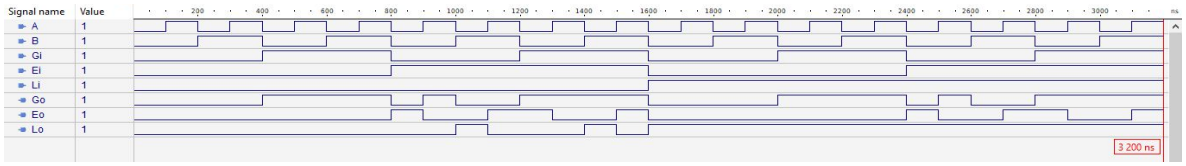
Código

```
 library ieee;
 use ieee.std_logic_1164.all;
 entity ComparadorCompleto is
          port(
          A, B          :       in      std_logic;
          Gi, Ei, Li    :       in      std_logic;
          Go, Eo, Lo    :       out std_logic
          );
 end entity;
architecture simple of ComparadorCompleto is
begin
         Eo <= Ei and (A xnor B);
         Go <=(Ei and (A and not(B))) or Gi;
         Lo <=(Ei and not(a) and B) or Li;
end architecture;
```

Diagrama



Simulación



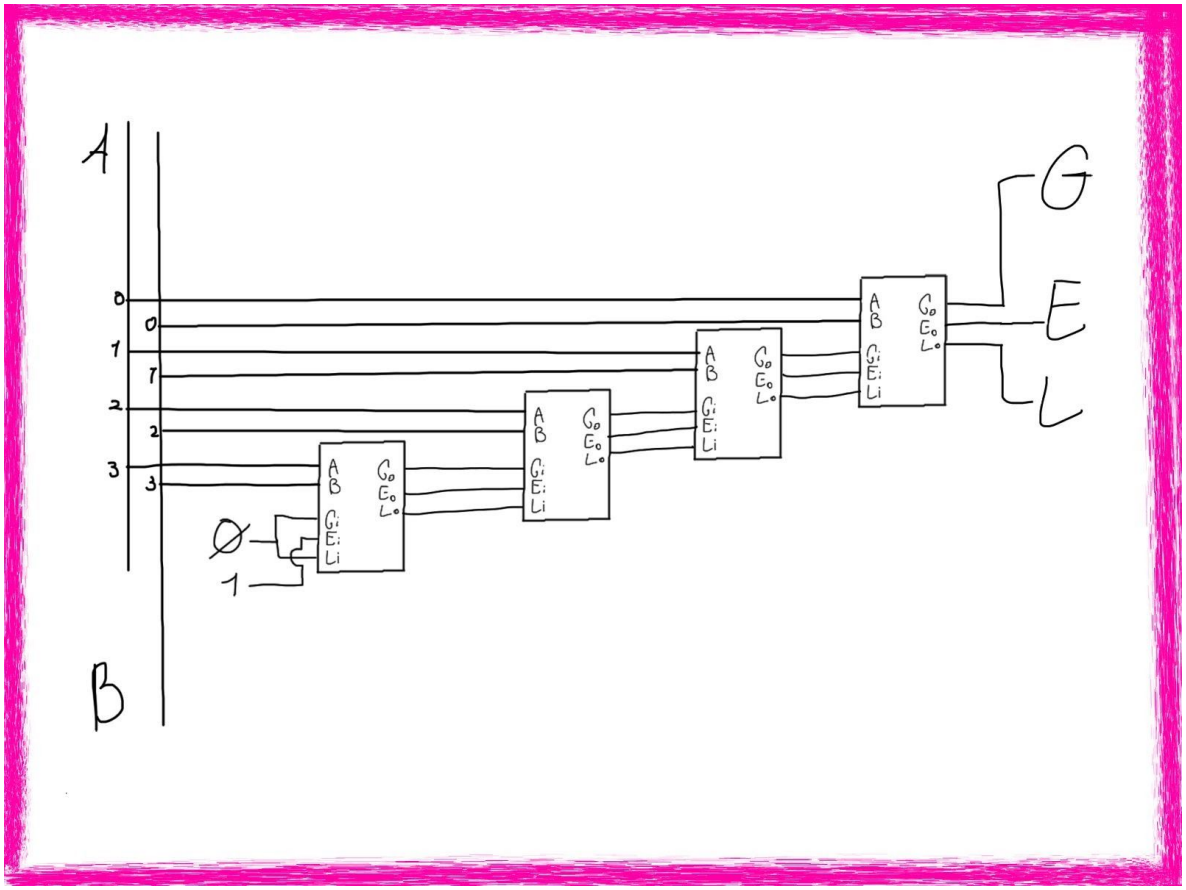
Comparador4Bits

Este comparador está hecho de comparadores completos mapeados. Así se muestra más claro la importancia de las salidas/entradas GEL.

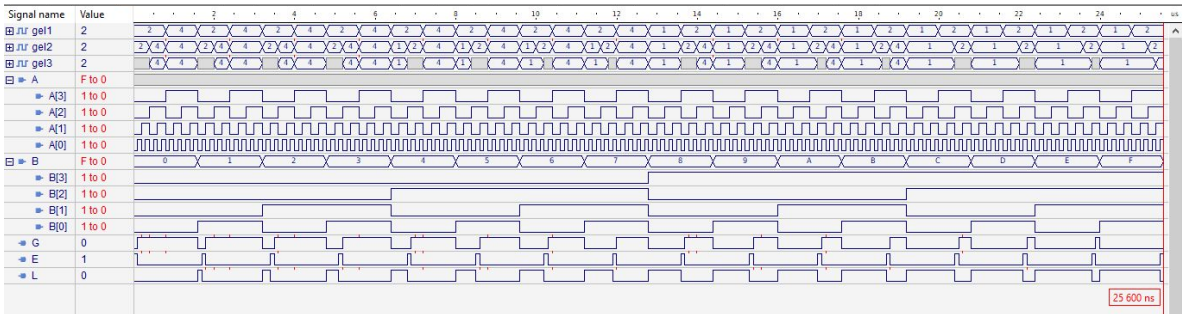
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Comparador4bits is
	port(
	 A, B		:	in	std_logic_vector(3 downto 0);
	 G, E, L	:	out std_logic
	);
end entity;
architecture simple of Comparador4bits is
component ComparadorCompleto is
	 port(
	 A, B		:	in	std_logic;
	 Gi, Ei, Li	:	in	std_logic;
	 Go, Eo, Lo	:	out std_logic
	 );
end component;
signal gel1, gel2, gel3:	std_logic_vector(2 downto 0);
begin
	bloque0: ComparadorCompleto port
map(A(3),B(3),'0','1','0',gel1(2),gel1(1),gel1(0));
	bloque1: ComparadorCompleto port
map(A(2),B(2),gel1(2),gel1(1),gel1(0),gel2(2),gel2(1),gel2(0));
	bloque2: ComparadorCompleto port
map(A(1),B(1),gel2(2),gel2(1),gel2(0),gel3(2),gel3(1),gel3(0));
	bloque3: ComparadorCompleto port
map(A(0),B(0),gel3(2),gel3(1),gel3(0),G,E,L);
end architecture;
```

Diagrama



Simulación

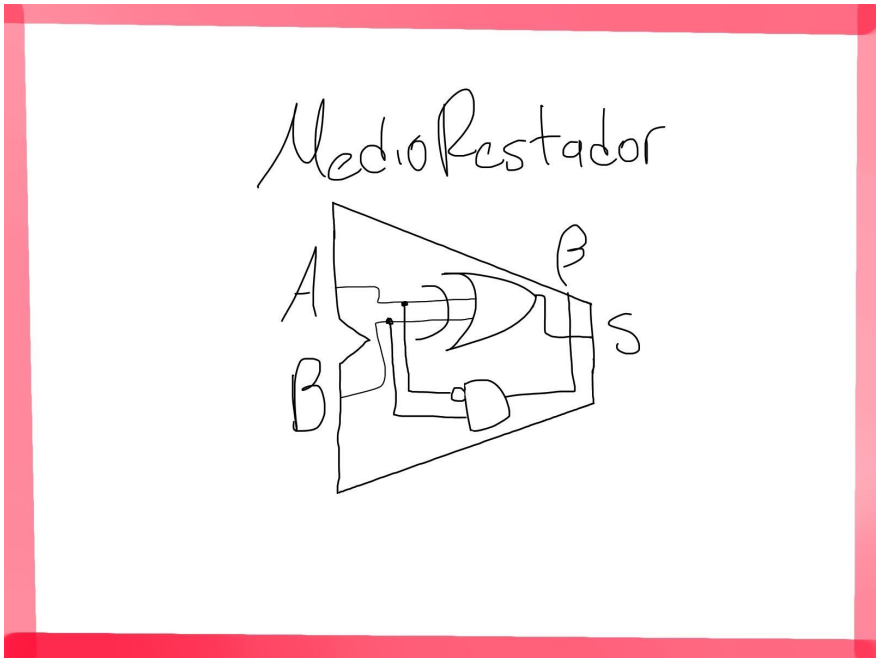


MedioRestador1bit

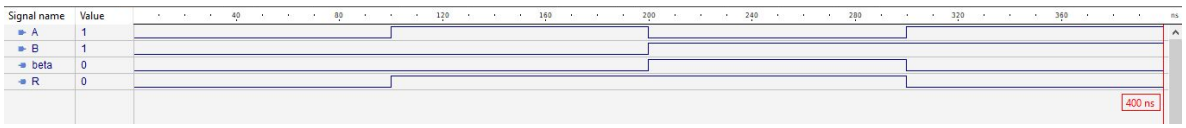
La lógica del restador es similar al sumador, solo que en lugar de tener un acarreo, lo llamaremos el prestado, que es cuando tenemos un resultado negativo.

```
library ieee;
use ieee.std_logic_1164.all;
entity MedioRestador1bit is
        port(
        A, B:   in      std_logic; --palabras binarias 1bit
        beta:   out     std_logic; --Lo que se lleva prestado(borrow)
        R       :       out std_logic --Resulado de restar palabras binarias
        );
end entity;
architecture simple of MedioRestador1bit is
begin
        R <=  A xor B;
        beta <= not(a) and B;
end architecture;
```

Diagrama



Simulación



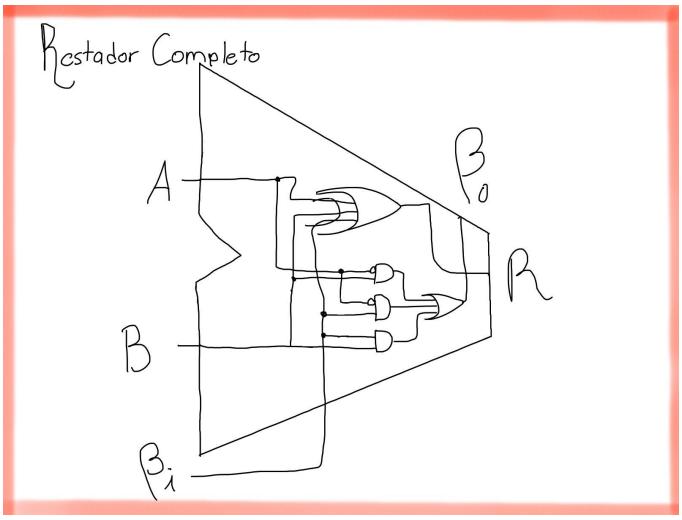
RestadorCompleto1bit

El restador completo toma en cuenta una posible resta anterior, así sabiendo si hubo borrow en la resta anterior y en la actual.

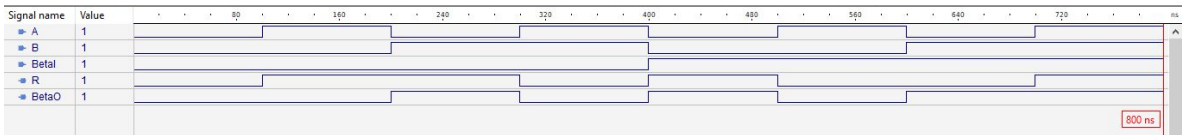
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity RestadorCompleto1bit is
	port(
	A		:	in	std_logic; --Palabras binarias de 1 bit
	B		:	in	std_logic; --Palabras binarias de 1 bit
	BetaI	:	in	std_logic; --Bit prestado de resta anterior
	R		:	out	std_logic;
	BetaO	:	out	std_logic  --Indica si se lleva pretado algún bit
	);
end entity;
architecture simple of RestadorCompleto1bit is
begin
	R		<= BetaI xor A xor B;
	BetaO		<= ((not A and B) or (BetaI and not A) or (BetaI and B));
end architecture;
```

Diagrama



Simulación



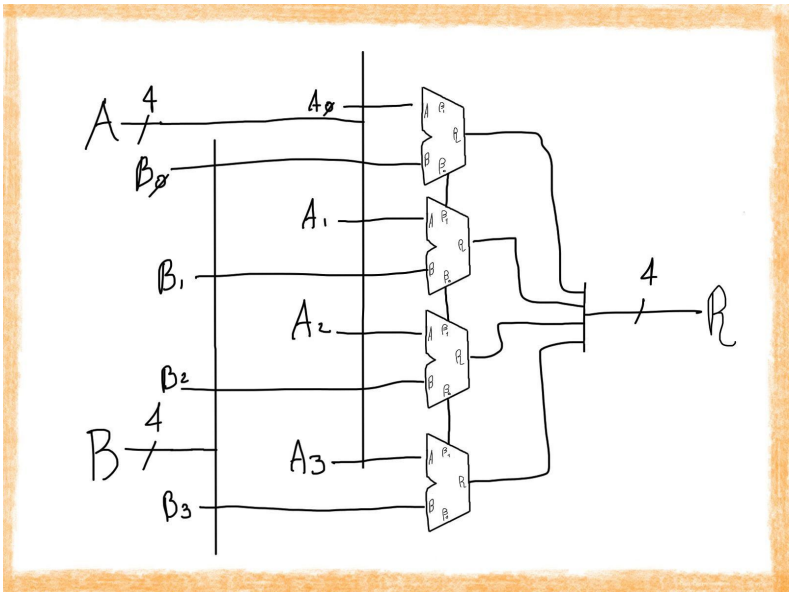
RestadorCompleto4bit

Al igual que el sumador y el comparador de 4 bits, este restador también está creado por mapeo, permitiéndonos hacer una resta más grande.

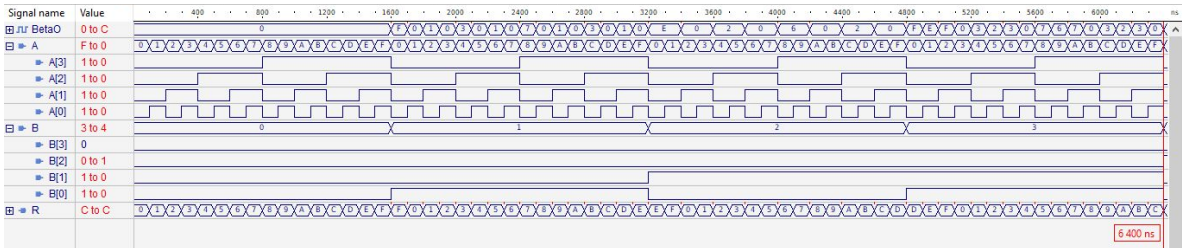
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity RestadorCompleto4bit is
	port(
	A,B		:	in	std_logic_vector(3 downto 0);
	R		:	out	std_logic_vector(3 downto 0)
	);
end entity;
architecture simple of RestadorCompleto4bit is
component RestadorCompleto1bit is
	port(
	A		:	in	std_logic; --Palabras binarias de 1 bit
	B		:	in	std_logic; --Palabras binarias de 1 bit
	BetaI	:	in	std_logic; --Bit prestado de resta anterior
	R		:	out	std_logic;
	BetaO	:	out	std_logic  --Indica si se lleva pretado algún bit
	);
end component;
signal BetaO:		std_logic_vector(3 downto 0);
begin
	bloque0: RestadorCompleto1bit port map (A(0),B(0),'0',R(0),BetaO(0));
	bloque1: RestadorCompleto1bit port map
(A(1),B(1),BetaO(0),R(1),BetaO(1));
	bloque2: RestadorCompleto1bit port map
(A(2),B(2),BetaO(1),R(2),BetaO(2));
	bloque3: RestadorCompleto1bit port map
(A(3),B(3),BetaO(2),R(3),BetaO(3));
end architecture;
```

Diagrama



Simulación



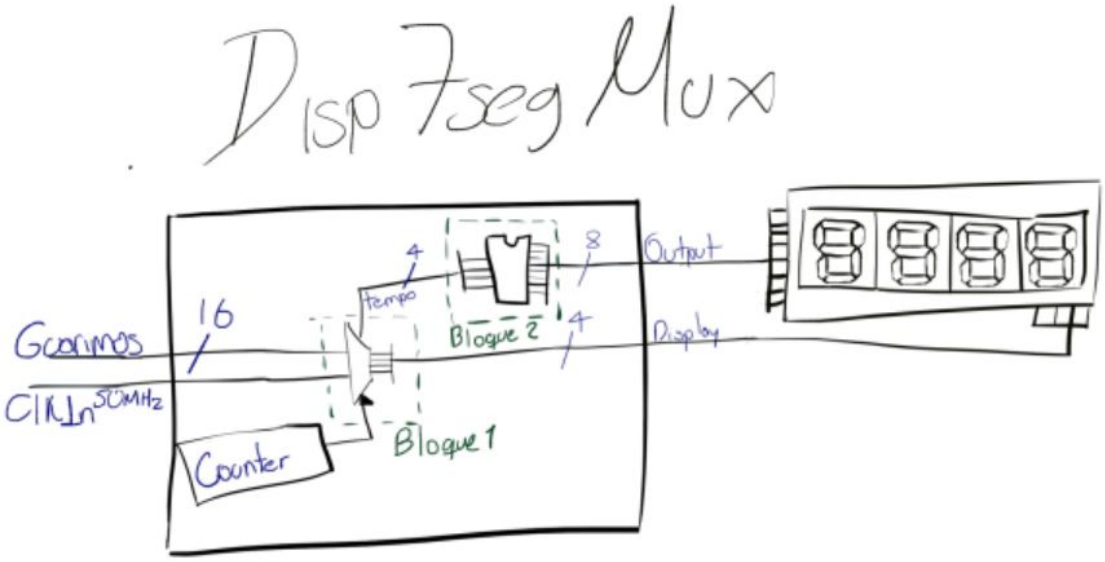
Disp7segMux

El siguiente código describe el circuito para mostrar guarismos en un display de 7 segmentos multiplexado. El código tiene como entradas un reloj a 50MHZ, un vector de 16 bits que son los guarismos a mostrar. Tiene dos salidas, una de 8 bits para representar los guarismos en cada display individual y la salida de 4 bits es para asignar el display a activar. Este circuito además cuenta con un decodificador. La señal tempo será la que lleve el guarismo decodificado para ser mostrado en el display

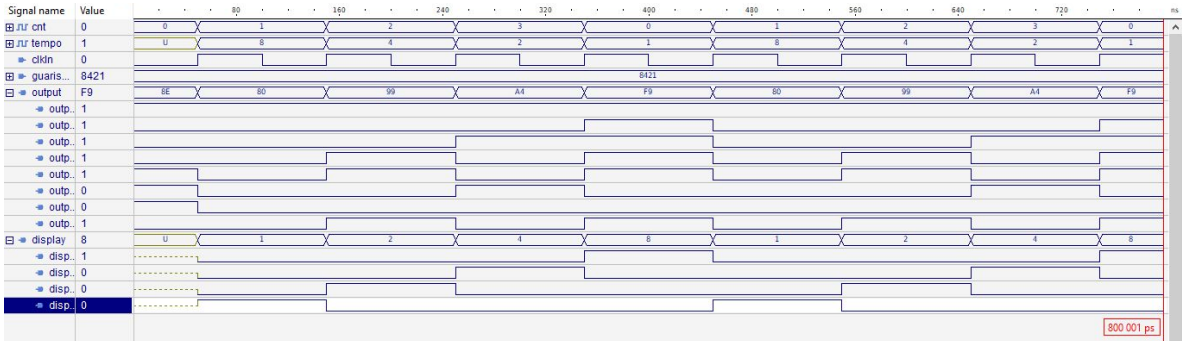
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;          --La necesitamos para el counter
use ieee.std_logic_unsigned.all;--tenemos que especificar que no tienen signo
entity Disp7segMux is
        port(
        clkIn           :       in      std_logic;
        guarismos       :       in      std_logic_vector(15 downto 0);
        output          :       out     std_logic_vector(7 downto 0);
        display                 :       out std_logic_vector(3 downto 0)
        );
end entity;
architecture simple of Disp7segMux is
signal cnt      : std_logic_vector(1 downto 0) := "00"; --Contador para saber que
display activar
signal tempo: std_logic_vector(3 downto 0);
begin
        process(clkIn)
        begin
                if (clkIn 'event and clkIn = '1') then
                        if(cnt = "11") then     --Evitamos que se desborde y reiniciamos
                                cnt <= "00";            --la cuenta
                        end if;
                        cnt<=cnt+"01";
                        if(cnt = "00") then
                                tempo           <=      guarismos(15 downto 12);
                                display <=      "0001";
                        elsif(cnt = "01") then
                                tempo           <= guarismos(11 downto 8);
                                display <=      "0010";
```

```
                elsif(cnt = "10") then
                        tempo           <= guarismos(7 downto 4);
                        display <=      "0100";
                else
                        tempo           <= guarismos(3 downto 0);
                        display <=      "1000";
                end if;
        end if;
    end process;
    output <=X"C0" when(tempo = X"0") else
                X"F9" when(tempo=X"1") else
                X"A4" when(tempo=X"2") else
                X"B0" when(tempo=X"3") else
                X"99" when(tempo=X"4") else
                X"92" when(tempo=X"5") else
                X"82" when(tempo=X"6") else
                X"F8" when(tempo=X"7") else
                X"80" when(tempo=X"8") else
                X"90" when(tempo=X"9") else
                X"88" when(tempo=X"A") else
                X"83" when(tempo=X"B") else
                X"C6" when(tempo=X"C") else
                X"A1" when(tempo=X"D") else
                X"86" when(tempo=X"E") else
              X"8E";
end architecture;
```

Diagrama



Simulación



Disp7segMultiplexor

Este código únicamente representa el multiplexor del circuito anterior. Ha sido dividido porque es necesario para poder ser simulado en proteus, ya que el número de pines es insuficiente para los PDLs disponibles en proteus.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;          --La necesitamos para el counter
use ieee.std_logic_unsigned.all;--tenemos que especificar que no tienen signo
entity Disp7segMultiplexor is
        port(
        clkIn   :       in      std_logic;
        tempo :         out     std_logic_vector(3 downto 0);
        display :       out std_logic_vector(3 downto 0)
        );
```

```
end entity;
architecture simple of Disp7SegMultiplexor is
signal guarismos     :        std_logic_vector(15 downto 0) := "1010101111001101";
signal cnt                    :        std_logic_vector(1 downto 0);
begin
        process(clkIn)
        begin
                if (clkIn 'event and clkIn = '1') then
                        if(cnt = "11") then     --Evitamos que se desborde y reiniciamos
                                cnt <= "00";    --la cuenta
                        end if;
                        cnt<=cnt+"01";
                        if(cnt = "00") then
                                tempo           <=      guarismos(15 downto 12);
                                display <=      "0001";
                        elsif(cnt = "01") then
                                tempo           <= guarismos(11 downto 8);
                                display <=      "0010";
                        elsif(cnt = "10") then
                                tempo           <= guarismos(7 downto 4);
                                display <=      "0100";
                        else
                                tempo           <= guarismos(3 downto 0);
                                display <=      "1000";
                        end if;
                end if;
        end process;
end architecture;
```

Simulación

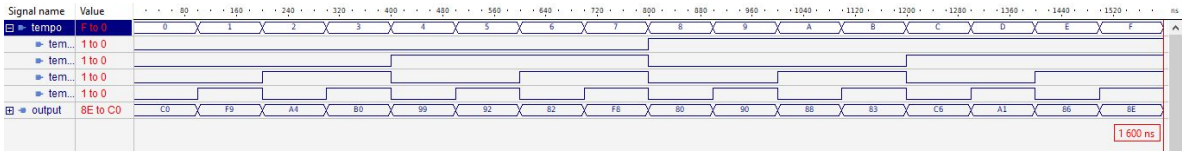
Disp7segDecodificador

Esta es la parte del decodificador. Solo recibe los guarismos de tempo y manda los números decodificados al display.

Código

```
library ieee;
use ieee.std_logic_1164.all;
entity Disp7segDecodificador is
	port(
	tempo :		in	std_logic_vector(3 downto 0);
	output :	out	std_logic_vector(7 downto 0)
	);
end entity;
architecture simple of Disp7segDecodificador is
begin
	output <=X"C0" when(tempo = X"0") else
			X"F9" when(tempo=X"1") else
			X"A4" when(tempo=X"2") else
			X"B0" when(tempo=X"3") else
			X"99" when(tempo=X"4") else
			X"92" when(tempo=X"5") else
			X"82" when(tempo=X"6") else
			X"F8" when(tempo=X"7") else
			X"80" when(tempo=X"8") else
			X"90" when(tempo=X"9") else
			X"88" when(tempo=X"A") else
			X"83" when(tempo=X"B") else
			X"C6" when(tempo=X"C") else
			X"A1" when(tempo=X"D") else
			X"86" when(tempo=X"E") else
		X"8E";
end architecture;
```

Simulación



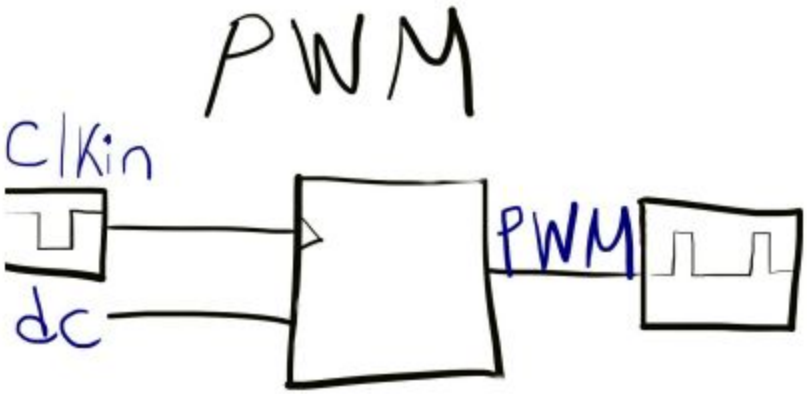
PWM

Un PWM (Pulse-width modulation) como su nombre lo dice, sirve para modular una señal. Es decir, controlar el ciclo de trabajo de una señal periódica.

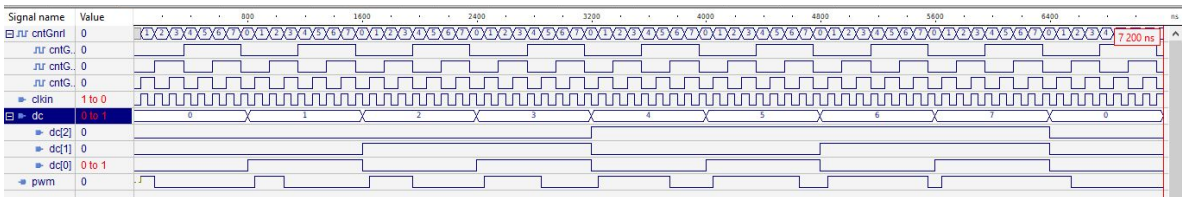
Código

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity PWM is
	port(
	clkin	:	in	std_logic;	--Reloj de entrada
	dc		:	in	std_logic_vector(2 downto 0); --Ciclo de trabajo
(duty Cycle)
	pwm		:	out std_logic --Señal PWM
	);
end entity;
architecture simple of PWM is
signal cntGnrl	:	std_logic_vector(2 downto 0):= "000";
begin
	process(clkin)
	begin
		if(clkin 'event and clkin='1') then
			if(dc >= cntGnrl) then
				PWM <= '1';
			else
				PWM <= '0';
			end if;
			if(cntGnrl >= "111") then
				cntGnrl <= "000";
			end if;
			cntGnrl <= cntGnrl+"001";
		end if;
	end process;
end architecture;
```

Diagrama



Simulación



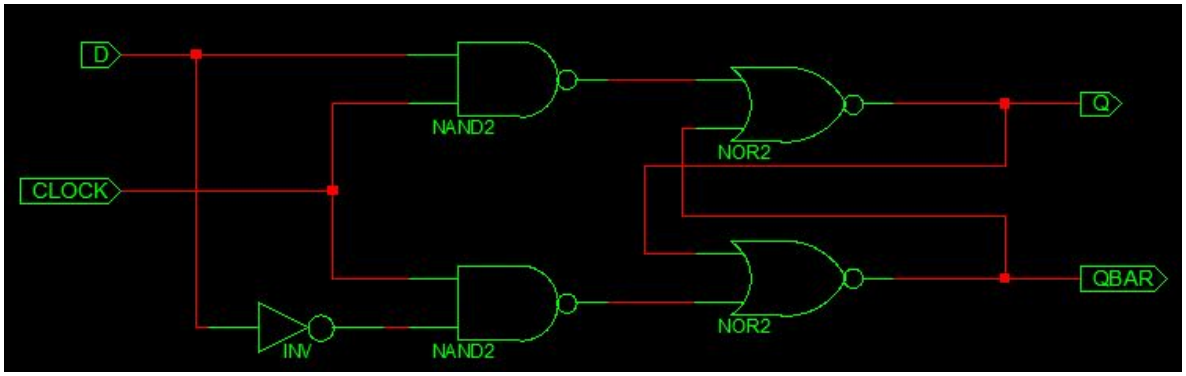
FlipFlopD

Un flip flop es la forma más básica de almacenamiento. Reacciona a un reloj de entrada y el estado cambia o se mantiene dependiendo de sus entradas. Como salida muestra el estado presente del flip flop y el estado presente negado

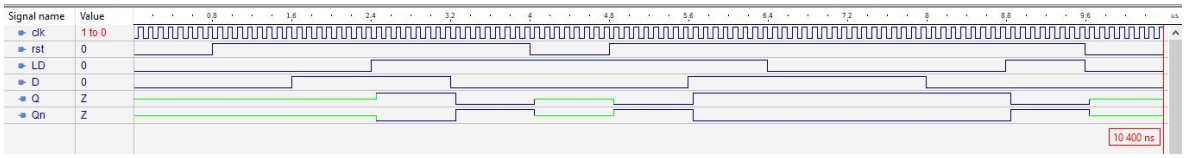
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity FlipFlopD is
	port(
	clk	:	in	std_logic; --Clock
	rst	:	in	std_logic; --reset
	LD	:	in	std_logic; --load
	D	:	in	std_logic; --Estado siguiente
	Q	:	out	std_logic; --Estado presente
	Qn	:	out	std_logic --estado presente negado
	);
end entity;
architecture behavioural of FlipFlopD is
begin
	process(clk)
	begin
		if rst='1'	then
			if clk 'event and clk='1' then
				if ld='1' then
					Q	<= D;
					Qn	<= not(D);
				end if;
			end if;
		else
			Q	<='Z';
			Qn	<='Z';
		end if;
	end process;
end architecture;
```

Diagrama



Simulación



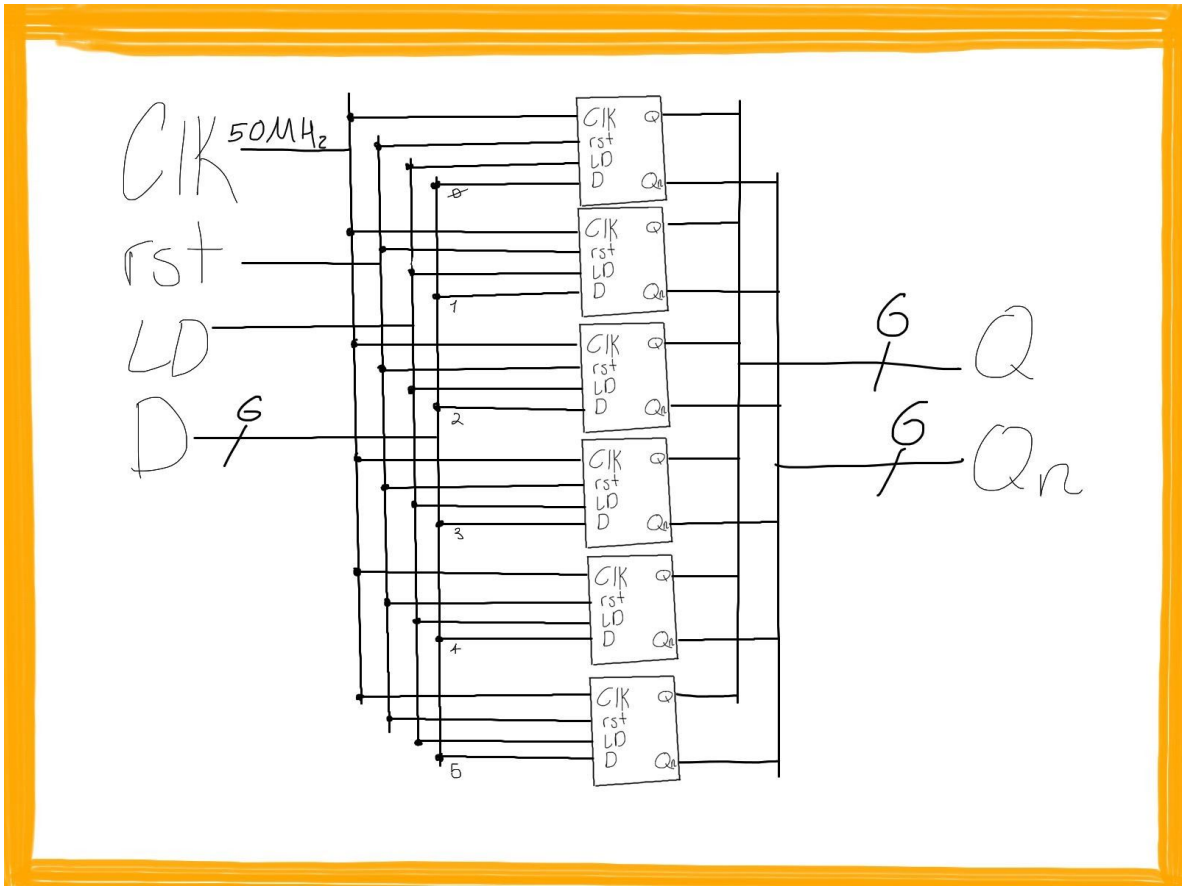
FlipFlopDmap

El siguiente circuito describe un flip flop tipo D de 6 bits creado con mapeo.

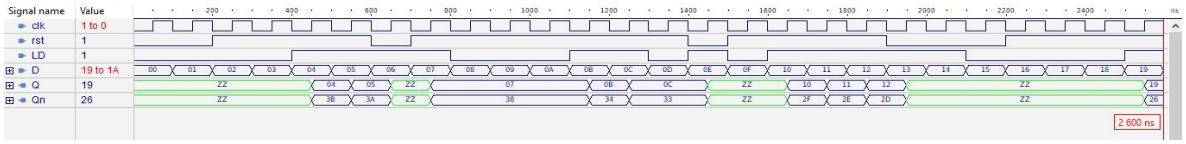
Código

```
library ieee;
use ieee.std_logic_1164.all;
entity FlipFlopDmap is
	port(
	clk	:	in	std_logic;--Clock
	rst	:	in	std_logic;
	LD	:	in	std_logic;
	D	:	in	std_logic_vector(5 downto 0);
	Q	:	out	std_logic_vector(5 downto 0);
	Qn	:	out	std_logic_vector(5 downto 0)
	);
end entity;
architecture behavioural of FlipFlopDmap is
component FlipFlopD is
	port(
	clk	:	in	std_logic; --Clock
	rst	:	in	std_logic; --reset
	LD	:	in	std_logic; --load
	D	:	in	std_logic; --Estado siguiente
	Q	:	out	std_logic; --Estado presente
	Qn	:	out	std_logic --estado presente negado
	);
end component;
begin
	bloque0: FlipFlopD port map(clk,rst,LD,D(0),Q(0),Qn(0));
	bloque1: FlipFlopD port map(clk,rst,LD,D(1),Q(1),Qn(1));
	bloque2: FlipFlopD port map(clk,rst,LD,D(2),Q(2),Qn(2));
	bloque3: FlipFlopD port map(clk,rst,LD,D(3),Q(3),Qn(3));
	bloque4: FlipFlopD port map(clk,rst,LD,D(4),Q(4),Qn(4));
	bloque5: FlipFlopD port map(clk,rst,LD,D(5),Q(5),Qn(5));
end architecture;
```

Diagrama



Simulación



ALU

La unidad aritmético-lógica se encarga de procesar todas las operaciones que llegan al CPU. Se accede a las operaciones disponibles de la ALU por medio de de la señal de entrada sel donde los dos primeros bits son para seleccionar la tabla de operaciones y los restantes para la operación en sí. Las señales de entrada A y B serán los vectores a procesar y la salido O el resultado.

Código

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ALU is
	port(
	sel:	in	std_logic_vector(5 downto 0);
	A,B:	in	std_logic_vector(5 downto 0);
	O:		out std_logic_vector(5 downto 0)
	);
end entity;


architecture simple of ALU is
begin

	with sel select
	O<=
			--Aritméticas Cn = 'L' M ='L' Tabla 2
			A - 1
								when("000000"),
			A(2 downto 0)*B(2 downto 0) - 1
				when("000001"),
			A(2 downto 0)*not(B(2 downto 0)) - 1
				when("000010"),
			(others => '1')
								when("000011"),	--así
representamos el -1
			A + (A + not(B))
							when("000100"),
```

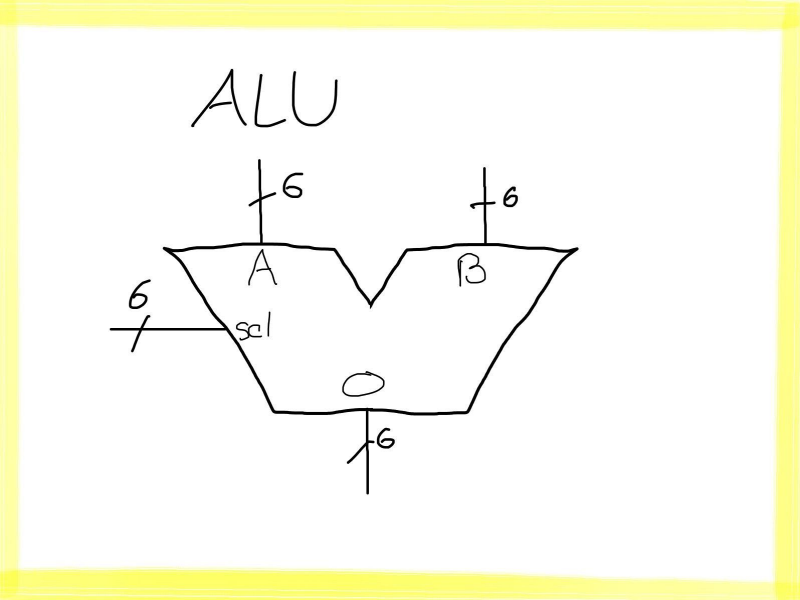
```
                A(2 downto 0)*B(2 downto 0) + (A(2 downto 0) + not(B(2
downto 0)))   when("000101"),
                A - B - 1
                                        when("000110"),
                A + not(B)
                                        when("000111"),
                A + (A + B)
                                        when("001000"),
                A + B
                                                when("001001"),
                A(1 downto 0) * not(B(1 downto 0)) * (A(1 downto 0)+B(1
downto 0))   when("001010"),
                A + B
                                        when("001011"),
                A + A
                                        when("001100"),
                A(2 downto 0) * B(2 downto 0) + A(2 downto 0)
                when("001101"),
                A(2 downto 0) * not(B(2 downto 0)) + A(2 downto 0)
                when("001110"),
                A
                                                when("001111"),

                --Lógicas Cn = 'L' M = 'H' Tabla 2
                not(A)                          when("010000"),
                not(A and B)            when("010001"),
                not(A) or B                     when("010010"),
                "000001"                        when("010011"),
                not (A or B)            when("010100"),
                not(B)                          when("010101"),
                not(A xor B)            when("010110"),
                A or not(B)                     when("010111"),
                not(A) and B            when("011000"),
                not(A xor B)            when("011001"),
                B                               when("011010"),
                A or B                          when("011011"),
                (others => '0')                 when("011100"),
                A and not(B)            when("011101"),
                A and B                         when("011110"),
```

```
                    A                                   when("011111"),

                    --Aritméticas Cn = 'H' M ='L' Tabla 1
                    A
                                                        when("100000"),
                    B-A
                                                        when("100001"),
                    A+not(b)
                                                when("100010"),
                    "111111"
                                                when("100011"),
                    A + (A + not(B))
                                            when("100100"),
                    A(2 downto 0)*B(2 downto 0) + (A(2 downto 0) + not(B(2
downto 0)))    when("100101"),
                    A - B - 1
                                                when("100110"),
                    A(2 downto 0) * B(2 downto 0) - 1
                            when("100111"),
                    A(2 downto 0) + (A(2 downto 0) * B(2 downto 0))
                            when("101000"),
                    A + B
                                                        when("101001"),
                    A(2 downto 0) * not(B(2 downto 0)) + (A(2 downto 0)*B(2
downto 0))    when("101010"),
                    A(2 downto 0) * B(2 downto 0) - 1
                            when("101011"),
                    A + A
                                                when("101100"),
                    A + B + A
                                                when("101101"),
                    A(2 downto 0) * not(B(2 downto 0)) + A(2 downto 0)
                    when("101110"),
                    A - 1
                                                when("101111"),

                    --Logicas Cn = 'H' M = 'H' Tabla 1
                    not(A)                              when("110000"),
                    not(A or B)             when("110001"),
```

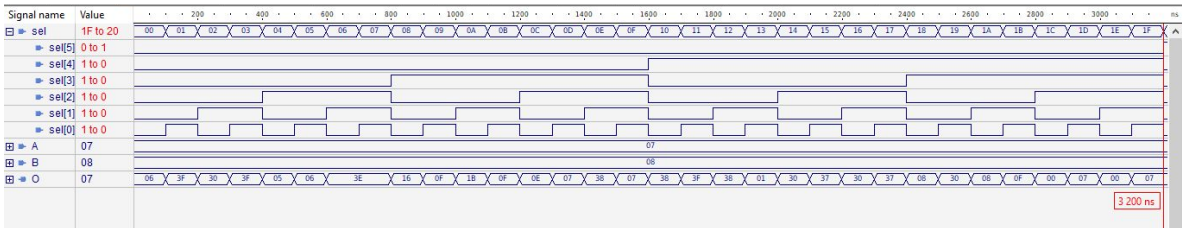
```
((not(A)) and B)      when("110010"),
(others => '0')             when("110011"),
not(A and B)          when("110100"),
not(B)                      when("110101"),
A xor B                           when("110110"),
A and (not(B))              when("110111"),
not(A) or B                 when("111000"),
B                                 when("111001"),
A and B                           when("111010"),
"000001"                    when("111011"),
A or (not(B))         when("111100"),
A or B                      when("111101"),
A                                 when others;   --111111
```

end architecture;

Diagrama



Simulación



GCM

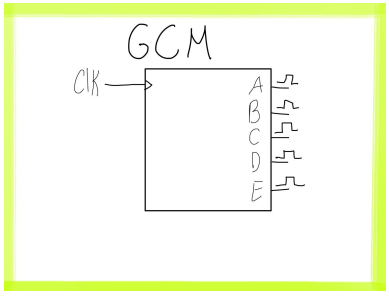
La gcm se encarga de sincronizar todos los módulos del cpu. Manda señales a varios módulos que deben actuar unos seguidos de otros.

Código

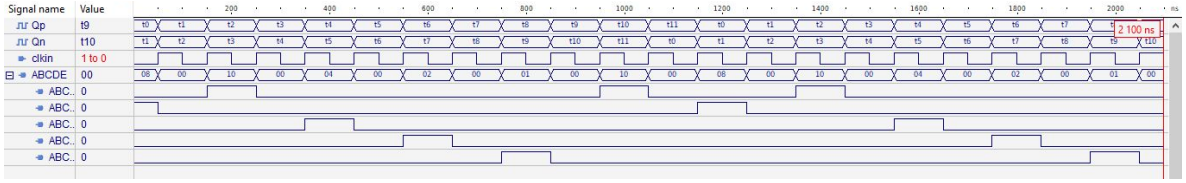
```
library ieee;
use ieee.std_logic_1164.all;
entity GCM is
	port(
		clkin:		in		std_logic;	-- 50 MHz Clock
		ABCDE:			out		std_logic_vector(4 downto 0)
	);
end entity;
architecture behavioral of  GCM is
type estados is (T0,T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11);
signal Qp, Qn:	estados;  -- Qp: estado actual, Qn: estado siguiente
begin
	P1: process(clkin)
	begin
		if clkin 'event and clkin ='1' then
			Qp <= Qn;
		end if;
	end process;
	P2: process(Qp)
	begin
		case Qp is
			when T0 =>
				ABCDE <= "01000";
				Qn <= T1;
			when T1 =>
				ABCDE <= "00000";
				Qn <= T2;
			when T2 =>
				ABCDE <= "10000";
				Qn <= T3;
			when T3 =>
				ABCDE <= "00000";
				Qn <= T4;
			when T4 =>
```

```
                ABCDE <= "00100";
                Qn <= T5;
            when T5 =>
                ABCDE <= "00000";
                Qn <= T6;
            when T6 =>
                ABCDE <= "00010";
                Qn <= T7;
            when T7 =>
                ABCDE <= "00000";
                Qn <= T8;
            when T8 =>
                ABCDE <= "00001";
                Qn <= T9;
            when T9 =>
                ABCDE <= "00000";
                Qn <= T10;
            when T10 =>
                ABCDE <= "10000";
                Qn <= T11;
            when T11 =>
                ABCDE <= "00000";
                Qn <= T0;
        end case;
    end process;
end architecture;
```

Diagrama



Simulación



ContProg

El contador de programa nos dará las direcciones de memoria de las instrucciones que la memoria de programa enviará a la alu.

Código

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity ContProg is
	port(
		clk:		in		std_logic;
		addr:		out		std_logic_vector(3 downto 0):=(others
=>'0')
	);
end entity;
architecture behavioral of ContProg is
signal cntP,cntS:		std_logic_vector(3 downto 0):=(others =>'0');
begin
	process(clk)
	begin
		if clk 'event and clk='1' then		-- rising_edge(clk)
			addr <=  cntP;
			cntP <= cntP+1;
		end if;
	end process;
end architecture;
```

Simulación

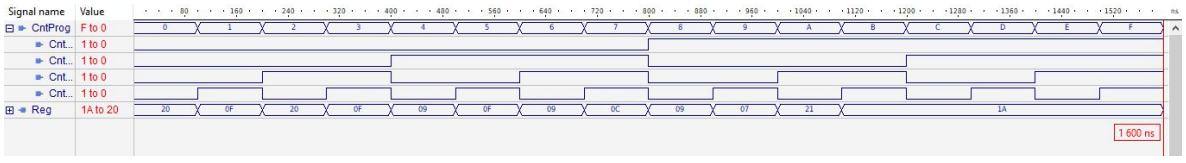
memoria

La memoria de programa contiene las instrucciones y datos que serán enviados a la ALU para ser procesados. El dato de salida dependerá de la dirección enviada por el contador de programa

Código

```
library ieee;
use ieee.std_logic_1164.all;
entity memoria is
	port(
	CntProg:	in	std_logic_vector(3 downto 0);
	Reg:		out std_logic_vector(5 downto 0)
	);
end entity;
architecture simple of memoria is
begin
	with CntProg select
	Reg	<=	"100000" when "0000",--carga A
			"001111" when "0001",		--15
			"100000" when "0010",--carga A
			"001111" when "0011",		--15
			"001001" when "0100",--suma A + B
			"001111" when "0101",		--15
			"001001" when "0110",--suma A + B
			"001100" when "0111",		--12
			"001001" when "1000",--suma A + B
			"000111" when "1001",		--7
			"100001" when "1010",--resta B - A
			"011010" when others;
end architecture;
```

Simulación

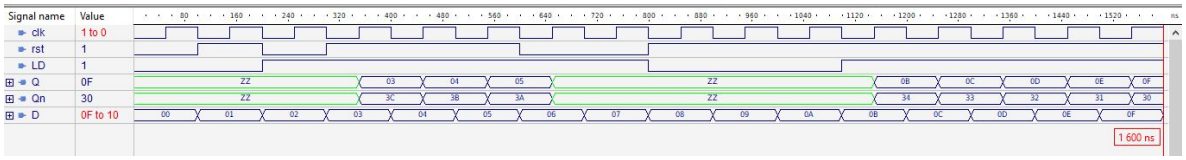


FlipFlopd6bits

El flip flop descrito por el siguiente circuito fue el usado para la práctica del procesador. A diferencia de los otros flip flops anteriores, este recibe un vector de 6 bits de entrada.

```
library ieee;
use ieee.std_logic_1164.all;
entity FlipFlopD6bits is
	port(
	clk	:	in	std_logic; --Clock
	rst	:	in	std_logic; --reset
	LD	:	in	std_logic; --load
	D	:	in	std_logic_vector(5 downto 0); --Estado siguiente
	Q	:	out	std_logic_vector(5 downto 0); --Estado presente
	Qn	:	out	std_logic_vector(5 downto 0) --estado presente negado
	);
end entity;
architecture behavioural of FlipFlopD6bits is
begin
	process(clk)
	begin
		if rst='1'	then
			if clk 'event and clk='1' then
				if ld='1' then
					Q	<= D;
					Qn	<= not(D);
				end if;
			end if;
		else
			Q	<=(others => 'Z');
			Qn	<=(others => 'Z');
		end if;
	end process;
end architecture;
```

Simulación



Procesador6bits

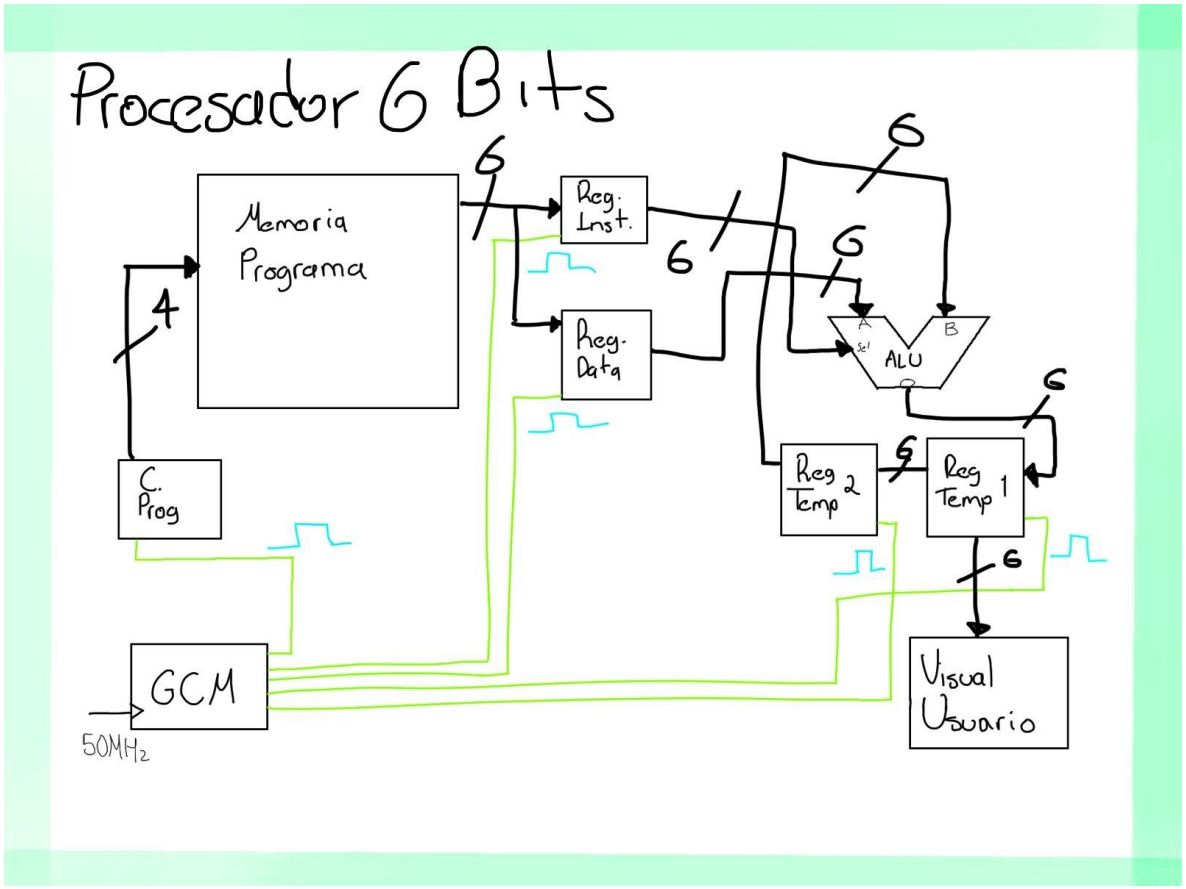
El circuito siguiente es donde se implementan todos los códigos anteriores para dar funcionamiento al procesador. Como entrada recibe un reloj y la salida un vector de 6 bits.

Código

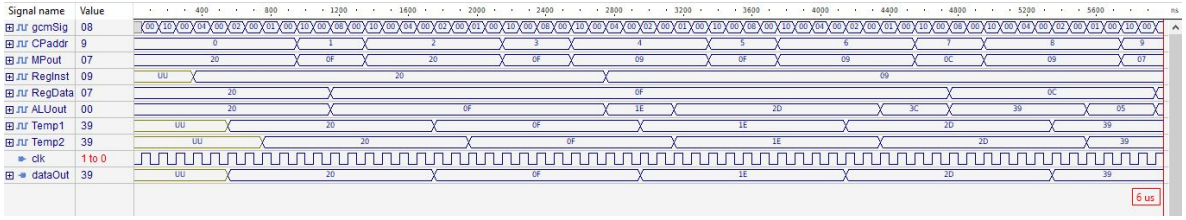
```
library ieee;
use ieee.std_logic_1164.all;
entity Procesador6bits is
	port(
	clk:		in	std_logic;
	dataOut:	out	std_logic_vector(5 downto 0)
	);
end entity;
architecture simple of Procesador6bits is
component GCM is
	port(
		clkin:			in		std_logic;	-- 50 MHz Clock
		ABCDE:				out		std_logic_vector(4 downto 0)
	);
end component;
component ContProg is
	port(
		clk:		in		std_logic;
		addr:		out		std_logic_vector(3 downto 0):=(others
=>'0')
	);
end component;
component FlipFlopD6bits is
	port(
	clk	:	in	std_logic; --Clock
	rst	:	in	std_logic; --reset
	LD	:	in	std_logic; --load
	D	:	in	std_logic_vector(5 downto 0); --Estado siguiente
	Q	:	out	std_logic_vector(5 downto 0); --Estado presente
	Qn	:	out	std_logic_vector(5 downto 0) --estado presente negado
	);
end component;
component memoria is
```

```
        port(
        CntProg:    in      std_logic_vector(3 downto 0);
        Reg:        out std_logic_vector(5 downto 0)
        );
end component;
component ALU is
        port(
        sel:    in      std_logic_vector(5 downto 0);
        A,B:    in      std_logic_vector(5 downto 0);
        O:              out std_logic_vector(5 downto 0)
        );
end component;
signal gcmSig:      std_logic_vector(4 downto 0):=(others => '0');
signal CPaddr:      std_logic_vector(3 downto 0):=(others => '0');
signal MPout, RegInst, RegData, ALUout, Temp1, Temp2:    std_logic_vector(5
downto 0):=(others => '0');
begin
        bloque0: GCM                        port map(clk, gcmSig);
        bloque1: FlipFlopD6bits port map(gcmSig(2),'1','1',MPout,RegInst, open);
        bloque2: ContProg           port map(gcmSig(4),CPaddr);
        bloque3: memoria            port map(CPaddr, MPout);
        bloque4: FlipFlopD6bits port map(gcmSig(3),'1','1',MPout,RegData, open);
        bloque5: FlipFlopD6bits port map(gcmSig(1),'1','1',ALUout,Temp1, open);
        bloque6: FlipFlopD6bits port map(gcmSig(0),'1','1',Temp1,Temp2, open);
        bloque7: ALU                        port
map(RegInst,RegData,Temp2,ALUout);
        dataOut <= Temp1;
end architecture;
```

Diagrama



Simulación



LCD

Una LCD (Liquid Crystal Display) es bastante útil para representar datos alfanuméricos. El siguiente código sirve para poder mostrar datos en una LCD de 16*2. Se hace uso de una máquina de estados, pues requiere que se hagan ciertas operaciones antes de empezar a escribir datos en ella.

Código

```
library ieee;
use ieee.std_logic_1164.all;
entity LCD is port(
	clk:	in		std_logic;
	ms:		in		std_logic; -- 0 es 4 bits: 1 es 8 bits
	rs, rw: out	std_logic;
	DB:		inout	std_logic_vector(7 downto 0)
	);
end entity;
architecture behaviorual of LCD is
type estados is (CLEARED, RETURNH, ENTRYMS, DISPLAYC, CURSORDS,
FUNCTIONS, SETCGRAM, WRITECGRAM, READBF);
signal epres, esig, euins:	estados;
signal flag: std_logic:='0'; --dirá que nibble enviar(0: MSN, 1_LSN)
begin
	Comb: process(clk)
	begin
		if clk'event and clk='1' then
			epres <= esig;
		end if;
	end process;
	Sseq: process(epres)
	begin
		case epres is
			when CLEARED =>
				rs <= '0';
				rw <= '0';
				if ms = '1' then	--8 bits
					DB <= X"01";
					esig <= RETURNH;
				else				--4 bits
					euins <= CLEARED;
```

```
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"10";
                        esig <= RETURNH;
                end if;
        end if;
when RETURNH =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"02";
                esig <= ENTRYMS;
        else                            --4 bits
                euins <= RETURNH;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"20";
                        esig <= ENTRYMS;
                end if;
        end if;
when ENTRYMS =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"03";
                esig <= DISPLAYC;
        else                            --4 bits
                euins <= ENTRYMS;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"30";
                        esig <= DISPLAYC;
                end if;
```

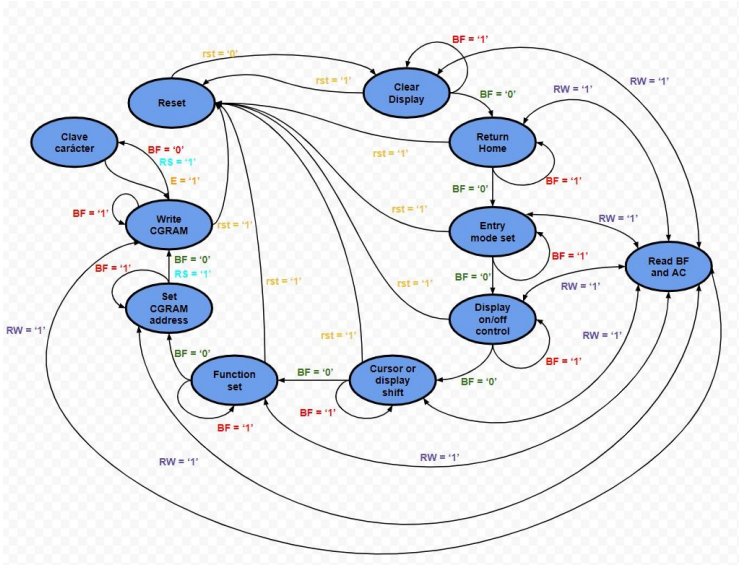
```
        end if;
when DISPLAYC =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"04";
                esig <= CURSORDS;
        else                            --4 bits
                euins <= DISPLAYC;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"40";
                        esig <= CURSORDS;
                end if;
        end if;
when CURSORDS =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"04";
                esig <= FUNCTIONS;
        else                            --4 bits
                euins <= CURSORDS;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"40";
                        esig <= FUNCTIONS;
                end if;
        end if;
when FUNCTIONS =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"05";
                esig <= SETCGRAM;
```

```
        else                            --4 bits
                euins <= FUNCTIONS;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"50";
                        esig <= SETCGRAM;
                end if;
        end if;
when SETCGRAM =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"06";
                esig <= WRITECGRAM;
        else                            --4 bits
                euins <= SETCGRAM;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"60";
                        esig <= WRITECGRAM;
                end if;
        end if;
when WRITECGRAM =>
        rs <= '0';
        rw <= '0';
        if ms = '1' then        --8 bits
                DB <= X"07";
                esig <= READBF;
        else                            --4 bits
                euins <= WRITECGRAM;
                if flag='0' then --nibble nibble
                        DB <= X"00"; -- 0000 XXXX
                        esig <= READBF;
                else
                        DB <= X"70";
```

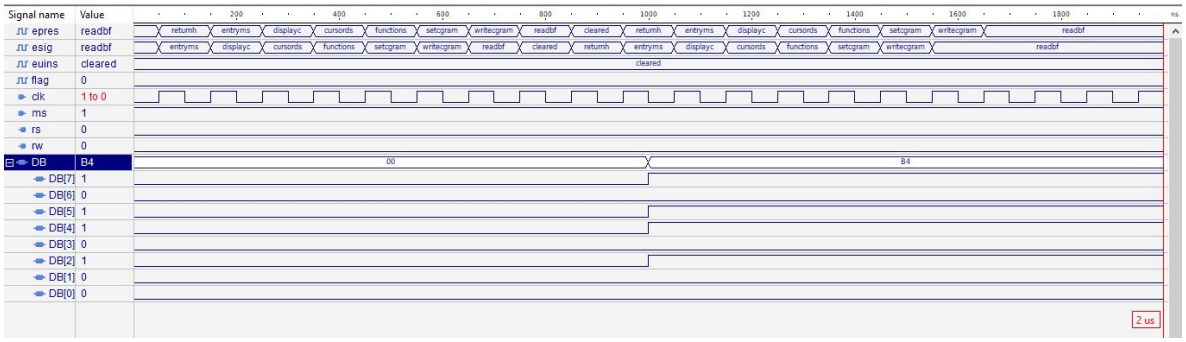
```
                                        esig <= READBF;
                                    end if;
                                end if;
                            when others =>
                            if DB(7) ='0' then
                                esig <= euins;
                            else
                                esig <= READBF;
                            end if;
                            end case;
                        end process;
end architecture;
```

Diagrama (máquina de estados)

figura cortesía de: Jaime Rodrigo Gonzáles Rodríguez.



Simulación



GLCD_128x64

El driver presentado nos permitirá inicializar nuestra GLCD y empezar a dibujar en ella. Se hace uso de una máquina de estados para inicializar, establecer posiciones en ‘x’ y ‘y’ y finalmente la escritura.

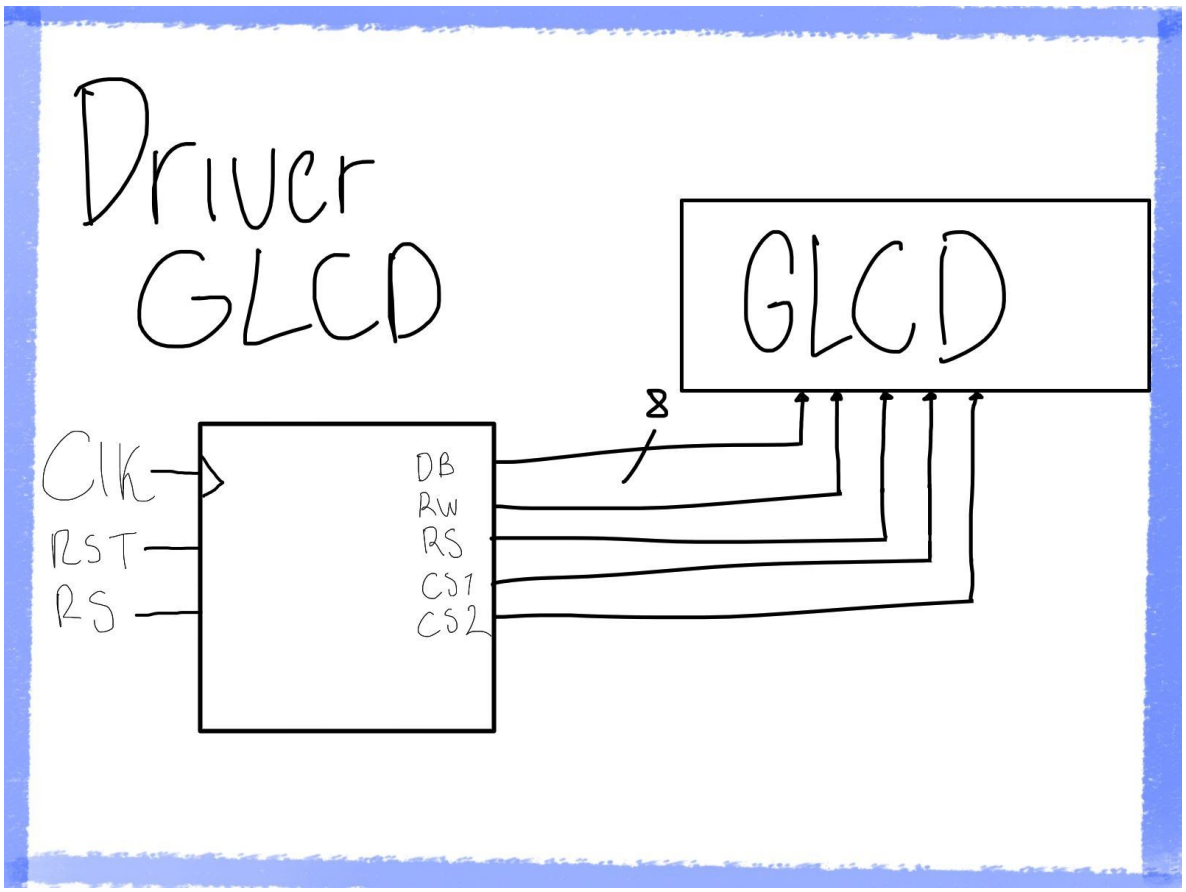
Código

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity GLCD_128x64 is
	port(
		clk	: in std_logic; --Reloj Físico
		rst	: in std_logic := '1'; --Reset General
		RS		: inout std_logic := '0'; --RS
		RW		: out std_logic := '0'; --RW
		CS1	: out std_logic := '1'; --Chip1
		CS2	: out std_logic := '1'; --Chip2
		DB		: out std_logic_vector(7 downto 0) := "00111111"
--Datos
	);
end entity;
architecture behaivoral of GLCD_128x64 is
type estados is (DISP, SET_Y, SET_X, WRITE);
signal epres, esig : estados;
signal start : std_logic := '0';
signal changePage : std_logic := '0'; --Señal para determinar el cambio de página
signal cntX : std_logic_vector(2 downto 0) := (others => '0'); --Paginas
signal cntY : std_logic_vector(6 downto 0) := (others => '0'); --Direcciones en Y
begin
	contadores: process(clk)
	begin
		if(rst = '0')then
			cntX <= (others => '0');
			cntY <= (others => '0');
		else
			 if(clk 'event and clk = '1')then
				if(start = '1')then
```

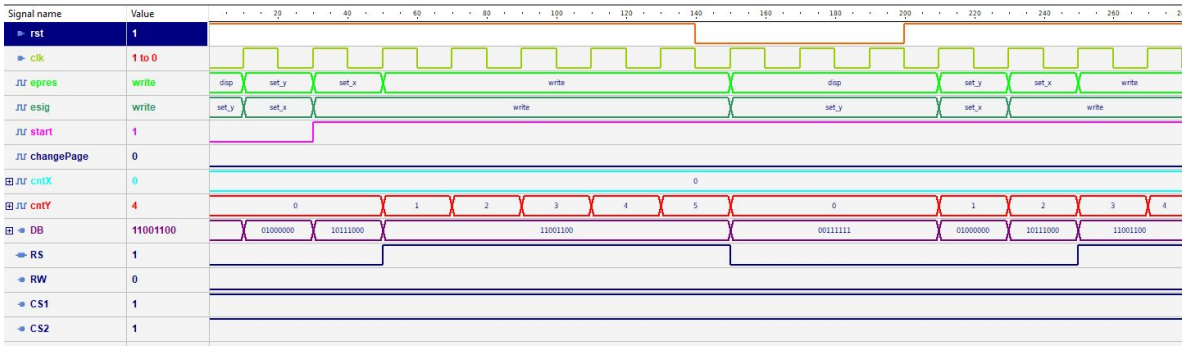
```
                        if(cntY < "1000000")then
                                changePage <= '0';
                                cntY <= cntY + 1;
                        else
                                changePage <= '1';
                                cntX <= cntX + 1;
                                cntY <= (others => '0');

                        end if;
                end if;
        end if;
    end if;
end process contadores;
--Asignación de estado de la máquina de estados
comb : process( clk )
begin
    if clk 'event and clk = '1' then
        if( rst = '1')then
            epres <= esig;
        else
            epres <= DISP;
        end if;
    end if;
end process;
seq : process( epres, changePage )
begin
    case epres is
        when DISP =>
            RS <= '0';
            RW <= '0';
            DB <= "00111111"; --Comando
            esig <= SET_Y;
        when SET_Y =>
            RS <= '0';
            RW <= '0';
            DB <= "01000000"; --Y = 0;
            esig <= SET_X;
        when SET_X =>
            RS <= '0';
```

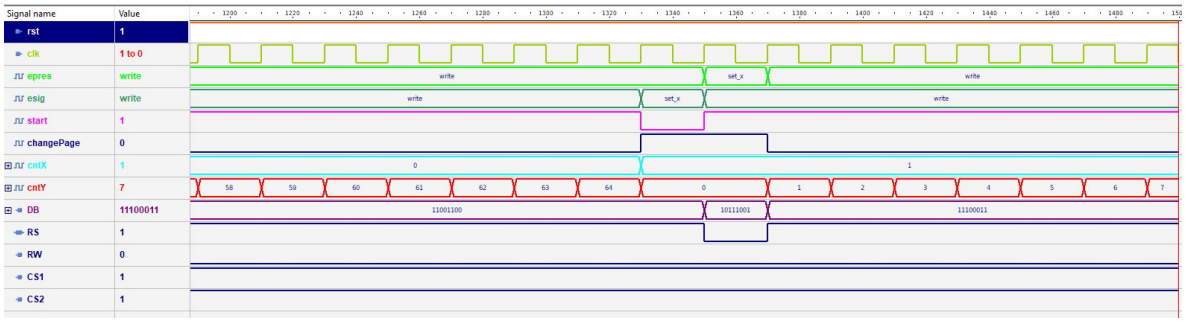
```
                        RW <= '0';
                        start <= '1';
                        DB <= "10111"&cntX;
                        esig <= WRITE;
                    when others =>
                        if(changePage = '0')then
                            RS <= '1';
                            RW <= '0';
                            case cntX is
                                when "000" =>
                                    DB <= "11001100";
                                when "001" =>
                                    DB <= "11100011";
                                when "010" =>
                                    DB <= "11110000";
                                when "011" =>
                                    DB <= "00001111";
                                when "100" =>
                                    DB <= "11101110";
                                when "101" =>
                                    DB <= "00010001";
                                when "110" =>
                                    DB <= "00000000";
                                when others =>
                                    DB <= "11111111";
                            end case;
                            esig <= WRITE;
                        else
                            start <= '0';
                            esig <= SET_X;
                        end if;
                end case;
        end process seq;
end architecture;
```

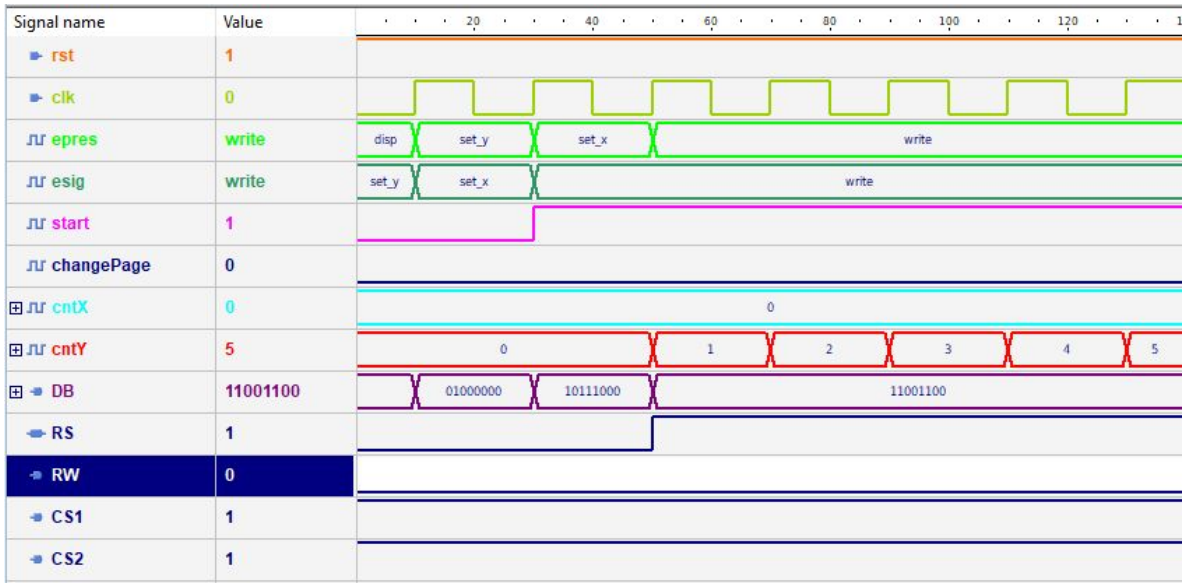
Diagrama



Simulación


Signal name
Value
rst 1
clk 1 to 0
epres write
esig write
start 1
changePage 0
cntX 0
cntY 4
DB 11001100
RS 1
RW 0
CS1 1
CS2 1


Signal name
Value
rst 1
clk 1 to 0
epres write
esig write
start 1
changePage 0
cntX 1
cntY 7
DB 11100011
RS 1
RW 0
CS1 1
CS2 1


Signal name
Value
rst 1
clk 0
epres write
esig write
start 1
changePage 0
cntX 0
cntY 5
DB 11001100
RS 1
RW 0
CS1 1
CS2 1
disp
set_y
set_x
write
01000000
10111000
11001100