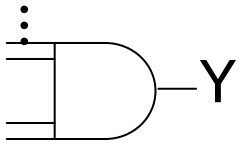
Puertas con múltiples entradas

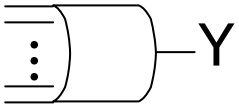
D(7 **downto** 0)



AND_Reduce

```
Y <= '1' when D=(7 downto 0=>'1') else '0';
```

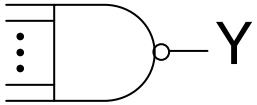
D(7 **downto** 0)



OR_Reduce

```
Y <= '0' when D=(7 downto 0=>'0') else '1';
```

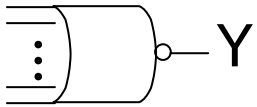
D(7 **downto** 0)



NAND_Reduce

```
Y <= '0' when D=(7 downto 0=>'1') else '1';
```

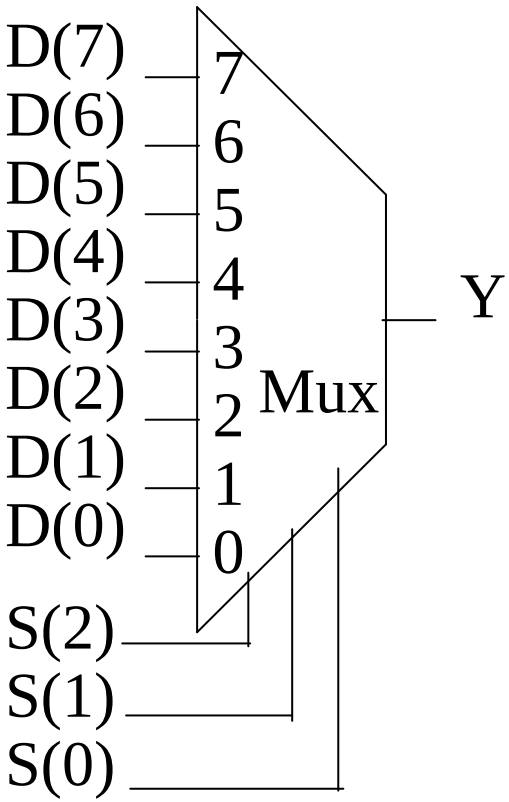
D(7 **downto** 0)



NOR_Reduce

```
Y <= '1' when D=(7 downto 0=>'0') else '0';
```

Multiplexor



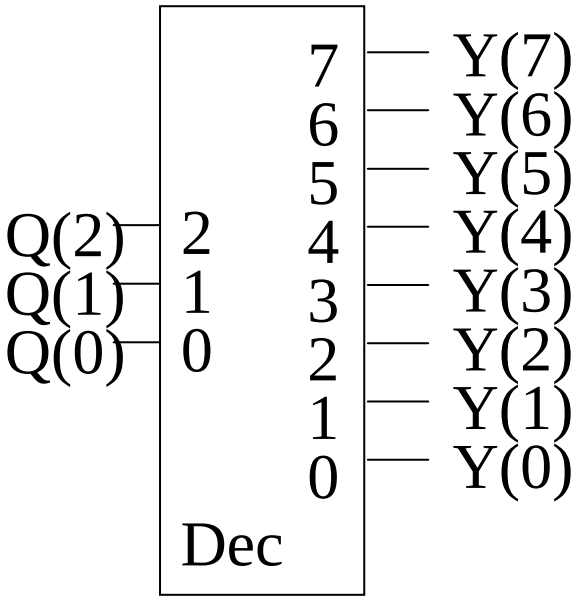

```
D: Std_Logic_vector(7 downto 0);
S: Std_Logic_vector(2 downto 0);
Y: Std_Logic;

        with S select
          Y <= D(0) when "000",
               D(1) when "001",
               D(2) when "010",
               D(3) when "011",
               D(4) when "100",
               D(5) when "101",
               D(6) when "110",
               D(7) when others;

      Y <= D(conv_integer(S));

-- usa IEEE.Std_Logic_unsigned.all;
```

Decoder



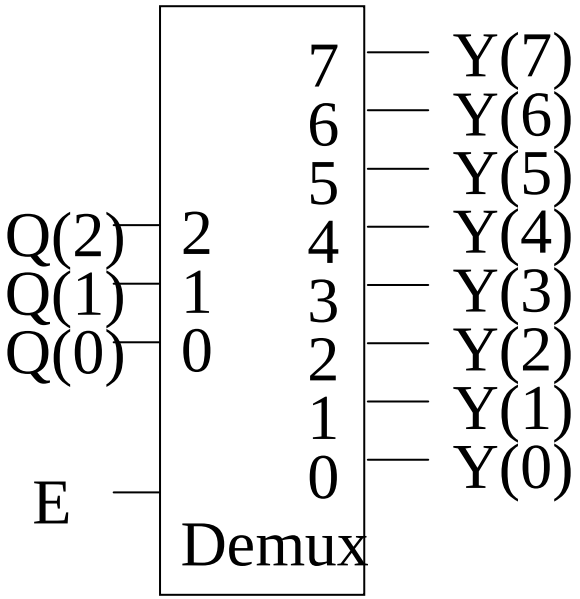

```
Q: Std_Logic_vector(2 downto 0);
Y: Std_Logic_vector(7 downto 0);

with Q select
  Y <= "00000001" when "000",
       "00000010" when "001",
       "00000100" when "010",
       "00001000" when "011",
       "00010000" when "100",
       "00100000" when "101",
       "01000000" when "110",
       "10000000" when others;
```

```
Y <= conv_std_logic_vector(2**(conv_integer(Q)) ,8);
```

```
--       usa    IEEE.Std_Logic_unsigned.all;
-- usa IEEE.Std_Logic_arith.all;
```

Decoder con enable



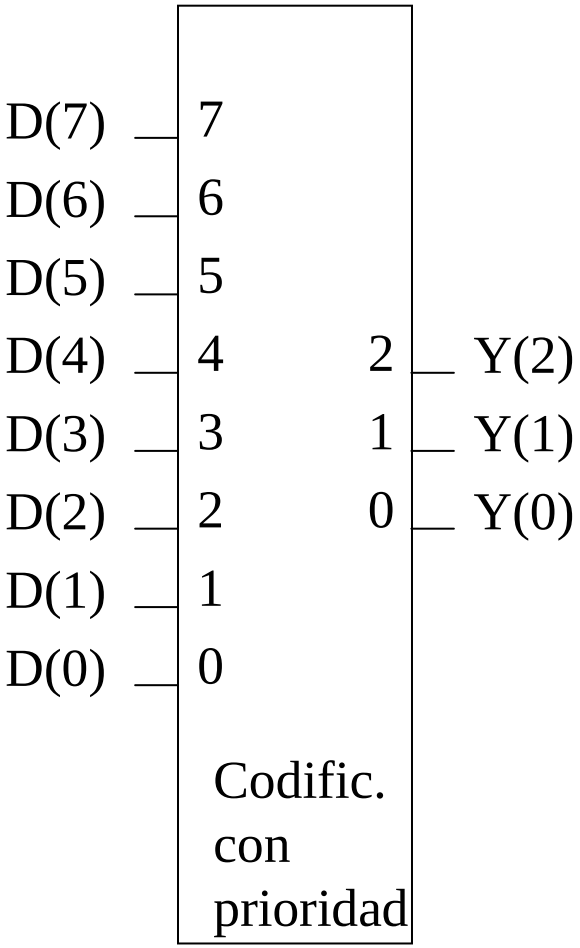

```
D: Std_Logic_vector(3 downto 0);

D <= E&Q; --concatenmos

with D select
 Y <= "00000001" when "1000",
      "00000010" when "1001",
      "00000100" when "1010",
      "00001000" when "1011",
      "00010000" when "1100",
      "00100000" when "1101",
      "01000000" when "1110",
      "10000000" when "1111",
      "00000000" when others;

Y <= Conv_Std_Logic_Vector(2**(Conv_integer(S)), 8)
            and (Y'range => E);
```

Codificador con prioridad

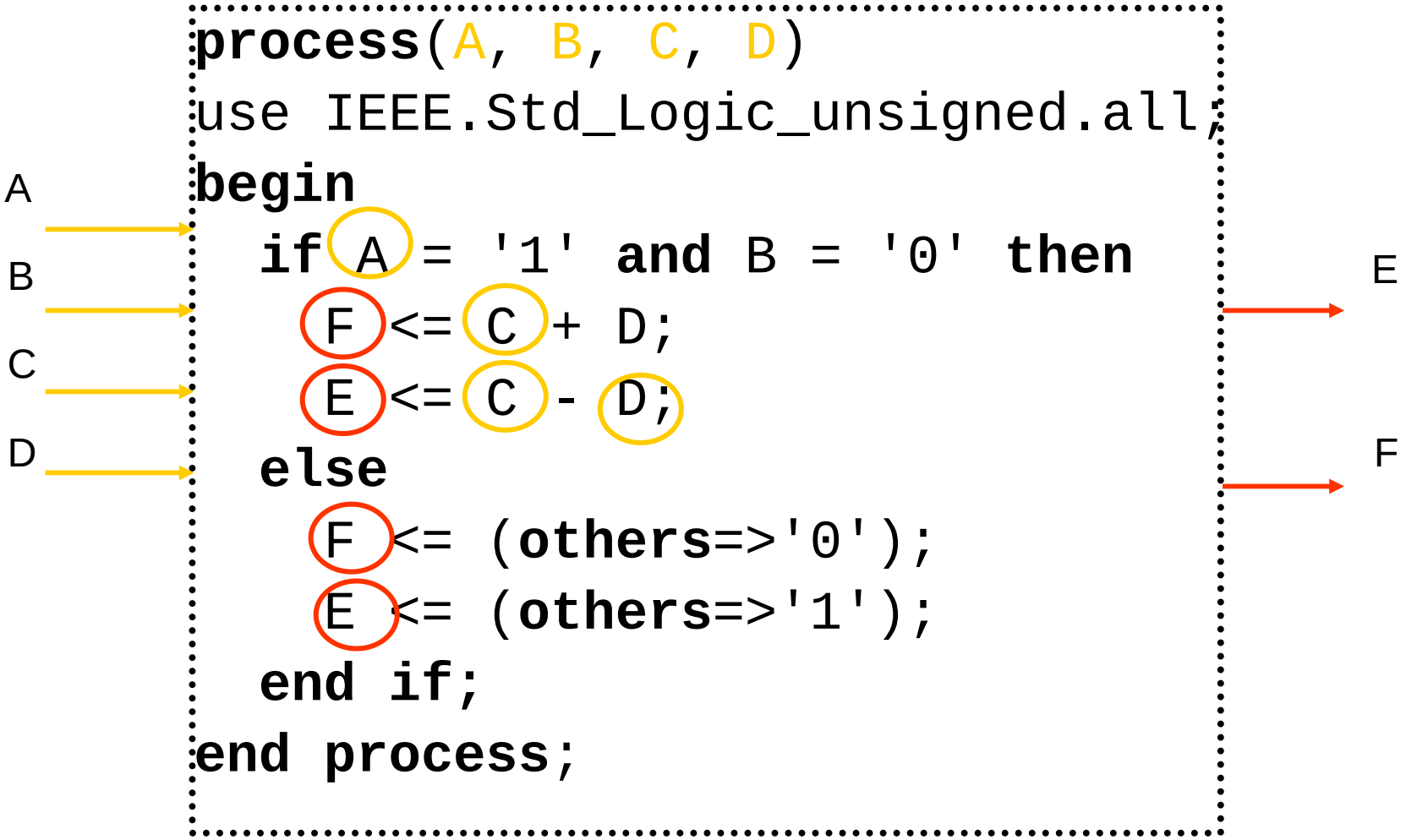



```
Y <= "111" when D(7) = '1' else
     "110" when D(6) = '1' else
     "101" when D(5) = '1' else
     "100" when D(4) = '1' else
     "011" when D(3) = '1' else
     "010" when D(2) = '1' else
     "001" when D(1) = '1' else
     "000";

process(D)
begin
  Y <= "000";--asignación por defecto
  for i in D'range loop
    if D(i)='1' then
      Y <= conv_std_logic_vector(i, 3);
      exit;
    end if;
  end loop;
end process;
```

Reglas par los procesos (1)

Los procesos Tienen entradas y salidas implícitas

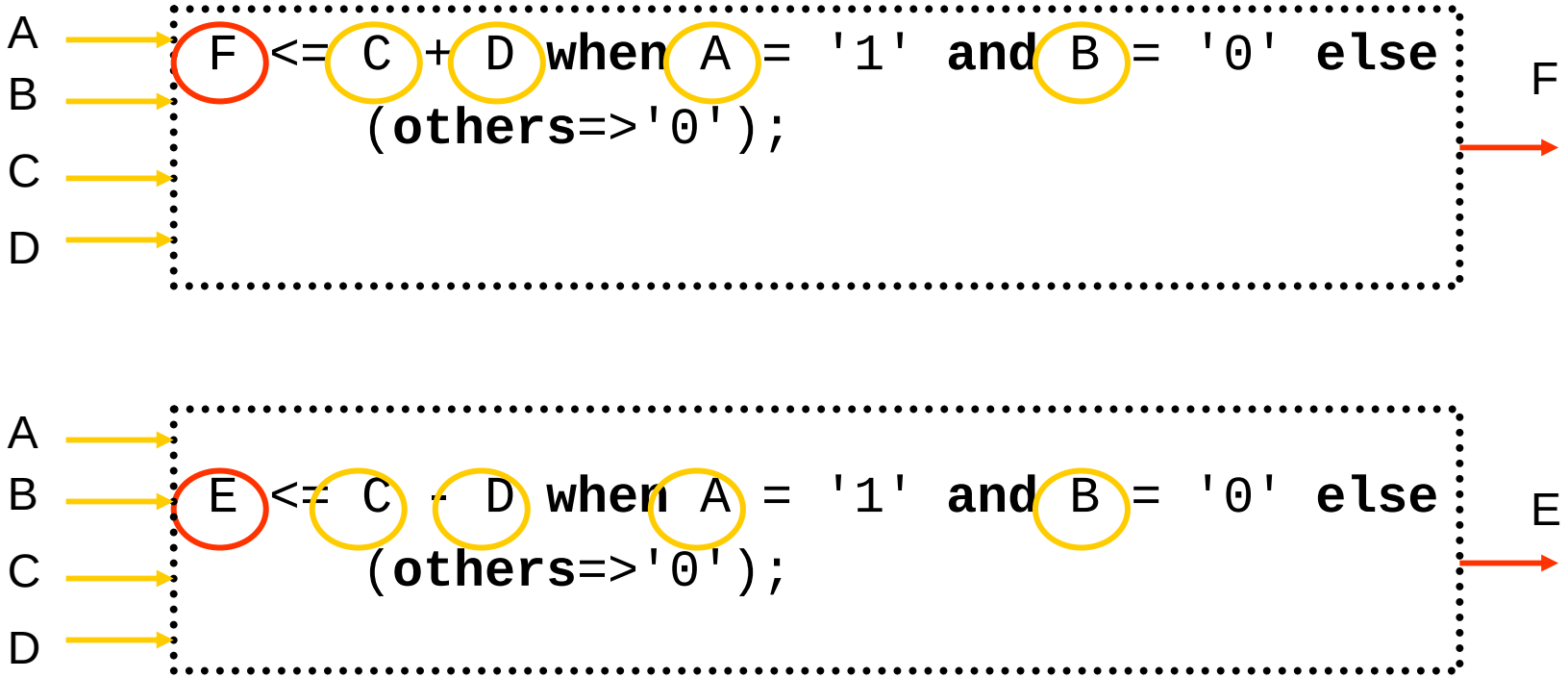



```
process(A, B, C, D)
use IEEE.Std_Logic_unsigned.all;
begin
  if A = '1' and B = '0' then
    F <= C + D;
    E <= C - D;
  else
    F <= (others=>'0');
    E <= (others=>'1');
  end if;
end process;
```

Note: Addition of vectors requires special package visibility

Reglas par los procesos (1)

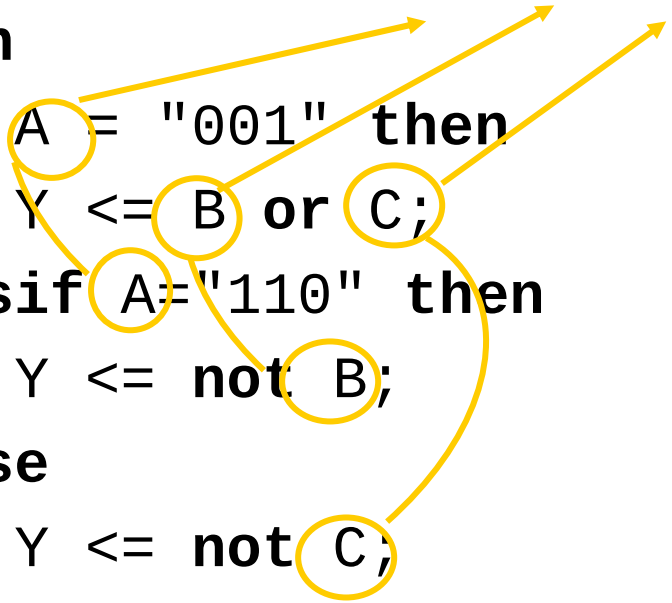
Los procesos deben ser particionado separadando las salidas



Reglas par los procesos Combinacionales (1)

Los P. Combinacionales deben tener todas las señales que son usadas en condiciones y a la derecha de las asignaciones o entradas en llamadas a procedimientos tienen que aparecer en la lista de sensibilidad

```
Comb_Proc:process(A, B, C)
begin
  if A = "001" then
     Y <= B or C;
  elsif A="110" then
     Y <= not B;
  else
     Y <= not C;
  end if;
end process Comb_Proc;
```



Reglas par los procesos Combinacionales(2)

Cuando usas if/then/elsif/else la asignación de Y debe estar en todos apartados

```
Comb_Proc: process(A, B, C)
     begin
       if A = "001" then
          Y <= B or C;
       elsif A="110" then
          Y <= not B;
       else
          Y <= not C;
       end if;
     end process Comb_Proc;
```

Reglas par los procesos Combinacionales(3)

la condición else debe ser cubierta.

```
Comb_Proc: process(A, B, C)
begin
  Y <= not C;
  if A = "001" then
     Y <= B or C;
  elsif A="110" then
     Y <= not B;
  end if;
end process Comb_Proc;
```

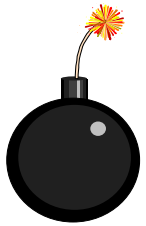
Reglas driver de una señal

Cualquier señal debe ser conducida solamente por un proceso.

```
architecture . . .
  signal F: Std_Logic;
begin
  Proc_1: process(A, B)
  begin
    F <= A and B;
  end process Proc_1;

  Proc_2: process(C, D)
  begin
    F <= C or D;
  end process Proc_2;
end;
```

F



Mas de un driver!!

Reglas Procesos Sincronizados (1)

El if/then de fuera debe incluir un asincrono set o reset esta condición debe aparecer antes de la condición de reloj.

```
Reg_Proc: process( Rst, Clk)
use IEEE.Std_Logic_unsigned.all;
begin
  if Rst = '1' then
    Cnt <= 0;
  elsif rising_edge(Clk) then
    if En = '1' then
      Cnt <= Cnt + 1;
    end if;
  end if;
end process Reg_Proc;
```

Evitar usar a la vez set y reset (limitaciones de tecnología).

Reglas procesos Sincronizados (2)

condiciones adicionales impide al la herramienta de sisntesiss reconocer el flanco de reloj. The outer if/then or wait statement should have only the clock condition.

```
Reg_Proc: process(Rst, Clk)
use IEEE.Std_Logic_unsigned.all;
begin
  if Rst='1' then
     Cnt <= (Cnt'range => '0')
  elsif rising_edge(Clk) and En = '1' then
     Cnt <= Cnt + 1;
   end if;
end process Reg_Proc;
```

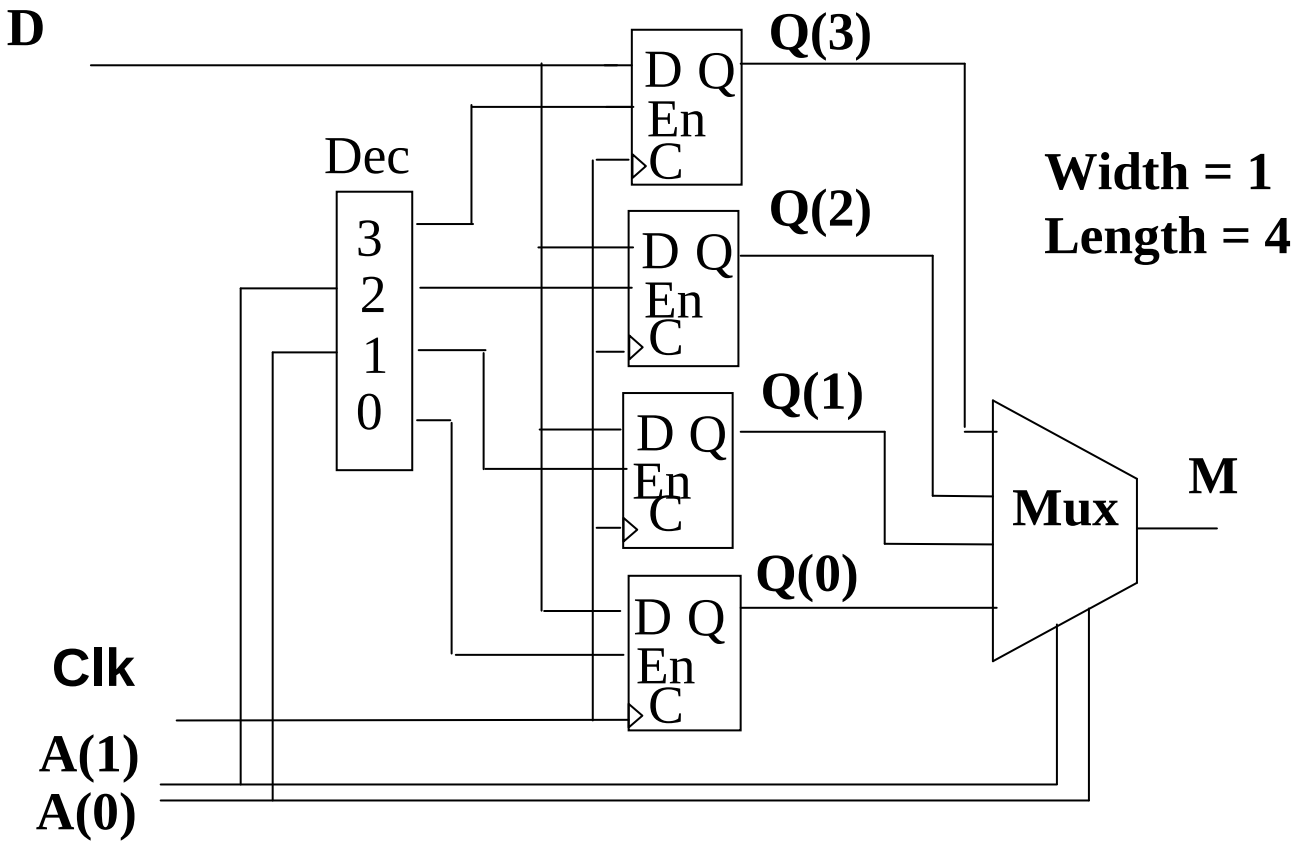
Additional condition!

Reglas procesos Sincronizados(3)

Cuando Usas la construcción if/then, solamente el reloj y el Reset o set debe aparecer en la lista de Sensibilidad.

```
Reg_Proc: process(Rst, Clk)
use IEEE.Std_Logic_unsigned.all;
begin
   if Rst = '1' then
     Cnt <= (Cnt'range => '0');
   elsif rising_edge(Clk) then
     if En = '1' then
       Cnt <= Cnt + 1;
     end if;
   end if;
end process Reg_Proc;
```

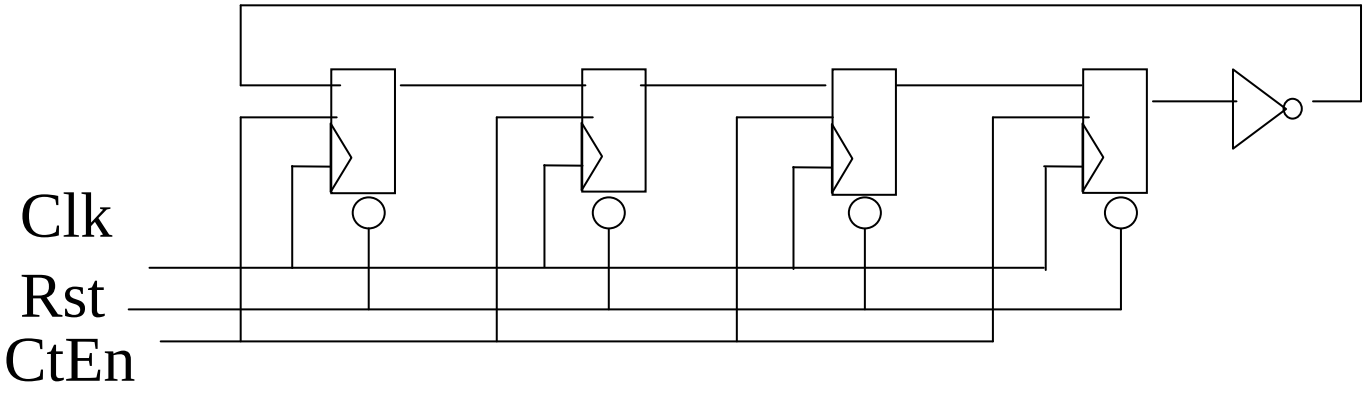
Memoria Read-Write

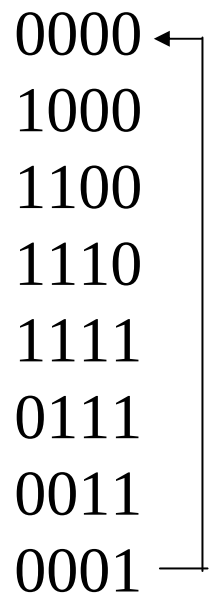



```
M <= Q(conv_integer(A));

process(Clk)
begin
   if rising_edge(Clk) then
     Q(conv_integer(A)) <= D;
   end if;
end process;
```

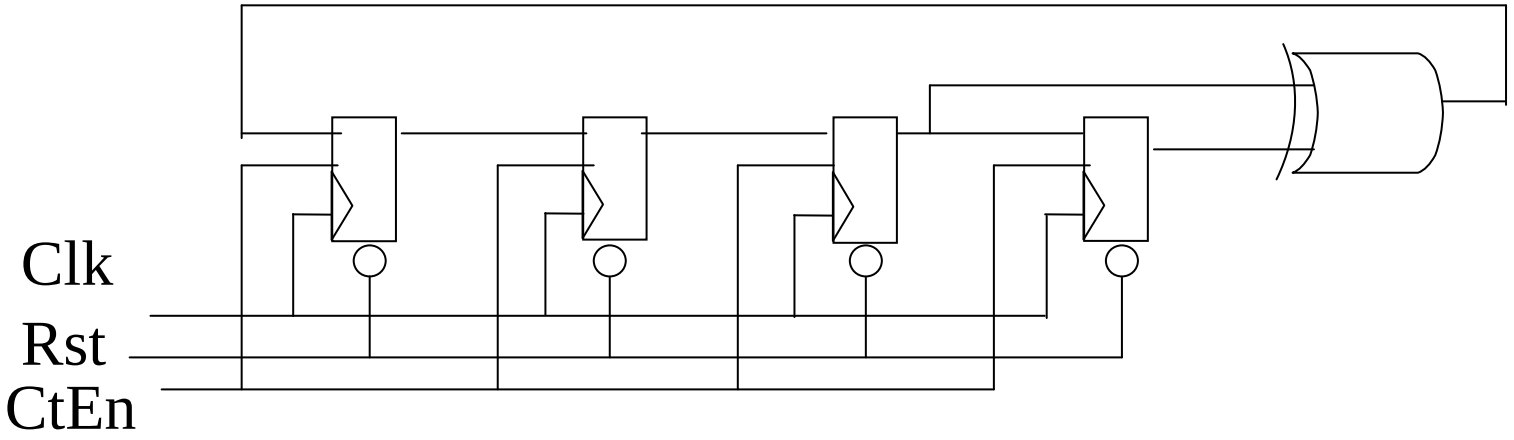
Contador Johnson

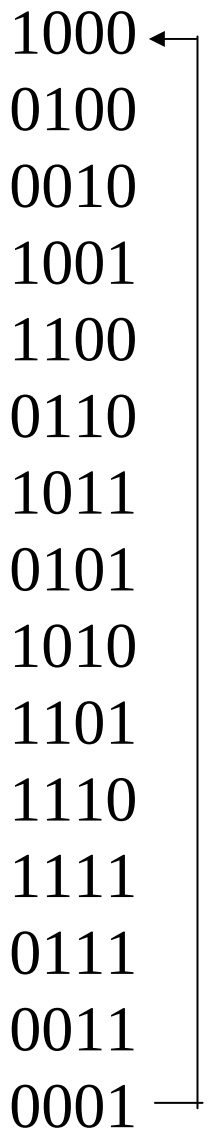





```
process(Rst, Clk)
begin
  if Rst='1' then
    Q <= (Q'range => '0');
  elsif rising_edge(Clk) then
    if CtEn = '1' then
       Q <= not Q(0) & Q(3 downto 1);
    end if;
  end if;
end process;
```

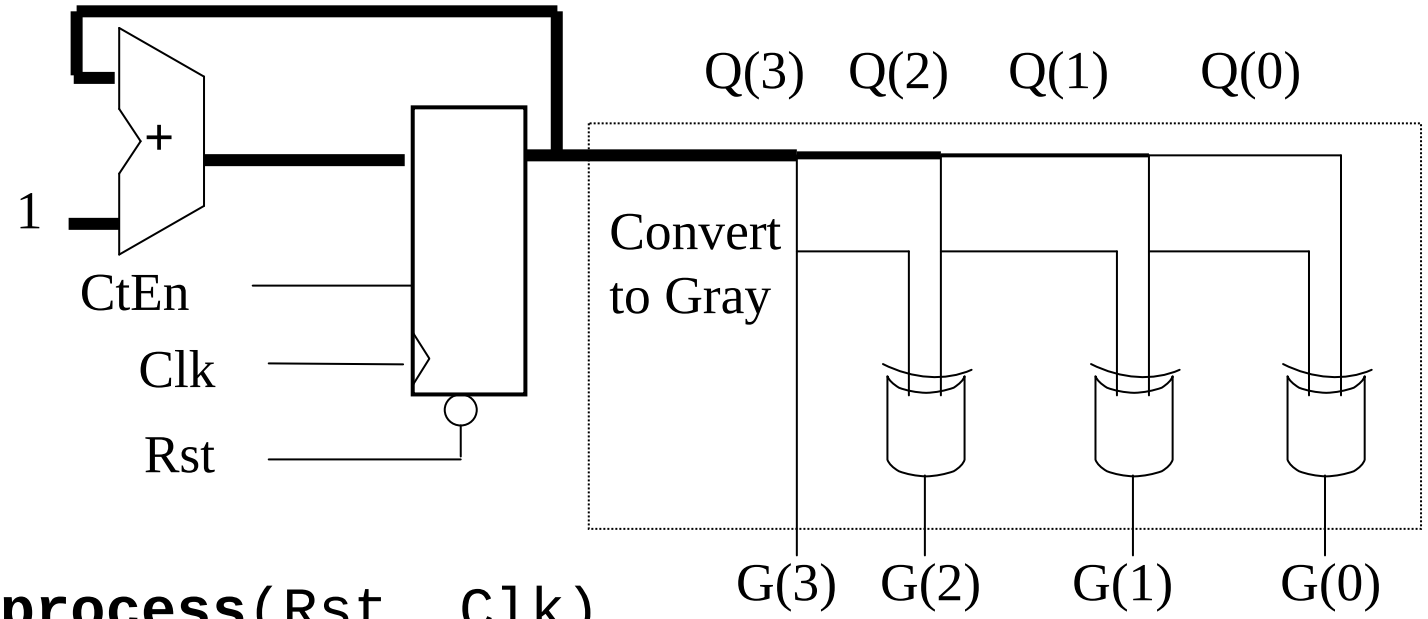
Linear Feedback Shift Register

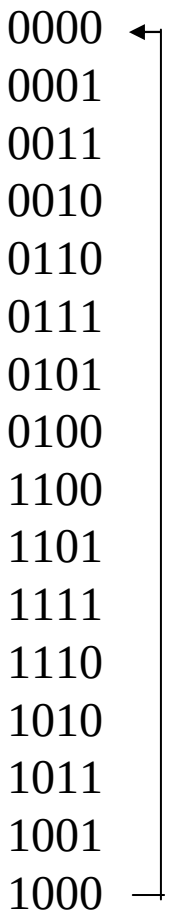





```
process(Rst, Clk)
begin
  if Rst='0' then
    Q <= (Q'range => '0');
  elsif rising_edge(Clk) then
    if CtEn = '1' then
      Q <= (Q(0) xor Q(1)) & Q(3 downto 1);
    end if;
  end if;
end process;
```

Contador Codigo Gray






```
process(Rst, Clk)
begin
  if Rst='1' then
    Q <= (Q'range => '0');
  elsif rising_edge(Clk) then
    if CtEn = '1' then
       Q <= Q + 1;
    end if;
  end if;
end process;
```

```
G(3) <= Q(3);
G(2) <= Q(3)xor Q(2);
G(3) <= Q(2)xor Q(1);
G(3) <= Q(1)xor Q(0);
```