INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Convertidor Binario a BCD

Contenido

Código BCD

El código BCD (Binary-Coded Decimal) es una representación numérica en la que cada dígito decimal se codifica en un número binario de 4 bits. En el código BCD, los números del 0 al 9 se representan directamente utilizando los cuatro bits correspondientes.

En el código BCD, cada dígito decimal se representa utilizando una combinación de cuatro bits. Por ejemplo:

- El número 0 se representa como 0000 en BCD.
- El número 1 se representa como 0001 en BCD.
- El número 2 se representa como 0010 en BCD.
- Y así sucesivamente hasta el número 9, que se representa como 1001 en BCD.

El código BCD es ampliamente utilizado en sistemas digitales, especialmente en aquellos que necesitan manipular y visualizar números decimales, hexadecimales, octales, etc. de forma directa.

Es común que se utilicen en aplicaciones donde se utilizan displays de 7 segmentos, donde cada segmento del display está controlado por un bit del código BCD para mostrar los dígitos decimales. También se utiliza en sistemas de contadores y en operaciones aritméticas con números decimales.

Para realizar la conversión de un número binario normal a un código BCD que indique sus unidades, decenas y centenas, se utiliza el método Shift Add-3, que será descrito a continuación.

Método Shift Add-3

El método Shift Add-3 es un algoritmo que sirve para convertir números binarios a código BCD (usado para que las maquinas puedan describir números).

En este caso vamos a convertir un número binario de máximo 8 bits, que podrá ir del 00000000 al 11111111, o del hexadecimal 00 al FF o del número decimal 0 al 255 a código BCD.

El método Shift Add-3 funciona siguiendo los siguientes pasos:

1. Recorre el número binario un bit a la izquierda.
2. Recorre a la izquierda el número binario hasta que el valor en cualquiera de las columnas de las unidades, decenas o centenas del código BCD sea mayor a 4 en decimal, osea a 100 en binario (fila **Shift 3** de la tabla).
3. Suma 3 decimal a ese valor en forma binaria, osea se le suma el 011 (fila **Add 3** de la tabla).
4. Se pone abajo el resultado de la suma y se bajan los demás ceros y unos que llevábamos también.
5. Se repiten los pasos anteriores hasta que en la columna de binario no quede ningún cero o uno.

Operación	Centenas	Decenas	Unidades	Binario	
Hexadecimal				F	F
Inicio				1111	1111
Shift 1			1	1111	111
Shift 2			11	1111	11
Shift 3			111	1111	1
Add 3			1010	1111	1
Shift 4		1	0101	1111	
Add 3		1	1000	1111	
Shift 5		11	0001	111	
Shift 6		110	0011	11	
Add 3		1001	0011	11	
Shift 7	1	0010	0111	1	
Add 3	1	0010	1010		
Shift 8	10	0101	0101		
BCD	2	5	5		

De la tabla podemos observar que el resultado deberá ser almacenado en un vector de 10 bits y para poder recorrer todas las posiciones de la tabla debemos crear un vector de 18 bits.

Resultado BCD =	A11 A10 A9 A8	A7 A6 A5 A4	A3 A2 A1 A0		
Shift 8	10	0101	0101		
BCD	2	5	5		
Posiciones =	S19 S18 S17 S16	S15 S14 S13 S12	S11 S10 S9 S8	S7 S6 S5 S4 S3 S2 S1 S0	

Los pasos descritos anteriormente son usados para resolver el método a mano, pero si queremos estandarizarlo para poder resolver el problema para un número binario de máximo 8 bits y pasarlo a código estos son los pasos que tenemos que seguir cuando se cree el código en los lenguajes VHDL y Verilog.

Primero que nada, en el código se deben crear dos vectores, uno de entrada y otro de salida o uno de entrada y 3 de salida, si es que se quiere separar el resultado de la conversión en unidades, decenas y centenas:

VHDL:

```
entity nombreEntidad is
   Port ( numBinario  : in STD_LOGIC_VECTOR (7 downto 0);
          --Vector de entrada de 8 bits para el número binario que quiero convertir
           salidaBCD : out  STD_LOGIC_VECTOR (11 downto 0)
          --Vector de salida de 12 bits para el código BCD de salida que será el equivalente del
          --número binario de entrada, este puede ser separado en centenas, decenas y unidades.
                              );
end nombreEntidad;
```

Además, se debe declarar un vector tipo variable que me permita recorrer todas las posiciones del método Shift Add-3 por medio de un bucle for para que pueda convertir un número binario de 8 bits:

```
variable posiciones: STD_LOGIC_VECTOR (17 downto 0);
--Variable tipo vector que puede almacenar todas las 18 posiciones posibles del método shift Add-3
```

Verilog:

```
module nombreModulo (
           input [7:0] numBinario,
          --Vector de entrada de 8 bits para el número binario que quiero convertir
           output [7:0] salidaBCD
          --Vector de salida de 12 bits para el código BCD de salida que será el equivalente del
          --número binario de entrada, este puede ser separado en centenas, decenas y unidades.
                              );
end nombreEntidad;
```

Además, se debe declarar un tipo de dato reg y otro integer que me permitan recorrer todas las posiciones del método Shift Add-3 por medio de un bucle for para que pueda convertir un número binario de 8 bits:

```
reg [17:0] posiciones;
integer ciclos1erBucleFor;
integer ciclos2doBucleFor;
--Variable tipo vector que puede almacenar todas las 18 posiciones posibles del método shift Add-3
```

Los pasos del algoritmo son iguales, no importando que lenguaje se elija:

1. Recorre el numero binario a la izquierda un bit, esto máximo se puede repetir 3 veces, durante las operaciones **SHIFT1**, **SHIFT2** y **SHIFT3** como se ve en la tabla donde se resolvió el método. Si recorremos 3 posiciones el número original, el vector posiciones de 8 bits estará situado de la posición **S3** a la **S10** porque en ese punto solo hay 3 bits en la columna de las Unidades y 5 bits en la columna Binario.
2. Después del **SHIFT3** se empezará a analizar cada columna para ver si el número binario contenido en cada una es mayor a 4. Después de haber ocurrido 8 corrimientos (osea después del SHIFT8),

el número a convertir ya estará abarcando las columnas de unidades, centenas y decenas, por lo que se debe analizar en estos corrimientos, osea **SHIFT4**, **SHIFT5**, **SHIFT6**, **SHIFT7** y **SHIFT8** si el número es mayor a 4 en cada columna para que si es así se le sumen 3 unidades.

a. Cada columna la puedo analizar individualmente si analizo las coordenadas del vector posiciones que las abarcan:
    i. Columna de Unidades: Posiciones **S11**, S10, S9 y S8.
    ii. Columna de Decenas: Posiciones **S15**, S14, S13 y S12.
    iii. Columna de Centenas: Posiciones **S19**, S18, S17 y S16.

Cada una de estas columnas debe ser analizada para ver si se le puede aplicar la operación **Add-3** con un condicional if, excepto por la columna de las Centenas, ya que nunca se llega a aplicar esta operación cuando queremos convertir un número binario de 8 bits.

3. Repetir el paso 1 hasta que no haya ningún bit en la columna de Binario, esto abarca las posiciones: **S19**, S18, S17, S16, **S15**, S14, S13, S12, **S11**, S10, S9 y S8.

Código de Conversión BCD en VHDL y Verilog

Código Verilog:

```
//CONVERTIDOR BINARIO a BCD: METODO SHIFT ADD-3
//Vamos a codificar este método en específico para un numero binario de máximo 8 bits, por lo que nuestro caso mas
//critico será el número 11111111, el método paso a paso se explica en el archivo 9.-Convertidor binario a BCD,
//de ahí vamos a sacar literalmente todos los pasos que se tienen que hacer y para poder pasarlo a código vamos a usar
//un bucle for.

//ALWAYS: Para poder usar bucles o condicionales debemos usar la instrucción always@(), dentro de su paréntesis van
//declaradas las entradas que se vaya a usar dentro del condicional o bucle y en específico para poder declarar el bucle
//definido for debo usar dos elementos llamados reg e integer.

//REG: Las variables no son ni entradas ni salidas de la NEXYS 2, solo se usan dentro del programa, en específico
//LOS REG SOLO SE PUEDEN USAR DENTRO DEL PROCESS y se declaran como una entrada o salida normal, ya sea en forma
//de 1 bit o en forma de vector.
//INTEGER: Son variables numéricas que sirven para indicar al bucle for cuantas veces se va a ejecutar.

//BUCLE FOR: Las líneas de código que existan dentro del bucle for se van a ejecutar varias veces cuando lleguemos a esa
//parte del código, para indicar cuantas veces se va a ejecutar usamos una variable integer.

module decodificadorBinBCD(
    input [7:0] numBinario,//Resultado del módulo anterior que le realizo una función matemática a un número binario.
    output [3:0] centenasBCD,
    output [3:0] decenasBCD,
    output [3:0] unidadesBCD

          //El código BCD describe cada digito decimal con 4 bits que indican sus unidades, decenas y centenas.
    );

        //REG: Existe solo en Verilog y sirve para almacenar datos que se puedan usar dentro de un condicional o bucle, solo
        //sobrevive durante la ejecución del programa, no está conectado a ningún puerto de la tarjeta de desarrollo y se le
        //asignan valores con el símbolo =
        reg [11:0] codigoBCDcompleto;
        reg [19:0] posiciones;

        integer ciclos;
        integer i;

        //Dentro de always@() se va a poner el algoritmo para ejecutar el método Shift Add-3, para entender el procedimiento
        //debo meterme al documento 9.-Convertidor Binario a BCD.
        always@(numBinario)
        begin
                //LLENAR DE CEROS TODAS LAS POSICIONES POSIBLES (OSEA LAS COLUMNAS) DE LA TABLA SHIFT ADD-3
                for(ciclos=0; ciclos<=19; ciclos=ciclos+1) begin
                        //Este bucle se va a repetir 20 veces y lo que hará es limpiar todos los bits de la variable
                        //posiciones.

                        posiciones[ciclos] = 1'b0;//Con limpiar nos referimos a llenar de ceros el vector.
                end
```

```
                //METER EL NUMERO BINARIO EN LA VARIABLE POSICIONES PARA QUE ENTRE A LA TABLA DEL MTODO SHIFT ADD-3
                posiciones[7:0] = numBinario[7:0];
                //Con esta instrucción metemos el valor de la entrada llamada binario a las posiciones 0,1,2,3,4,
                //5,6 y 7 del vector posiciones, esto se hace para que este en la posición inicial de la tabla.


                //METODO SHIFT ADD-3 A LAS COLUMNAS: UNIDADES, DECENAS Y CENTENAS
                for(i=0; i<=7; i=i+1) begin
                        //Este bucle se va a repetir 8 veces y lo que hará es ejecutar las operaciones SHIFT1, SHIFT2,
                        //SHIFT3, SHIFT4, SHIFT5, SHIFT6, SHIFT7 y SHIFT8, ya que estas son todas las veces que se puede
                        //recorrer el numero binario a la izquierda en la tabla del método SHIFT ADD-3, aplicado a un número
                        //binario de máximo 8 bits, durante estos recorrimientos se debe analizar cada columna de Unidades,
                        //Decenas y Centenas.

                        //COLUMNA DE UNIDADES: ADD-3
                        if(posiciones[11:8]>4'b0100) begin
                                //Cada columna la puedo analizar individualmente si analizo las coordenadas del vector
                                //posiciones que las abarcan, las posiciones 11, 10, 9 y 8 abarcan la columna de Unidades.
                                posiciones[11:8] = posiciones[11:8] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el número decimal 3, osea 011.
                        end

                        //COLUMNA DE DECENAS: ADD-3
                        if(posiciones[15:12]>4'b0100) begin
                                //Las posiciones 15, 14, 13 y 12 abarcan la columna de Decenas.
                                posiciones[15:12] = posiciones[15:12] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el numero decimal 3, osea 011.
                        end

                        //COLUMNA DE CENTENAS: ADD-3
                        if(posiciones[19:16]>4'b0100) begin
                                //Las posiciones 19, 18, 17 y 16 abarcan la columna de Centenas.
                                posiciones[19:16] = posiciones[19:16] + 4'b0011; //Add3
                                //Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                //osea 100 se le suma el numero decimal 3, osea 011.
                        end

                        //SHIFT: Este pedazo de código aplicara los SHIFT1,2,3,4,5,6,7 y SHIFT8.
                        posiciones[19:1] = posiciones[18:0];
                        //Esta operación del bucle es la operación SHIFT y es usada para mover un lugar a la izquierda todo
                        //el vector cuando ningún número en ninguna columna sea mayor a 4, osea 100.
                end

                //GUARDO EL RESULTADO DEL METODO SHIFT ADD-3 EN EL REG
                codigoBCDcompleto = posiciones[19:8];
                //Las posiciones 19 downto 8 son todas las que abarca el código BCD al terminar de ejecutarse el método
                //SHIFT ADD-3 porque ya no debe quedar ningún bit en la columna BINARIO.
        end


        //SEPARAR EL RESULTADO DEL MTODO SHIFT ADD-3, OSEA AL CODIGO BCD EN UNIDADES, DECENAS Y CENTENAS
        //A las salidas les asigno valores usando la palabra reservada assign y lo puedo hacer en cualquier lugar del código.
        assign centenasBCD = codigoBCDcompleto[11:8];//4 bits del código BCD representan un digito decimal, osea 1 centena
        assign decenasBCD  = codigoBCDcompleto[7:4];//4 bits del código BCD representan un digito decimal, osea 1 decena
        assign unidadesBCD = codigoBCDcompleto[3:0];//4 bits del código BCD representan un digito decimal, osea 1 unidad
endmodule
```

Código VHDL:

```
--CONVERTIDOR BINARIO a BCD: MTODO SHIFT ADD-3
--Vamos a codificar este método en específico para un numero binario de máximo 8 bits, por lo que nuestro caso mas
--critico será el número 11111111, el método paso a paso se explica en el archivo 9.-Convertidor binario a BCD
--de ahí vamos a sacar literalmente todos los pasos que se tienen que hacer y para poder pasarlo a código vamos a usar
--un bucle for.

--PROCESS: Para poder usar bucles o condicionales debemos usar la instrucción process(), dentro de su paréntesis van
--declaradas las entradas que vaya a usar dentro del condicional o bucle y en específico para poder declarar el bucle
--definido for debo usar en elemento llamado variable.

--VARIABLE: Las variables no son ni entradas ni salidas de la NEXYS 2, solo se usan dentro del programa, en especifico
--LAS VARIABLES SOLO SE PUEDEN USAR DENTRO DEL PROCESS y se declaran como una entrada o salida normal, ya sea en forma
--de 1 bit o en forma de vector.

--BUCLE FOR: Las líneas de código que existan dentro del bucle for se van a ejecutar varias veces.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--Estas libreras solo se declaran para poder usar el lenguaje VHDL.
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--Esta librería nos permite hacer operaciones matemáticas con vectores o bits sin considerar su signo.
use IEEE.STD_LOGIC_ARITH.ALL;
--Esta librería nos permite hacer operaciones matemáticas usando process.

entity DecodificadorBinBCD is
    Port ( numBinario : in  STD_LOGIC_VECTOR (7 downto 0);
           centenasBCD : out  STD_LOGIC_VECTOR (3 downto 0);
           decenasBCD : out  STD_LOGIC_VECTOR (3 downto 0);
```

```
          unidadesBCD : out  STD_LOGIC_VECTOR (3 downto 0));
end DecodificadorBinBCD;

architecture SHIFTADD3 of DecodificadorBinBCD is
--SIGNAL: No es ni una entrada ni una salida porque no puede estar vinculada a ningún puerto de la NEXYS 2, solo
--existe durante la ejecución del código y sirve para poder almacenar algún valor, se debe declarar dentro de la
--arquitectura y antes de su begin, se le asignan valores con el símbolo :=
signal codigoBCDcompleto : STD_LOGIC_VECTOR (11 downto 0);
--Esta signal sirve para almacenar el código BCD completo que incluye sus 4 bits de unidades, decenas y centenas.
begin
          --Dentro del process se va a poner el algoritmo para ejecutar el método Shift Add-3, para entender el procedimiento
          --debo meterme al documento 9.-Convertidor Binario a BCD.
          process(numBinario)
          --Las variables se deben declarar dentro de process y deben estar antes de su begin, estas solo van a poder existir
          --y ser usadas dentro del process donde estén declaradas, su sintaxis es: variable nombreVariable: tipo_de_dato;
          variable posiciones : STD_LOGIC_VECTOR (19 downto 0);
          begin
                    --LLENAR DE CEROS TODAS LAS POSICIONES POSIBLES (OSEA LAS COLUMNAS) DE LA TABLA SHIFT ADD-3
                    for ejecucionBucle in 0 to 19 loop
                              --Este bucle se va a repetir 20 veces y lo que hará es limpiar todos los bits de la
                              --variable posiciones.
                              posiciones(ejecucionBucle) := '0'; --Con limpiar nos referimos a llenar de ceros el vector.
                    end loop;

                    --METER EL NUMERO BINARIO EN LA VARIABLE POSICIONES PARA QUE ENTRE A LA TABLA DEL METODO SHIFT ADD-3
                    posiciones(7 downto 0) := numBinario(7 downto 0);
                    --Con esta instrucción metemos el valor de la entrada llamada numBinario a las posiciones 0,1,2,3,4,

                    --5,6 y 7 del vector posiciones, esto se hace para que este en la posición inicial de la tabla.

                    --MÉTODO SHIFT ADD-3 A LAS COLUMNAS: UNIDADES, DECENAS Y CENTENAS
                    for i in 0 to 7 loop --En la tabla se repite la operación hasta SHIFT8, por eso se ejecuta 8 veces.
                              --Este bucle se va a repetir 8 veces y lo que hará es ejecutar las operaciones SHIFT1, SHIFT2,
                              --SHIFT3, SHIFT4, SHIFT5, SHIFT6, SHIFT7 y SHIFT8 ya que estas son todas las veces que se puede
                              --recorrer el numero binario a la izquierda en la tabla del método SHIFT ADD-3 aplicado a un número
                              --binario de máximo 8 bits, durante estos recorrimientos se debe analizar cada columna de Unidades,
                              --Decenas y Centenas.

                              --COLUMNA DE UNIDADES: ADD-3
                              if posiciones(11 downto 8) > "100" then
                                        --Cada columna la puedo analizar individualmente si analizo las coordenadas del vector
                                        --posiciones que las abarcan, las posiciones 11, 10, 9 y 8 abarcan la columna de Unidades.
                                        posiciones(11 downto 8) := posiciones(11 downto 8)+"11"; --Add3
                                        --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                        --osea 100 se le suma el numero decimal 3, osea 011.
                              end if;

                              --COLUMNA DE DECENAS: ADD-3
                              if posiciones(15 downto 12) > "100" then
                                        --Las posiciones 15, 14, 13 y 12 abarcan la columna de Decenas.
                                        posiciones(15 downto 12) := posiciones(15 downto 12)+"11";--Add3
                                        --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                        --osea 100 se le suma el numero decimal 3, osea 011.
                              end if;

                              --COLUMNA DE CENTENAS: ADD-3
                              if posiciones(19 downto 16) > "100" then
                                        --Las posiciones 19, 18, 17 y 16 abarcan la columna de Centenas.
                                        posiciones(19 downto 16) := posiciones(19 downto 16)+"11";--Add3
                                        --Si el numero binario contenido en estas coordenadas del vector posiciones es mayor a 4,
                                        --osea 100 se le suma el numero decimal 3, osea 011.
                              end if;

                              --SHIFT: Este pedazo de código aplicara los SHIFT1,2,3,4,5,6,7 y SHIFT8
                              posiciones(19 downto 1) := posiciones(18 downto 0);
                              --Esta operación del bucle es la operación SHIFT y es usada para mover un lugar a la izquierda todo
                              --el vector cuando ningún número en ninguna columna sea mayor a 4, osea 100.
                    end loop;

          --GUARDO EL RESULTADO DEL METODO SHIFT ADD-3 EN LA SIGNAL.
          codigoBCDcompleto <= posiciones(19 downto 8);
          --Las posiciones 19 downto 8 son todas las que abarca el código BCD al terminar de ejecutarse el método
          --SHIFT ADD-3 porque ya no debe quedar ningún bit en la columna BINARIO.
          end process;

          --SEPARAR EL RESULTADO DEL MTODO SHIFT ADD-3, OSEA AL CODIGO BCD EN UNIDADES, DECENAS Y CENTENAS:
          --A las salidas les asigno valores usando el símbolo <= y lo puedo hacer en cualquier lugar del código.
          centenasBCD <= codigoBCDcompleto(11 downto 8);--4 bits del código BCD representan un dígito decimal, osea 1 centena.
          decenasBCD  <= codigoBCDcompleto(7 downto 4);--4 bits del código BCD representan un dígito decimal, osea 1 decena.
          unidadesBCD <= codigoBCDcompleto(3 downto 0);--4 bits del código BCD representan un dígito decimal, osea 1 unidad.
end SHIFTADD3;
```