MÉTODO DE CONVERSIÓN DE BINARIO A BCD CON VHDL

Juan Antonio Jaramillo Gómez, Dr., UPIITA-IPN

jantonioj@yahoo.com, jjaramillo@ipn.mx.

Mirna Salmerón Guzmán, M. en C., UPIITA-IPN

yupisg2@yahoo.com.mx.

Brahim El Filali, Dr., UPIITA-IPN

braelf@hotmail.com

Resumen

La necesidad de comunicar un dato que todo mundo entienda lleva a los sistemas digitales a presentar la información a través del uso de convertidores de datos binarios al formato de representación decimal. En este artículo se presentan algunas metodologías de conversión para display de 7 u 8 segmentos de una cantidad de dígitos específica. Cabe resaltar que una metodología comúnmente utilizada es la conocida como corre y suma 3 (shift-add 3), implementada en una tarjeta de desarrollo con un FPGA y lenguaje de VHDL en el programa ISE de Xilinx.

Abstract

The need to communicate information that everyone understands leads digital systems to present the information through the use of binary data converters to the decimal representation format. In this article we present some conversion methodologies for 7 or 8 segment display of a specific number of digits. It should be noted that a methodology commonly used is known as shift and sum 3 (shift-add 3), implemented in a development card with an FPGA and VHDL language in the Xilinx ISE program.

Introducción.

La conversión de números entre sus distintas bases y formatos es aplicable día con día no solo en los sistemas digitales para saber las equivalencias de un número binario a decimal o de decimal a binario o de hexadecimal a decimal, etc., sino que también permite saber el estado de un sistema, ya sea como medidores o indicadores en algún proceso o sistema controlado.

Todas las conversiones llevan una metodología para poderlas realizar, dependiendo del número a convertir, se busca la que sea más fácil y rápida. La conversión más básica es la de comparación, siendo aplicable cuando la cantidad de números a convertir no es tan grande. Esta metodología se muestra en la tabla 1, para realizar la conversión entre sí, de los primeros 16 números decimales a binarios y a hexadecimales.

Tabla 1. Conversión entre decimal, binario y hexadecimal.

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	111	F

Cuando el valor del número a convertir crece, no es tan fácil hacer una tabla de 1000 o más datos, por lo que se recurre a otras metodologías o artilugios lógico-matemáticos como de restas sucesivas (ver figura 1) o la de desplazar y sumar 3, que es la que se abarca aquí y se explica un poco más adelante.

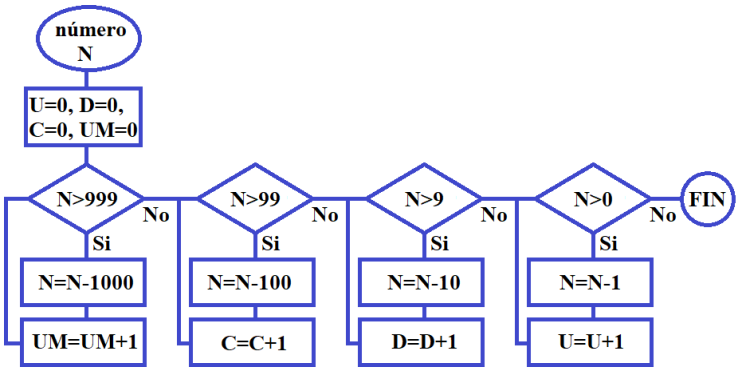

Figura 1. Diagrama de flujo de la metodología de restas.

El presente artículo mostrará cómo realizar la conversión de un número de 4 dígitos (9,999) dentro de una aplicación de medidor de milisegundos colocando un led testigo para ver su funcionamiento con una señal PWM. La medición de la variable tiempo es de gran potencial porque a partir de esta metodología se pueden realizar distintos medidores de velocidad,

distanca, aceleración, tiempo de caída libre, etc., además de que las partes del código es reutilizable para hacer indicadores de posición, ángulos, llenado, etc.

Desarrollo

Para mostrar la utilidad de la conversión de un número binario a BCD, primero se establece el uso de la metodología de desplazar y añadir 3 (shift and add 3) para convertir el número binario de 14 bits “10_0111_0000_1111” a decimal 9,999 para los 4 display (UM-C-D-U) de 7 segmentos de la tarjeta nexys 2 (ó 3 ó 4), en una aplicación de medidor de tiempo en base de milisegundos (9,999 ms). A partir de estos datos se implementa en la tabla 2 el llenado con la metodología mencionada, resaltando el número binario se desplaza hacia la izquierda hasta que cualquiera de las posiciones en unidades (U), decenas (D), centenas (C) o unidad de millar (UM) tengan un valor mayor o igual a 5, para lo cual se le sumará 3 y se continuará con el bucle de desplazamiento y suma.

En la tabla 2, el primer renglón es el título de las columnas de conversión (UM-C-D-U), los siguientes dos renglones son la representación hexadecimal (Hex) y binaria (bin) del número 9,999, a partir del cuarto renglón es la aplicación de la metodología de conversión (shift1, shift2,...add3,... etc.) acabado en shift14. Enseguida se pone el resultado de las columnas de conversión en decimal (9999) y el último renglón representa el tamaño del vector a utilizar para dicha conversión que es de 14 bits, así como los lugares del vector para asignar desde las unidades (bits 14 a 17) hasta la unidad de millar (bits 26 a 29).

Tabla 2. Conversión de binario a decimal para el número 9,999.

	UM			C				D				U																			
Hex																2		7				0				F					
Bin																1	0	0	1	1	1	0	0	0	0	1	1	1	1		
shift1															**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**			
shift2														**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**				
shift3													**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**					
shift4												**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**						
add3 (U)												1	1	0	0	1	1	0	0	0	0	1	1	1	1						
shift5											**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**							
add3 (U)											1	1	1	0	0	1	0	0	0	0	1	1	1	1							
shift6										**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**								
add3 (U)										1	1	1	1	0	0	0	0	0	0	1	1	1	1								
shift7									**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**									
add3 (D U)								1	0	1	0	1	0	1	1	0	0	0	1	1	1	1									
shift8							**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**										
add3 (D U)							1	1	0	0	0	1	0	0	1	0	0	1	1	1	1										
shift9						**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**											
shift10					**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**												
add3 (C)				1	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1												
shift11			**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**													
add3 (U)			1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	1													
shift12		**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**														
add3 (D U)		1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1														
shift13	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**															
add3 (C D U)	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1															
shift14	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**																
Decimal	**9**			**9**				**9**				**9**																			
UM_C_D_U	**29**		**26**	**25**		**22**		**21**		**18**		**17**		**14**		**13**	**12**	**11**		**8**		**7**		**4**		**3**		**0**			

En la tabla 3 se resaltan los dos datos importantes de la conversión para realizar el programa en el lenguaje VHDL, el primero es que se necesitan hacer 14 pasos para convertir este dato a BCD, y el segundo es el tamaño del vector de conversión de 30 bits. Con estos dos datos se realiza un código en VHDL para implementar la aplicación, disponible en la liga *código VHDL* y la liga del archivo de restricciones *código UCF*.

Tabla 3. Datos importantes de la conversión.

	UM			C			D			U																					
Hex													2			7						0					F				
Bin													1	0	0	1	1	1	0	0	0	0	1	1	1	1					
shift1												1	0	0	1	1	1	0	0	0	0	1	1	1	1						
shift2											1	0	0	1	1	1	0	0	0	0	1	1	1	1							
shift3										1	0	0	1	1	1	0	0	0	0	1	1	1	1								
shift4										1	0	0	1	1	1	0	0	0	0	1	1	1	1								
add3 (U)										1	1	0	0	1	1	0	0	0	0	1	1	1	1								
shift5									1	1	0	0	1	1	0	0	0	0	1	1	1	1									
add3 (U)									1	1	1	0	0	1	0	0	0	0	1	1	1	1									
shift6								1	1	1	0	0	1	0	0	0	0	1	1	1	1										
add3 (U)								1	1	1	1	0	0	0	0	0	0	1	1	1	1										
shift7							1	1	1	1	0	0	0	0	0	0	1	1	1	1											
add3 (D U)							1	0	1	0	1	0	1	1	0	0	0	1	1	1											
shift8						1	0	1	0	1	0	1	1	0	0	0	1	1	1	1											
add3 (D U)						1	1	0	0	0	1	0	0	1	0	0	1	1	1	1											
shift9					1	1	0	0	0	1	0	0	1	0	0	1	1	1	1												
shift10					1	1	0	0	0	1	0	0	1	0	0	1	1	1													
add3 (C)					1	0	0	1	0	0	1	0	0	1	0	0	1	1	1												
shift11				1	0	0	1	0	0	1	0	0	1	0	0	1	1	1													
add3 (U)				1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	1												
shift12			1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	1													
add3 (D U)			1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1													
shift13		1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1														
add3 (C D U)		1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1														
shift14	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1															
Decimal	9				9				9				9																		
UM_C_D_U	29			26	25			22	21			18	17			14	13	12	11			8	7			4	3			0	

14 pasos shift+add

9999 = UM_C_D_U[29:14]

30 bits

Al utilizar una FPGA que trabaja bajo arquitectura concurrente, el código está implementado en procesos, mostrados como bloques en la figura 2 y descritos a continuación.

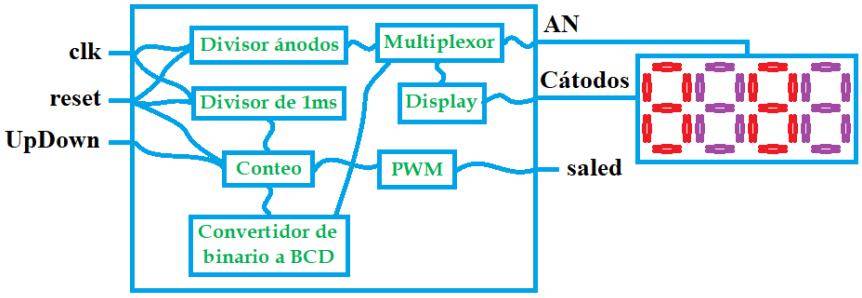

Figura 2. Diagrama a bloques de los procesos implementados.

- **Divisor de 1ms**: en este proceso se genera una señal "SAL_1ms" de 1ms de periodo.
- **Conteo**: contador cuya señal de reloj es SAL_1ms y con entradas Up para incrementar el conteo y Down decrementa el conteo en el rango 0<CONT<9999.
- **PWM** controlado por el contador de ms: se genera una señal PWM que se observa en la salida hacia un led.
- **Convertidor de binario a BCD**: este proceso contiene un algoritmo recorre y suma 3 para convertir un número binario a bcd, y que se manda a los displays. El algoritmo consiste en desplazar (shift) el vector inicial (en binario) el número de veces según sea

el número de bits, y cuando alguno de los bloques de 4 bits (U-D-C-UM, que es el número de bits necesarios para que cuente de 0 a 9 por cifra) sea igual o mayor a 5 (por eso el >4) se le debe sumar 3 a ese bloque, después se continua desplazando hasta que otro (o el mismo) bloque cumpla con esa condición y se le sumen 3. Inicialmente se rota 3 veces porque es el número mínimo de bits que debe tener para que sea igual o mayor a 5. Finalmente se asigna a otro vector, el vector ya convertido, que cuenta con 4 bloques para las 4 cifras de 4 bits cada una.

- **Divisor ánodos**: este proceso sirve para hacer un reloj que permita barrer o visualizar los datos del display con el control de los ánodos.
- **Multiplexor**: aquí se van barriendo los valores a mostrar en el display en base a la velocidad del divisor ánodos.
- **Display**: envía los datos decimales codificados para display de 7 segmentos.

El código ya escrito y sintetizado es implementado en la tarjeta de desarrollo Nexys 2 como se muestra en las fotos siguientes (ver figuras 3 y 4). También se muestra en la liga un ***video*** (*contador_ms.mp4*) del funcionamiento del sistema, resaltando por último que a partir de este medidor de milisegundos es posible realizar varias aplicaciones, como las que se muestran en seguida:

- Medidor de tiempo en alto de una señal
- Medidor de caída libre
- Medidor de distancia con el sensor SRF-05 (SRF-XX)
- Sistema de frenado automático
- Sistema de caracterización de servomotores
- Sistemas de control de velocidad, sentido de giro de servos
- Sistemas de control de posición de servos
- Sistemas de control de velocidad para motores de CD con y sin engranes
- Generadores de señales PWM

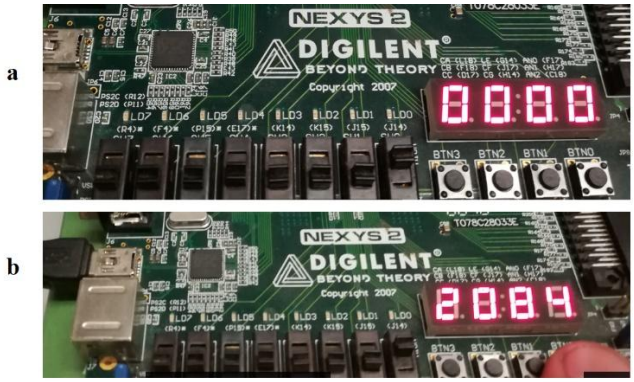

Figura 3. Foto del funcionamiento. (a) se muestra el reset, (b) inicio del conteo ascendente.

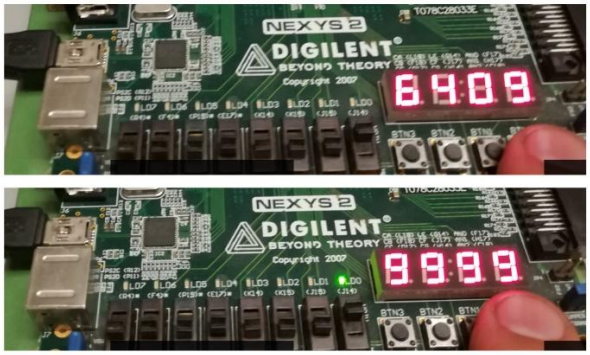

Figura 4. Foto del funcionamiento. (a) se muestra el conteo descendente, (b) máximo valor del contador y led de pwm encendido.

Conclusiones

Se ha presentado la implementación de una aplicación de medidor de milisegundos con un código en VHDL y en UCF, partiendo de una metodología de conversión de datos binarios a BCD conocida como desplaza y suma 3 (shift + add 3). Las fotos y los videos son la evidencia demostrativa de cómo es su funcionamiento, y también se han mostrado algunos alcances posibles en un pequeño listado de aplicaciones.

Referencias y Recursos electrónicos

1. Haskell, Richard E. y Hanna, Darrin M., Digital Design using FPGA Boards VHDL, LBE Books, pp. 93-108, 203-209.
2. Haskell, Richard E. y Hanna, Darrin M., Digital Design using FPGA Boards Verilog, LBE Books, pp. 109-119, 185-190.
3. Binary to BCD converter, recuperado en octubre 2017, disponible en: http://www.tkt.cs.tut.fi/kurssit/1426/S12/Ex/ex4/Binary2BCD.pdf
4. Binario a BCD recorre y suma+3, recuperado en octubre 2017, disponible en: http://www.circuitoselectronicos.org/2011/04/binario-bcd-recorre-y-suma3.html
5. Ramos Carlos, De binario a siete segmentos: la conversión, recuperado en octubre 2017, disponible en: http://www.estadofinito.com/binario-bcd-7seg/

Códigos para anexarse en ligas:

Primer código

```
----------------------------------------------------------------------------------
-- Contador de ms ascendente y descendente, con reset asíncrono
-- y pwm manual con período de 10s con salida a un led como indicador visual.

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

            -- Declaración de la entidad
entity Contador is
Port (UpDown: in STD_LOGIC_VECTOR(1 DOWNTO 0); -- botones para subir y bajar los
ms
      CLK: in STD_LOGIC; -- reloj de 50MHz para la nexys 2
      RESET: in STD_LOGIC; -- reset
      SALED: OUT STD_LOGIC; -- salida del led testigo
      DISPLAY: out STD_LOGIC_VECTOR(7 DOWNTO 0); -- segmentos del display
                                                      --"abcdefgP"
            AN: out STD_LOGIC_VECTOR(3 DOWNTO 0)); -- ánodos del display
end Contador;

            -- Declaración de la arquitectura
architecture Behavioral of Contador is

            -- Declaración de señales de los divisores
```

```
signal Conta_500us: integer range 1 to 250000:=1; -- uso en el pulso de 1ms
(pro. divisor 1ms)

signal contadors: integer range 1 to 6250:=1; --  pulso1 de 0.25ms (pro. divisor
ánodos)

signal SAL_1ms,SAL_250us: std_logic; --igual que pulso y pulso1, respectivamente


            -- Declaración de señales de los contadores

signal CONT: std_logic_vector (15 DOWNTO 0):=(others=>'0'); -- 16 bits (proc.
conteo)


            -- Declaración de señales de la asignación de U-D-C-UM

SIGNAL P: std_logic_vector (15 DOWNTO 0); -- asigna UNI, DEC,CEN, MIL

signal UNI,DEC,CEN,MIL: std_logic_vector (3 DOWNTO 0); -- digitos unidades,
decenas,

                                                    -- centenas y unidad
de millar


      -- Declaración de señales de la multiplexación y asignación de U-D-C-UM
al display

signal SEL: std_logic_vector (1 downto 0):="00"; -- selector de barrido

signal D: std_logic_vector (3 downto 0); -- sirve para almacenar los valores del
display


            -- Declaración de señales de la base de tiempo

signal PERIOD: std_logic_vector (15 downto 0):=x"0000"; -- base de tiempo ms
para el PWM

                                    -- (máximo “10 0111 0000 1111”=x”270F”)


BEGIN
```

```
----------------------DIVISOR 1ms-----------------------------------
-- en este proceso se genera una señal "SAL_1ms" de 1ms de periodo
PROCESS(reset, CLK)
begin
	if reset ='1' then
		Conta_500us <= 1; -- se reinicializa
	elsif(CLK'event and CLK='1') THEN
		if(Conta_500us = 25000) then -- pregunta si ya se alcanzo 0.5ms
			SAL_1ms <= not SAL_1ms; --se genera 0.5ms en bajo y 0.5ms en
alto
			Conta_500us <= 1; --reinicia el Contador a 1
		else
			Conta_500us <= Conta_500us + 1;
		end if;
	end if;
end process; --termina el proceso de generación de señal 1ms



--------------------------------CONTEO-------------------------------------
-- con Up se incrementa y Down decrementa en el rango 0<CONT<9999
PROCESS(RESET, SAL_1ms, UpDown, CONT)
begin
	if RESET='1' then
		CONT<=(others=>'0');
	else
		if(SAL_1ms'EVENT and SAL_1ms='1') then -- reloj SAL de 1ms
			if UpDown="01" then -- decrementa (Down)
				if(CONT=x"0000") then -- compara contra 0
```

```
                              CONT<=CONT; -- si llegó a cero mantiene el cero
                        else
                              CONT<=CONT-'1'; -- sino decrementa
                        end if;
                  elsIF UpDown="10" then --incrementa
                        if(CONT >= "0010011100001111") then -- compara contra
9,999
                                                      -- (270F hex)
                              CONT<=CONT; -- si llego a 9999 mantiene 9999
                        else
                              CONT<=CONT+'1'; -- sino incrementa
                        end if;
                  else --cubre UpDown="00" e UpDown="11"
                        CONT<=CONT;
                  end if;
            end if;
      end if;
end process;



--------------------PWM CONTROLADO POR CONTADOR DE ms-------------------------


PROCESS (RESET, SAL_1ms, PERIOD)
BEGIN
      if RESET='1' OR PERIOD >= "0010011100001111" then
            PERIOD <= (others=>'0');
      elsif (SAL_1ms'EVENT and SAL_1ms='1') then -- reloj SAL_1ms de 1ms
            PERIOD <= PERIOD + '1';
```

```
--            if (PERIOD <= CONT111) then SALED <='1'; -- Salida a led testigo
              if (PERIOD <= CONT) then SALED <='1'; -- Salida a led testigo
              else                                          SALED <='0';
              end if;
       end if;
end PROCESS;
```

Este es el otro Código de la otra liga

```
//                                          Archivo de restricciones de usuario
(UCF)                                          //

//DISPLAYS

NET "DISPLAY(7)" LOC = "L18"; // a

NET "DISPLAY(6)" LOC = "F18"; // b

NET "DISPLAY(5)" LOC = "D17"; // c

NET "DISPLAY(4)" LOC = "D16"; // d

NET "DISPLAY(3)" LOC = "G14"; // e

NET "DISPLAY(2)" LOC = "J17"; // f

NET "DISPLAY(1)" LOC = "H14"; // g

NET "DISPLAY(0)" LOC = "C17"; // P


//ANODOS

NET "AN(3)" LOC = "F17"; // AN0

NET "AN(2)" LOC = "H17"; // AN1

NET "AN(1)" LOC = "C18"; // AN2

NET "AN(0)" LOC = "F15"; // AN3


//UpDown,RESET--PUSH BUTTON
```

```
NET "UpDown(1)" LOC = "B18"; // btn0 SUBE

NET "UpDown(0)" LOC = "D18"; // btn1 BAJA

NET "RESET" LOC = "H13";          // btn2 RST


//CLK reloj 50MHz

NET "CLK" LOC = "B8";


//SALED salida a led testigo

NET "SALED" LOC = "J14"; // LD0
```