**Universidad Nacional Autónoma de México**

**Facultad de Ingeniería**

**Diseño de Sistemas Digitales**

Aguilar Enriquez Paul Sebastian – 415028130

Cabrera López Oscar Emilio - 312333261

2016/09/12

**Practica 1**

**Descripción del problema**

Diseñar y montar un circuito que muestre mediante un display el número de cuenta de uno de los integrantes del equipo.

El número de cuenta se mostrara de dos maneras diferentes. Considere el número de cuenta por valores posicionales, ejemplo: Si el número de cuenta es 123456789, la posición 0 tiene al digito 1, la posición 1 tiene al digito 2, la posición 2 tiene al digito 3 y así sucesivamente.

Primer forma: Se mostraran los dígitos del número de cuenta de izquierda a derecha, al llegar al último digito, se reiniciaría mostrando el digito de la siguiente posición inicial. Ejemplo, si se inició en la posición 0, al terminar tendrá que reiniciar en la posición 1, luego en la posición 2, y así sucesivamente hasta la última posición.

Segunda forma: Se mostraran los dígitos del número de cuenta de derecha a izquierda, al llegar al último digito (primer digito del número de cuenta leído de izquierda a derecha), se reiniciaría mostrando el digito de la siguiente posición inicial de derecha a izquierda. Ejemplo, si se inició en la posición 8, al terminar tendrá que reiniciar en la posición 7, luego en la posición 6, y así sucesivamente hasta la última posición (primer digito del número de cuenta de izquierda a derecha).

**Modelado de las variables internas**

*Convertidor Binario -> Número de Cuenta*

Número de cuenta a usar: 312333261

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Entradas** | | | | **Salidas** | | | | | | |
| **w** | **x** | **y** | **z** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

*Expresiones mínimas de cada salida del display*

-Para a1 = Ʃm (1,8) utilizando Quine-McCluskey

**a1** = I0 + I1 =

|  |  |  |
| --- | --- | --- |
| **Tabla de Grupos** | | |
| G0 | - |  |
| G1 | 0001 (1) | I1 |
|  | 1000 (8) | I0 |

|  |  |  |
| --- | --- | --- |
| **Tabla de Implicantes** | | |
|  | 1 | 8 |
| I0 |  | X |
| I1 | X |  |

-Para b1 = Ʃm (2,7) utilizando Quine-McCluskey

|  |  |  |
| --- | --- | --- |
| **Tabla de Implicantes** | | |
|  | 2 | 7 |
| I0 |  | X |
| I1 | X |  |

**b1** = I0 + I1 =

|  |  |  |
| --- | --- | --- |
| **Tabla de Grupos** | | |
| G0 | - |  |
| G1 | 0010 (2) | I1 |
| G2 | - |  |
| G3 | 0111 (7) | I0 |

-Para c1 = Ʃm (6)

Al ser un solo término, podemos deducir la salida directamente. **c1** =

-Para d1 = Ʃm (1,8)

Observamos que d1 = a1 por lo tanto **d1** =

-Para e0 = ƩM (2, 6,7) utilizando Quine-McCluskey

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tabla de Grupos** | | | | |
| G0 | - |  | 0-10 (2,6) | I1 |
| G1 | 0010 (2) | x | 011- (6,7) | I0 |
| G2 | 0110 (6) | x |  |  |
| G3 | 0111 (7) | X |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Tabla de Implicantes** | | |
|  | 2 | 6 | 7 |
| I0 | X | X |  |
| I1 |  | X | X |

**e0** = I0 + I1 =

-Para f0 = ƩM (7)

Al ser un solo término, podemos deducir la salida directamente. **F0** =

-Para g1 = Ʃm (1,8)

Observamos que g1 = d1 = a1 por lo tanto **g1** =

**Método de solución**

Para el desarrollo de esta práctica se decidió utilizar el FPGA Cyclone II.

El problema se abordó dividiéndolo en “módulos”.

Los módulos que se diseñaron e implementaron fueron:

* Temporizador de señal cuadrada de un segundo.
* Contador de 4 bits, con modo ascendente/descendente y semilla para valor inicial de cuenta.
* FlipFlop tipo T
* Decodificador de binario al número de cuenta a entregar.

Dentro del módulo principal se utilizaron los módulos mencionados anteriormente, la descripción general de la solución sería la siguiente:

Se implementó un temporizador de un segundo, el cual esta como entrada a un contador (inicialmente en modo ascendente) al que llamaremos “contador1”. El contador1 cada vez que llega a su tope superior emite una señal que le indica al otro contador (al que llamaremos “contador2”) que debe incrementar (también inicialmente en modo ascendente), el valor del contador2 es tomado como semilla para el reinicio del conteo en el contador1. Una vez que el contador2 llega a su tope superior, se produce un cambio de modo en ambos contadores, ahora están en modo descendente, el funcionamiento es el mismo, cada vez que el contador1 llega a su tope inferior, se indica un incremento en el contador2, y el valor del contador2 es tomado como valor semilla para reiniciar el conteo en el contador1. Cuando el contador2 llega a su tope inferior, se produce de nuevo un cambio de modo en ambos contadores. Para poder estar intercalando los modos de operación se utilizó la implementación del Flip Flop T.

Adicionalmente se diseñó e implemento una solución similar (ya que contiene ciertas adaptaciones) en el software de simulación “Proteus”. Esta solución es similar a la descrita anteriormente, salvo por los componentes usados (que fueron añadidos directamente y no implementados desde cero), la adición de un demultiplexor (este si implementado con compuertas) para poder intercalar entre la señal que le indica a los contadores si deben incrementar o decrementar su valor y algunas condiciones para determinar cuándo intercalar la señal de los contadores, estas condiciones se implementaron mediante compuertas lógicas.

**Código**

-- Copyright (c) 2016

-- Emilio Cabrera <emilio1625@gmail.com>

-- Sebastian Aguilar <penserbjorne>

--

-- GNU GENERAL PUBLIC LICENSE

-- Version 3, 29 June 2007

--

-- This program is free software: you can redistribute it and/or modify

-- it under the terms of the GNU General Public License as published by

-- the Free Software Foundation, either version 3 of the License, or

-- (at your option) any later version.

--

-- This program is distributed in the hope that it will be useful,

-- but WITHOUT ANY WARRANTY; without even the implied warranty of

-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

-- GNU General Public License for more details.

--

-- You should have received a copy of the GNU General Public License

-- along with this program. If not, see <http://www.gnu.org/licenses/>.

*Archivo Principal “proyecto.vhd”*

library ieee;

use ieee.std\_logic\_1164.all;

entity proyecto is

port (

clk: in std\_logic; -- entrada del reloj, pin 17

clr: in std\_logic := '1'; -- boton de reset, logica negada, pin 144

seg: out std\_logic\_vector(6 downto 0); -- salida a display de 7 segmentos, pines 63,64,66,67,69,70,71

ledRelojC1, ledRelojC2, ledUp: out std\_logic; -- indicadores de cuenta, ritmo y flipflop logica negada, pines 3, 7, 9

ledC1, ledC2: out std\_logic\_vector(3 downto 0) -- muestran la cuenta en binario, pines 51-53,55 y 57-60

);

end entity;

architecture archProyecto of proyecto is

-- señales para conectar del contador 1 al contador 2 y del contador 1 al decodificador respectivamente

signal busConta, busDeco: std\_logic\_vector(3 downto 0);

-- señales para conectar el timer como reloj del contador 1 y del flipflop, el cable para marcar el reloj del contador 2

signal cableReloj, cableLdClk: std\_logic;

-- la salida del flipflop (down), su valor negado (up) que indica a los contadores si ir en ascenso (1) o descenso (0)

signal down, up: std\_logic;

-- y la entrada al flipflop, una serie de compuertas cuya finalidad es detectar si se ha llegado al ultimo valor

signal inff: std\_logic;

begin

tock: entity work.tock

port map(

clk => clk, -- entrada: reloj de 50MHz

clr => not clr, -- entrada: master reset, logica negada

clk\_out => cableReloj -- salida: reloj de 1 Hz

);

ledRelojC1 <= not cableReloj; --salida: led de 1 Hz logica negada

C1: entity work.contAscDesBCD

port map(

clk => cableReloj, --

clr => not clr, -- el push button trabaja en lógica negada

Q => busDeco,

data => busConta,

load => cableLdClk,

up => up

);

ledC1 <= busDeco;

cableLdClk <= '1' when (busDeco = "1000" and up = '1') or (busDeco = "0000" and up = '0')

else '0'; --

ledRelojC2 <= not cableLdClk; --los led del fpg son logica negada

C2: entity work.contAscDesBCD

port map(

clk => cableLdClk,

clr => not clr, -- el push button trabaja en lógica negada

Q => busConta,

up => up

);

ledC2 <= busConta;

DEC: entity work.decoCuenta

port map(

bcd => busDeco,

led => seg

);

inff <= '1' when (busConta = "0000" and up = '0') or (busConta = "1000" and up = '1') else '0';

FF: entity work.flipflop

port map(

clk => cableReloj,

clr => not clr, -- el push button trabaja en lógica negada

T => inff,

Q => down

);

up <= not down;

ledUp <= not up; --los led del fpga son logica negada

end architecture archProyecto;

*Archivo del modulo del temporizador “tock.vhd”*

library ieee;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_1164.all;

entity tock is

port (

clk, clr: in std\_logic;

clk\_out: buffer std\_logic := '0'

);

end tock;

architecture tick of tock is

signal counter : integer range 0 to 24999999;

begin

process (clk, clr)

begin

if (clk'event and clk = '1') then

if (clr = '1') then

counter <= 0;

clk\_out <= '0';

end if;

counter <= counter + 1;

if (counter = 24999999) then

counter <= 0;

clk\_out <= NOT(clk\_out);

end if;

end if;

end process;

end tick;

*Archivo del modulo contador* “*contAscDesBCD.vhd”*

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity contAscDesBCD is

port(

clk: in std\_logic;

load, clr, up: in std\_logic := '0';

data: in std\_logic\_vector(3 downto 0) := "0000";

Q: buffer std\_logic\_vector(3 downto 0)

);

end entity contAscDesBCD;

architecture archContAscDesBCD of contAscDesBCD is

begin

process(clk, clr)

begin

if (clk'event and clk = '1') then

if clr = '1' then

Q <= "0000";

elsif load = '1' then

Q <= data;

elsif up = '1' then

if Q = "1001" then

Q <= "0000";

else

Q <= Q + 1;

end if;

else

if Q = "0000" then

Q <= "1001";

else

Q <= Q - 1;

end if;

end if;

end if;

end process;

end architecture archContAscDesBCD;

*Archivo del modulo flip flop tipo T “flipflop.vhd”*

library ieee;

use ieee.std\_logic\_1164.all;

entity flipflop is

port (

clk: in std\_logic;

T, clr: in std\_logic := '0';

clk\_enable: in std\_logic := '1';

Q: buffer std\_logic := '0'

);

end entity flipflop;

architecture archT of flipflop is

begin

process (clr, clk)

begin

if (clk'event and clk = '1') then

if clr = '1' then

Q <= '0';

elsif clk\_enable ='1' then

Q <= T xor Q;

end if;

end if;

end process;

end archT;

*Archivo del modulo decodificador “decoCuenta.vhd”*

library ieee;

use ieee.std\_logic\_1164.all;

entity decoCuenta is

port (

bcd: in std\_logic\_vector(3 downto 0);

led: out std\_logic\_vector(6 downto 0)

);

end entity decoCuenta;

architecture arqDecoCuenta of DecoCuenta is

begin

with bcd select

led <= "0000110" when "0000",

"1001111" when "0001",

"0010010" when "0010",

"0000110" when "0011",

"0000110" when "0100",

"0000110" when "0101",

"0010010" when "0110",

"0100000" when "0111",

"1001111" when "1000",

"1111111" when others;

end architecture arqDecoCuenta;

**Construcción virtual**

**Conclusiones**

* Aguilar Enriquez Paul Sebastian: El diseño e implementación para el problema planteado requirió de bastante investigación por nuestra parte. El problema en si fue sencillo de abordar, pero las complicaciones surgieron al tener que investigar sobre los componentes necesarios y su implementación, tanto en el lenguaje VHDL como en el software de simulación Proteus así como el ir haciendo adaptaciones a cada implementación dependiendo de las opciones y limitantes que las tecnologías usadas nos ponían. Considero que nuestra implementación en ambos caso cumple con su objetivo, pero que tal vez realizando un análisis y un diseño más profundo ambas soluciones pueden quedar más sencillas.
* Cabrera López Oscar Emilio: Con respecto al problema planteado, no tuvimos problema alguno en diseñar una solución, sin embargo al momento de implementar tanto en VHDL como en Proteus fue necesario documentarnos, sobre todo para saber cómo funcionaban los elementos que deseábamos utilizar y hacer algunas adaptaciones a nuestro diseño inicial ya que no es lo mismo trabajar con el lenguaje de programación en donde podemos abstraer nuestras ideas, a trabajar con componentes con características específicas que se deben respetar si o si.

**Repositorio en GitHub** <https://github.com/penserbjorne/clase-diseniosistemasdigitales-2017-1>