Diseño de Circuitos Integrados Digitales
IEE3753

Cronómetro

Norman F. Sáez
Sebastián Santelices Herrera
ssantelices@uc.cl
nfsaez@uc.cl

September 11, 2012

Contents

1 Descripción Modular

La descripción se realizará de manera que el módulo counter (que es el módulo top) sea el último en explicarse, ya que utilizará los módulos descritos previamente.

1.1 hex7seg

Este módulo corresponde a un decodificador de 7 segmentos, que será usado en el módulo top Counter para poder utilizar los displays de la tarjeta FPGA.

Tal como se puede apreciar en el diagrama, la lógica de las salidas es inversa. Por ejemplo, cuando la entrada es 0, en vez de prenderse todos los leds del decodificador y apagarse el segmento "g", este último se mantiene prendido y los demás aparecen en 0.

Codigo fuente ver en A.5

1.2 decoder_2to4

En este caso el decodificador recibe una señal de 2 bits y emite una señal de 4, en la que describe cuál de los displays debe encenderse en el decodificador de 7 segmentos antes mencionado, ya que debe haber uno para centésimas, otro para décimas y dos para los segundos. A continuación se muestra un diagrama donde se muestran los LOC de cada uno de los displays:

A continuación se presenta la simulación del funcionamiento de este módulo. La lógica establece que la salida seleccionada es aquella que está en 0.

Código fuente en A.4

1.3 secuence

El propósito de este módulo fue tener un reconocedor de secuencias que emitiera una señal cuando detectara la secuencia 1-0 en una entrada específica. Esto, aplicado a un botón, nos permitió guardar los tiempos en el instante específico en que el dedo suelta el botón por primera vez, ya que antes de eso nos enfrentábamos al hecho que cuando un dedo presiona, lo hace por muchos ciclos de clock a 50 MHz, con lo que no podría guardarse un dato si se activara sólo con un 1 (dedo presionado) en la entrada, ya que se guardarían los 3 tiempos seguidos, debido a lo rápido del clock del sistema. Tal como se puede apreciar en el código, este módulo no es más que una máquina de Mealy que reconoce la secuencia 1-0. Código fuente en A.6

Tal como se puede apreciar en la Ilustración 4, el módulo emite un 1 cuando la señal de entrada varía de 1-0, como se esperaba.

A Apéndice

A.1 counter.ucf

```
NET "AN[0]" LOC=F17;
NET "AN[1]" LOC=H17;
NET "AN[2]" LOC=C18;
NET "AN[3]" LOC=F15;
NET "CLK_50M" LOC=B8;
```

```
NET "CLK_50M" PERIOD =20.0ns HIGH 50%;
NET "LED0" LOC = J14;
NET "LED1" LOC = J15;
NET "LED2" LOC = K15;
NET "LED3" LOC = K14;
NET "LED4" LOC = E17;
NET "LED5" LOC = P15;
NET "LED6" LOC = F4;
NET "LED7" LOC = R4;
NET "RESET" LOC = H13;
//NET "PAUSE" LOC = B18;
NET "LAP" LOC = D18;
NET "LAP1" LOC = G18;
NET "LAP2" LOC = H18;
NET "LAP3" LOC = K18;
NET "PAUSE" LOC = R17;
NET "SEVEN_SEG[6]" LOC = L18;
NET "SEVEN_SEG[5]" LOC = F18;
NET "SEVEN_SEG[4]" LOC = D17;
NET "SEVEN_SEG[3]" LOC = D16;
NET "SEVEN_SEG[2]" LOC = G14;
NET "SEVEN_SEG[1]" LOC = J17;
NET "SEVEN_SEG[0]" LOC = H14;
NET "DP" LOC = C17;
```

A.2 counter.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    11:50:23 09/04/2012
// Design Name:
// Module Name:    counter
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module counter(
        input reset,
        input pause,
        input lap,
        input lap1,
        input lap2,
```

```
    input lap3,
    input clk_50M,
    output [6:0] seven_seg,
    output [3:0] an,
    output led0,
    output led1,
    output led2,
    output led3,
    output led4,
    output led5,
    output led6,
    output led7,
    output dp
);

reg clk_1Hz = 1'b0;
reg [24:0] counter_50M;
reg [3:0] mins;

reg [3:0] centesimas;
reg [3:0] decimas;
reg [3:0] segundos0;
reg [3:0] segundos1;
reg [3:0] minutos;
reg [1:0] pos;
reg [3:0] count;

reg [3:0] l1_cen;
reg [3:0] l1_dec;
reg [3:0] l1_seg0;
reg [3:0] l1_seg1;
reg [3:0] l1_min;

reg [3:0] l2_cen;
reg [3:0] l2_dec;
reg [3:0] l2_seg0;
reg [3:0] l2_seg1;
reg [3:0] l2_min;

reg [3:0] l3_cen;
reg [3:0] l3_dec;
reg [3:0] l3_seg0;
reg [3:0] l3_seg1;
reg [3:0] l3_min;

reg [1:0] lap_count;

wire lap_debounce;
wire lap_clean;

parameter COUNT_05M  = 250000; // para clock de 0.01s (centesimas)

decoder_2to4 D24(pos,an);
hex7seg H1(count, seven_seg);
```

```
debounce DB(lap,clk_1Hz,reset,lap_debounce);
secuence SC(lap_debounce, clk_1Hz, reset, lap_clean);


always @ (posedge clk_50M or posedge reset)
begin

    if (reset)
        begin
            counter_50M <=0;
            clk_1Hz <= 1'b0;
            pos <= 2'b00;
        end
    else
        if (counter_50M == COUNT_05M)
            begin
                counter_50M <= 0;
                clk_1Hz <= ~clk_1Hz;
            end
        else
            begin
                counter_50M <= counter_50M + 1;
                pos <= counter_50M[12:11];

            end
end

always @ (posedge clk_1Hz or posedge reset)
begin
    if (reset)
        begin
            centesimas <= 4'b0000;
            decimas    <= 4'b0000;
            segundos0  <= 4'b0000;
            segundos1  <= 4'b0000;
            minutos    <= 4'b0000;

            lap_count  <= 2'b00;

            l1_cen     <= 4'b0000;
            l1_dec     <= 4'b0000;
            l1_seg0    <= 4'b0000;
            l1_seg1    <= 4'b0000;
            l1_min     <= 4'b0000;

            l2_cen  <= 4'b0000;
            l2_dec  <= 4'b0000;
            l2_seg0 <= 4'b0000;
            l2_seg1 <= 4'b0000;
            l2_min  <= 4'b0000;

            l3_cen  <= 4'b0000;
            l3_dec  <= 4'b0000;
            l3_seg0 <= 4'b0000;
```

```
        l3_seg1 <= 4'b0000;
        l3_min  <= 4'b0000;
    end
else
    begin
    if (lap_clean)
    begin
    if (lap_count == 2'b00)
        begin
            l1_cen  <= centesimas;
            l1_dec  <= decimas;
            l1_seg0 <= segundos0;
            l1_seg1 <= segundos1;
            l1_min  <= minutos;
            lap_count <= lap_count + 1;
        end

    if (lap_count == 2'b01)
        begin
            l2_cen  <= centesimas;
            l2_dec  <= decimas;
            l2_seg0 <= segundos0;
            l2_seg1 <= segundos1;
            l2_min  <= minutos;
            lap_count <= lap_count + 1;
        end

    if (lap_count == 2'b10)
        begin
            l3_cen  <= centesimas;
            l3_dec  <= decimas;
            l3_seg0 <= segundos0;
            l3_seg1 <= segundos1;
            l3_min  <= minutos;
            lap_count <= lap_count + 1;
        end

    end

        if (centesimas == 4'b1001 && ~pause) //09 centesimas!
            begin
                centesimas <= 4'b0000;
                decimas <= decimas +1;
            end
        else
            if (~pause)
            begin
            centesimas <= centesimas + 1;
            end
       if (decimas == 4'b1001 && centesimas == 4'b1001 && ~pause)
            begin
                decimas <= 4'b0000;
                segundos0 <= segundos0 + 1;
            end
```

```
                if (segundos0 == 4'b1001 && decimas == 4'b1001 && centesimas == 4'b1001 && ~pause)
                    begin
                        segundos0 <= 4'b0000;
                        segundos1 <= segundos1 + 1;
                    end

                if (segundos1 == 4'b0101 && segundos0 == 4'b1001 && decimas == 4'b1001 && centesimas
                    begin
                        segundos1 <= 4'b0000;
                        minutos <= minutos + 1;
                    end
                if (segundos1 == 4'b0101 && segundos0 == 4'b1001 && minutos == 4'b1000 && decimas ==
                    begin
                        minutos <= 4'b0000;
                    end
            end
    end



always @(*)
begin
    if (lap1)
    begin
        case (pos)
            0: count = l1_cen;
            1: count = l1_dec;
            2: count = l1_seg0;
            3: count = l1_seg1;
        endcase
        mins = l1_min;
    end
    else
        if (lap2)
        begin
            case (pos)
            0: count = l2_cen;
            1: count = l2_dec;
            2: count = l2_seg0;
            3: count = l2_seg1;
        endcase
        mins = l2_min;
        end
        else
        if (lap3)
        begin
            case (pos)
                0: count = l3_cen;
                1: count = l3_dec;
                2: count = l3_seg0;
                3: count = l3_seg1;
            endcase
        mins = l3_min;
```

```
        end
        else
        begin
            case (pos)
            0: count = centesimas;
            1: count = decimas;
            2: count = segundos0;
            3: count = segundos1;
        endcase
        mins = minutos;
    end
end

assign dp = (pos == 2) ? 1'b0:1'b1;
assign led0 = (mins >= 1) ? 1'b1:1'b0;
assign led1 = (mins >= 2) ? 1'b1:1'b0;
assign led2 = (mins >= 3) ? 1'b1:1'b0;
assign led3 = (mins >= 4) ? 1'b1:1'b0;
assign led4 = (mins >= 5) ? 1'b1:1'b0;
assign led5 = (mins >= 6) ? 1'b1:1'b0;
assign led6 = (mins >= 7) ? 1'b1:1'b0;
assign led7 = (mins >= 8) ? 1'b1:1'b0;

endmodule
```

A.3 debounce.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    17:28:21 09/10/2012
// Design Name:
// Module Name:    debounce
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module debounce(
    input [3:0] inp,
    input clock,
    input reset,
    output [3:0] outp
    );
```

```
reg [3:0] delay1;
reg [3:0] delay2;
reg [3:0] delay3;

always @(posedge clock or posedge reset)
begin
    if (reset ==1)
        begin
            delay1 <= 4'b0000;
            delay2 <= 4'b0000;
            delay3 <= 4'b0000;
        end
    else
        begin
            delay1 <= inp;
            delay2 <= delay1;
            delay3 <= delay2;
        end
end

assign outp = delay1 & delay2 & delay3;
endmodule
```

A.4 decoder.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    22:12:48 09/04/2012
// Design Name:
// Module Name:    decoder_2to4
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module decoder_2to4(
    input [1:0] in,
    output reg [3:0] out
    );
always @(*)
case (in)
0: out = 4'b1110;
1: out = 4'b1101;
```

```
2: out = 4'b1011;
3: out = 4'b0111;
default: out = 4'b1111;
endcase

endmodule
```

A.5 hex7seg.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    03:45:25 09/04/2012
// Design Name:
// Module Name:    hex7seg
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module hex7seg(
    input [3:0] x,
    output reg [6:0] a_to_g
    );

always @(*)
    case (x)
        0: a_to_g = 7'b0000001;
        1: a_to_g = 7'b1001111;
        2: a_to_g = 7'b0010010;
        3: a_to_g = 7'b0000110;
        4: a_to_g = 7'b1001100;
        5: a_to_g = 7'b0100100;
        6: a_to_g = 7'b0100000;
        7: a_to_g = 7'b0001111;
        8: a_to_g = 7'b0000000;
        9: a_to_g = 7'b0000100;
        'hA: a_to_g = 7'b0001000;
        'hB: a_to_g = 7'b1100000;
        'hC: a_to_g = 7'b0110001;
        'hd: a_to_g = 7'b1000010;
        'hE: a_to_g = 7'b0110000;
        'hF: a_to_g = 7'b0111000;
        default: a_to_g = 7'b0000001;
    endcase
```

```
endmodule
```

A.6 secuence.v

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:37:58 09/10/2012
// Design Name:
// Module Name:    pausa
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module secuence(
    input entrada,
    input clk,
    input reset,
    output salida
    );

reg state, nextstate;

parameter s0 = 1’b0;
parameter s1 = 1’b1;
reg y;


always @(posedge reset, posedge clk )
    begin
        if (reset)
            state <= s0;
        else
            state <= nextstate;
    end


always @(*)
    begin
        case(state)
            s0: if(entrada)
                    nextstate = s1;
                else
```

```
                    nextstate = s0;
            s1: if(entrada)
                    nextstate = s1;
                else
                    nextstate = s0;
        endcase
    end

assign salida = state & ~entrada ;
endmodule
```