Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica

IEE2713 — Sistemas Digitales – 2019-2

Pauta Tarea 6

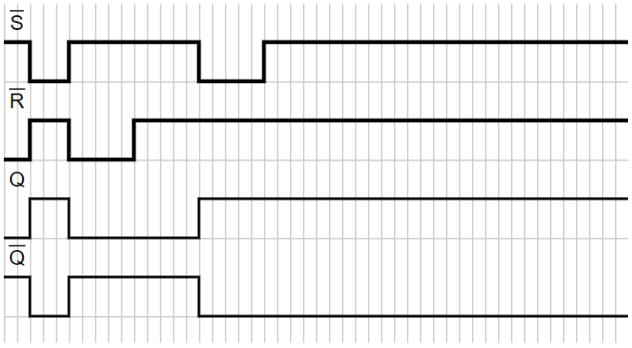
1. A diferencia de un latch SR con compuertas NOR, es que los inputs Set (S) y Reset (R) realizan su función de manera inversa (active low). Es decir, S lleva la salida a 1 cuando éste es 0, a diferencia de un latch SR con compuertas NOR que lo haría cuando S vale 1. Lo mismo para el input R: lleva la salida a 0 cuando éste es 0 y no 1. Por lo tanto, generalmente cuando el latch está construido con compuertas NAND, las entradas Set y Reset se definen como $\bar{S}$ y $\bar{R}$ para indicar que cumplen su función de manera inversa.

   La tabla de verdad es la siguiente:

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
0	0	1(X)	1(X)
0	1	1	0
1	0	0	1
1	1	0/1	1/0

   El latch no funciona cuando ambas entradas son 0, por lo que pueden ser consideradas don't care. Cuando ambas son 1, entonces se conserva el estado anterior.

   Un diagrama de tiempo simple se muestra a continuación:



2. Para analizar la secuencia necesitamos primero definir el número en binario de salida como $Q_2Q_1Q_0$ y no en orden inverso. Además, asumimos que todos parten en 0.

Luego, notamos que al ser la entrada siempre 1, entonces Q_0 va a alternar entre 0 y 1 en cada flanco positivo de clock.

Para Q_1, se produce el mismo fenómeno sólo que cambia cada dos flancos de clock. En un ciclo, $Q_0 = 0$ y por ende Q_1 no cambia en el siguiente flanco. Una vez pasado el ciclo anterior, ahora $Q_0 = 1$ y hará que en el siguiente flanco Q_1 cambie. Es decir, tiene un período el doble de largo que Q_0.

Con Q_2 ocurre algo diferente. Durante dos ciclos de clock $Q_1 = 0$ por lo que Q_2 no cambia. Cuando $Q_1 = 1$ genera que en el cuarto ciclo Q_2 cambie. Sin embargo, como Q_1 se mantiene en 1 un ciclo más de clock, genera que Q_2 vuelva a 0 inmediatamente y sólo genere un 1 cada cuatro ciclos de clock.

Finalmente la cuenta es la siguiente:

Clk	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	1	1	1
4	0	0	0
5	0	0	1
6	0	1	0
7	1	1	1
8/0	0	0	0

3. Hay que enfocarse en dos puntos. El primero es el retardo ocasionado por el camino más largo (desde la salida de un flip-flop hasta la entrada de otro). El segundo es el tiempo mínimo que la señal de entrada a un flip-flop tiene que estar estable antes del cambio en el clock y cuando tiene que permanecer constante después de este cambio.

   Para el primero, el camino más largo corresponde desde el primer flip-flop hasta la entrada del cuarto. El retardo es de 5ns (3 AND, 1 XOR, 1 MUX).

   En segundo lugar, sumamos t_{su} con t_h para tener el mínimo tiempo total necesario en que no se puede cambiar la entrada a un flip-flop.

   De esta forma, llegamos al retardo total mínimo $T_{min} = 5 + 3 + 1 = 9$ns. Finalmente:

$$f_{max} = 1/T_{min} = 111\text{MHz}$$

Con una frecuencia más rápida un cambio en una señal puede no alcanzar a llegar a tiempo para ser reconocido por un flip-flop antes del siguiente flanco de clock.

4. Antes de iniciar con los módulos específicos, necesitamos crear un módulo base de un flip-flop. Noten que este es un flip-flop tanto con un reset como con un enable.

Listing 1: FlipFlopRE.v

```
`timescale 1ns / 1ps
//////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/20/2019 10:18:37 AM
// Design Name:
// Module Name: FlipFlopRE
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////

module FlipFlopRE(
    clk,        // Reloj de entrada
    rst,        // Reset asincrono
    en,         // Habilita flip flop
    d,          // Entrada
    q           // Salida
    );

        parameter WIDTH = 1;
        input clk;
        input rst;
        input en;
        input [WIDTH-1:0] d;
        output reg [WIDTH-1:0] q;

        always @(posedge clk or posedge rst)
                if (rst)
                        q <= {WIDTH{1'b0}};
```

```
40                  else if (en)
41                          q <= d;
42                  else
43                          q <= q;
44  endmodule
```

Luego, creamos un módulo contador estándar, que describimos de forma parametrizada para contar hasta distintos máximos según necesitemos.

Listing 2: Contador.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date: 11/20/2019 10:16:19 AM
7   // Design Name:
8   // Module Name: Contador
9   // Project Name:
10  // Target Devices:
11  // Tool Versions:
12  // Description:
13  //
14  // Dependencies: Requiere un flip-flop FlipFlopRE
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////
21
22
23  module Contador(
24      clk,     // reloj de entrada
25      rst,     // reset
26      en,      // habilita contador
27      count    // cuenta de salida
28      );
29
30          parameter n=2;
31
32          input clk, rst, en;
33          output [n-1:0] count;
34          wire [n-1:0] next_count;
35
```

```
36          FlipFlopRE #(n) count_reg(.clk(clk), .rst(rst), .en(en)
               , .d(next_count), .q(count));
37          assign next_count = rst ? {n{1'b0}} : count + 1;
38  endmodule
```

Siguiendo con los módulos base, a continuación describimos un módulo decodificador de binario a un display de 7 segmentos.

Listing 3: sevenseg_mod.v

```
1   //************************************************
2   //File: sevenseg_mod.v
3
4   /**
5    * sevenseg_mod.v
6    * --------------
7    * Decodificador de binario a 7 segmentos BASYS3
8    * Cada segmento se activa en low.
9    * Maquina combinacional
10   *
11   *
12   *
13   * Autor: Angel Abusleme
14   */
15
16  module sevenseg_mod(aN,EnN,Led);
17      input [3:0] aN; // Numero a desplegar en 7 segmentos
18      input EnN;           // Habilita el display
19      output [6:0] Led;  // Salidas a display de 7 segmentos
20      reg [6:0] Led;
21
22          always @(EnN or aN) // Cada vez que el binario de
               entrada o el enable cambian
23          begin
24                  case ({EnN,aN})
25                     5'b10000:    Led <= 7'b1000000;   // Cuando
                          la entrada es 0 solo G esta desactivado
26                     5'b10001:    Led <= 7'b1111001;
27                     5'b10010:    Led <= 7'b0100100;
28                     5'b10011:    Led <= 7'b0110000;
29                     5'b10100:    Led <= 7'b0011001;
30                     5'b10101:    Led <= 7'b0010010;
31                     5'b10110:    Led <= 7'b0000010;
32                     5'b10111:    Led <= 7'b1011000;
33                     5'b11000:    Led <= 7'b0000000;
34                     5'b11001:    Led <= 7'b0010000;
```

```
35                      5'b11010:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
36                      5'b11011:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
37                      5'b11100:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
38                      5'b11101:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
39                      5'b11110:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
40                      5'b11111:    Led <= 7'b0111111;   // Entrada
                            invalida - sirve para debug
41                      default:     Led <= 7'b1111111;    // Todo
                           desactivado
42                  endcase
43          end
44  endmodule
```

De esta forma, creamos un módulo que utiliza los módulos anteriores y despliega en el display respectivo el número correspondiente según los inputs que recibe.

Listing 4: Four_Seven_Segment.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////
3  // Company: IEE2713
4  // Engineer: Angel Abusleme
5  //
6  // Create Date: 11/20/2019 07:22:37 PM
7  // Design Name:
8  // Module Name: Four_Seven_Segment
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description: maneja 4 displays de 7 segmentos
13 // utiliza un clock global y un enable de 1ms
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////
21
22
23 module Four_Seven_Segment(
```

```
24      input [3:0] D0,
25      input [3:0] D1,
26      input [3:0] D2,
27      input [3:0] D3,
28      input clk,
29      input En,                   // Habilita la cuenta, hace reloj
            mas lento
30      output reg [3:0] Anodo,
31      output reg [6:0] LedsAG
32      );
33
34      wire [6:0] Leds0;           // Salida del display cero
35      wire [6:0] Leds1;           // Salida del display uno
36      wire [6:0] Leds2;           // Salida del display dos
37      wire [6:0] Leds3;           // Salida del display tres
38      wire [3:0] Cuenta7seg;       // Cuenta para pasar de un
            display al otro
39
40      Contador #(4) Contador_7seg(.clk(clk), .rst(1'b0), .en(En),
            .count(Cuenta7seg));
41
42      // Modulos combinacionales para decodificar a 7 segmentos
43      sevenseg_mod SevenSeg0(.aN(D0),.EnN(1'b1),.Led(Leds0));
44      sevenseg_mod SevenSeg1(.aN(D1),.EnN(1'b1),.Led(Leds1));
45      sevenseg_mod SevenSeg2(.aN(D2),.EnN(1'b1),.Led(Leds2));
46      sevenseg_mod SevenSeg3(.aN(D3),.EnN(1'b1),.Led(Leds3));
47
48      always @(Cuenta7seg[3:2] or Leds0 or Leds1 or Leds2 or
            Leds3)
49      begin
50          case (Cuenta7seg[3:2])
51              2'b00:  {Anodo, LedsAG} <= {4'b1110, Leds0};
52              2'b01:  {Anodo, LedsAG} <= {4'b1101, Leds1};
53              2'b10:  {Anodo, LedsAG} <= {4'b1011, Leds2};
54              2'b11:  {Anodo, LedsAG} <= {4'b0111, Leds3};
55              default:{Anodo, LedsAG} <= {4'b1111, 7'b1111111};
56          endcase
57      end
58  endmodule
```

5. En esta parte implementamos un módulo que genera un pulso one-shot dependiendo del clock que tenga como input. En esta parte también será corregido el módulo Top final del siguiente punto.

Listing 5: OneShot.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////
3  // Company: IEE2713
4  // Engineer: Angel Abusleme
5  //
6  // Create Date: 11/20/2019 10:47:31 AM
7  // Design Name:
8  // Module Name: OneShot
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description: Este modulo genera un pulso de longitud igual
       al periodo del reloj
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////
21
22
23 module OneShot(
24     input in,        // Entrada
25     input clk,       // Clock
26         input rst,
27     output outp      // Salida
28     );
29
30 reg in_dly;          // Entrada con retardo
31
32 assign outp = in & ~in_dly;      // La salida del modulo es el
       and entre entrada y entrada con retardo negada
33                                  // Con esto la salida es 1 solo
                                        si la entrada es 1 y la
                                       entrada anterior fue 0
34
35 always @(posedge clk or posedge rst)
36     if(rst)
37         in_dly <= 1'b0;
38     else
39         in_dly <= in;
40 endmodule
```

6. El debouncer se encuentra en el siguiente código:

Listing 6: Debounce.v

```
1  `timescale 1ns / 1ps
2  //////////////////////////////
3  // Company: IEE2713
4  // Engineer: Angel Abusleme
5  //
6  // Create Date: 11/20/2019 11:22:24 PM
7  // Design Name:
8  // Module Name: Debounce
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description: Este modulo limpia la salida ruidosa de un
   boton. Para ello, cuando detecta un cambio en la entrada,
13 //              congela la salida y comienza un contador.
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////
21
22 module Debounce_veto(
23     input rst,              // reset global
24     input clk,              // reloj global
25     input noisy,            // entrada ruidosa
26     output reg clean        // salida limpia
27     );
28
29          parameter NDELAY = 5000000;       // retardo del
               contador interno
30          parameter NBITS = 23;             // numero de bits
               del contador interno
31
32          reg [NBITS-1:0] count;            // cuenta interna
33
34          always @(posedge clk)
35             if (rst == 0)                  // reset
36                begin
37                    clean <= noisy;
38                    count <= 0;                 // no estoy
                         vetando al resetear
39                end
```

```
40          else if (count != 0)              // Si estoy vetando,
               entonces
41             count <= count-1;          // reduzco el contador en
                  una cuenta
42          else if (clean != noisy)      // Si no estoy vetando y
               mi entrada cambio, es un cambio valido
43              begin                     // entonces
44                  clean <= noisy;      // Salida se convierte en
                       la entrada que cambio
45                  count <= NDELAY;     // inicio contador para
                       comenzar a vetar nuevas entradas
46              end
47  endmodule
```

Finalmente, generamos el módulo Top que engloba todos los demás módulos e implementa el cronómetro de forma completa.

Listing 7: TopT6.v

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////
3   // Company: IEE2713
4   // Engineer: Angel Abusleme
5   //
6   // Create Date: 2019-11-20
7   // Design Name:
8   // Module Name: T6
9   // Project Name:
10  // Target Devices:
11  // Tool Versions:
12  // Description: Tarea 6. No es la forma mas eficiente de
       hacerla porque tengo muchos
13  //              contadores que operan bajo la base de tiempo de
        100MHz. Seria mas
14  //              adecuado y menos enredado usar un reloj mas
       lento para todo el circuito.
15  //              Traten de entender la forma en que operan los
       circuitos secuenciales:
16  //              Todos tienen el mismo reloj de entrada, pero
       son habilitados por pulsos
17  //              cortos llamados one-shot.
18  //              Observen tambien el uso de la jerarquia para
       dividir y conquistar.
19  // Dependencies:
20  //
21  // Revision:
```

```
22 // Revision 0.01 - File Created
23 // Additional Comments:
24 //
25 //////////////////////////////////
26
27
28 module TopT6(
29     input clk,                  // 100-MHz clock
30     input [15:0]sw,             // Slide switch inputs
31     input reset,                // resetea todo, reset global
32     input btnL,                 // start y stop
33     output [15:0]led,           // Led outputs
34     output [3:0]an,             // 7segment anode, active low
35     output [6:0]seg             // 7segment leds, active low
36     );
37
38     wire [16:0] CounterClock1kHz;   // Cuenta para producir
          reloj de 1kHz
39     wire [9:0] CounterClock1Hz;     // Cuenta para producir
          reloj de 1Hz
40     wire cuenta_100k;               // Se activa cuando la
          cuenta del contador para crear reloj de 1kHz llega a
          100000
41     wire cuenta_1k;                 // se activa cuando la
          cuenta del reloj que avanza a 1kHz llega a 1000
42     wire clk_1kHz;                  // Clock de 1kHz, no es
          flanco sino un pulso largo
43     wire OS_1kHz;                   // one-shot de 1kHz
44     wire clk_1Hz;                   // Clock de 1Hz, es un
          pulso
45
46     wire [3:0] cuentamil;           // Contador de milesimas
47     wire [3:0] cuentacen;           // Contador de centesimas
48     wire [3:0] cuentadec;           // Contador de decimas
49     wire [3:0] cuentauni;           // Contador de unidades
50     wire clk_1kHz_contando;         // Habilita el contador del
           timer
51     wire contando;                  // indica el estado del
          timer, si cuenta o detiene cuenta
52     wire btnL_clean;                // boton sin rebotes
53     wire btnL_clean_OS;             // boton limpio y one-shot
54
55     OneShot OS1khz_mod(clk_1kHz, clk, reset, OS_1kHz);
                    // pulso one shot de 1kHz
56     OneShot OSButton_mod(btnL_clean, clk, reset, btnL_clean_OS)
```

```
    ;    // pulso one shot del boton de inicio/pausa
57  Debounce_veto debounce_mod(reset, clk, btnL, btnL_clean);
58  Four_Seven_Segment Display(cuentamil,cuentacen,cuentadec,
       cuentauni,clk,OS_1kHz,an,seg);
59
60  assign clk_1kHz_contando = clk_1kHz && contando;
61  assign cuenta_100k = (CounterClock1kHz==17'd100_000);
            // Se activa brevemente cuando la cuenta llega a
       100000
62  assign clk_1kHz = (CounterClock1kHz==17'd1);
                     // Cuando la cuenta es 1, la salida esta
       activada y dura 1ms
63  assign cuenta_1k = (CounterClock1Hz==10'd1000);
                  // Se activa brevemente cuando la cuenta
       llega a 1000
64  assign clk_1Hz = (CounterClock1Hz==10'd0);
                       // Cuando la cuenta es 0, la salida
       esta activada y dura harto
65  assign led = sw;
                                                  // podemos
       usarlos para depurar el circuito
66
67  FlipFlopRE contando_mod(clk, reset, btnL_clean_OS, ~
       contando, contando);        // Flip-flop toggle para
       start/pause
68
69  // Contadores principales del timer
70  Contador #(4) cuentamil_mod(.clk(clk), .rst(reset ||
       cuentamil == 4'd10), .en(clk_1kHz_contando), .count(
       cuentamil));
71  Contador #(4) cuentacen_mod(.clk(clk), .rst(reset ||
       cuentacen == 4'd10), .en(clk_1kHz_contando && (cuentamil
        == 4'd9)), .count(cuentacen));
72  Contador #(4) cuentadec_mod(.clk(clk), .rst(reset ||
       cuentadec == 4'd10), .en(clk_1kHz_contando && (cuentacen
        == 4'd9) && (cuentamil == 4'd9)), .count(cuentadec));
73  Contador #(4) cuentauni_mod(.clk(clk), .rst(reset ||
       cuentauni == 4'd10), .en(clk_1kHz_contando && (cuentadec
        == 4'd9) && (cuentacen == 4'd9) && (cuentamil == 4'd9))
       , .count(cuentauni));
74
75  // Este contador cuenta hasta 100000 y luego se resetea
       solo
76  // Su  s e al  de reset es la comparacion entre la cuenta y
       100000
```

```
77      Contador #(17) Contador_1kHz(.clk(clk), .rst(cuenta_100k ||
            reset), .en(1'b1), .count(CounterClock1kHz));
78
79      // Este contador cuenta hasta 1000 y se resetea solo.
80      // Se habilita con la cuenta del contador de 100mil
81      Contador #(10) Contador_1Hz(.clk(clk), .rst(cuenta_1k ||
            reset), .en(clk_1kHz), .count(CounterClock1Hz));
82
83  endmodule
```