Laboratorio #7:
Lógica Secuencial

Escuela de Ingeniería Eléctrica

Universidad de Costa Rica

1 Introducción

1.1 Objetivos del Laboratorio

La lógica secuencial es un pilar fundamental en el diseño digital moderno, permitiendo la implementación de sistemas que dependen del tiempo y del estado previo del circuito. A diferencia de la lógica combinacional, donde las salidas dependen exclusivamente de las entradas actuales, los circuitos secuenciales incluyen elementos de memoria que almacenan información de estados previos.

Este laboratorio tiene como propósito que los estudiantes diseñen, simulen e implementen circuitos secuenciales en Verilog, incluyendo contadores, relojes digitales y sumadores sincrónicos. Además, se abordará la importancia de la sincronización mediante señales de reloj y la gestión de retardos en los sumadores sincrónicos.

A través de esta actividad, los estudiantes lograrán:

- Comprender el funcionamiento de los circuitos secuenciales y su diferencia con la lógica combinacional.
- Diseñar e implementar contadores síncronos y su aplicación en la visualización de números en un display de siete segmentos.
- Construir un reloj digital que mida minutos, segundos y centésimas de segundo.

2 Conceptos teóricos necesarios

2.1 Lógica Secuencial

Los circuitos digitales pueden clasificarse en dos grandes categorías:

- **Lógica combinacional:** La salida depende solo de las entradas actuales (ejemplo: multiplexores, sumadores, comparadores). Hasta el momento sólo se ha trabajado este tipo de lógica en este laboratorio.
- **Lógica secuencial:** La salida depende de las entradas actuales y de estados previos, utilizando elementos de memoria como registros y Flip-Flops.

La lógica secuencial síncronica se basa en una señal de reloj que sincroniza los cambios de estado. Esto es fundamental para diseñar contadores, registros y máquinas de estado.

2.2 Señales de Reloj

Los circuitos secuenciales dependen de una señal de reloj, que establece los instantes en los que se permite actualizar el estado del sistema.

En la FPGA utilizada en este laboratorio, la señal de reloj opera a 50 MHz (20 ns por ciclo), por lo que se necesita una división de frecuencia para visualizar los cambios de estado de manera perceptible en los displays de siete segmentos.

Un divisor de reloj es un circuito que reduce la frecuencia de una señal de reloj. Para lograrlo, se puede utilizar un contador que active una señal de sincronización cada cierto número de ciclos, como el que se presenta más adelante en este laboratorio.

La generación y distribución de la señal de reloj a todo el circuito es vital para asegurarse un funcionamiento correcto, sin embargo, ese tema está más allá del alcance de este curso.

2.3 Contadores Sincrónicos

Un contador síncrónico es un circuito secuencial que incrementa (o disminuye, dependiendo de la configuración) su valor en cada pulso de reloj. A diferencia de los contadores asíncronos, los contadores síncronos presentan menos problemas de retardo, ya que todos los flip-flops cambian de estado simultáneamente.

3 Ejemplo introductorio - Contador

Recuerde que el dispositivo que se usará es el Max 10 10M50DAF484C7G.

En este laboratorio se mostrará el procedimiento para implementar un contador en la FPGA de la tarjeta de desarrollo. La cuenta se mostrará en los *displays* de siete segmentos de la tarjeta como un número decimal y el valor de cuenta máximo será definido por el usuario mediante los interruptores `SW9 - SW0` de la tarjeta de desarrollo. El período de actualización del contador (tiempo que permanece en un valor antes de pasar al valor siguiente) deberá ser constante una vez que se cargue el diseño en la FPGA (no se puede cambiar por el usuario) y deberá ser un tiempo lo suficientemente extenso como para poder identificar claramente la cuenta actual. Adicionalmente, el botón `Key0` funcionará como una entrada de reset para el contador, la cuenta deberá permanecer en cero mientras el botón esté siendo presionado.

3.1 Consideraciones de Diseño

De lo anterior, se pueden establecer puntualmente los siguientes requerimientos para el contador:

- La cuenta se incrementará de uno en uno, de manera periódica.
- La cuenta será desplegada como un número decimal en los *displays* de siete segmentos
- Mientras `Key0` esté presionado, la cuenta será cero.
- El estado de los interruptores `SW9-SW0` definirán el valor máximo que la cuenta alcance.

Considerando lo anterior, hay que tener las siguientes consideraciones de diseño:

- Deberá haber un registro (o Flip-Flop) de varios bits en el que se tenga almacenado el valor actual la cuenta.
- El valor de cuenta más alto posible estará definido por la cantidad total de interruptores y corresponde al caso en que todos los interruptores tengan un estado de uno lógico. Como son 10 interruptores este valor corresponde a $2^{10} - 1 = 1023$ (en binario: `10'b1111111111`), de esto también se puede concluir que el registro en el que se almacene la cuenta deberá ser de 10 bits (este valor debe ser constante porque el usuario podrá modificar la cuenta máxima en cualquier momento) y que se requerirá utilizar a lo sumo 4 de los 6 *displays* de siete segmentos.
- Cada *display* mostrará uno de los valores posicionales de la cuenta (unidades, decenas, centenas ...) por lo que se deben calcular estos valores de la cuenta. Adicionalmente, como el número se desplegará en formato decimal, cada *display* solo podrá mostrar un dígito del 0 al 9.

- Para lograr que la cuenta se incremente a intervalos regulares se requiere de una señal periódica, es decir, una señal de reloj.
- Las entradas del módulo contador serán los interruptores, el botón de reset y la señal de reloj.
- Las salidas del módulo serán las señales de control de los *displays* de siete segmentos.

3.2 Diseño Modular

El contador podría ser implementado como un único módulo de Verilog, sin embargo se utilizará un diseño modular, ya que esto brinda mayor claridad sobre las señales internas del contador y su funcionamiento, además de permitir reutilizar estos submódulos en diseños futuros.

En la Figura 1 se presenta un diagrama de bloques del diseño que se implementará y el funcionamiento de los submódulos se especifica en las siguientes secciones.

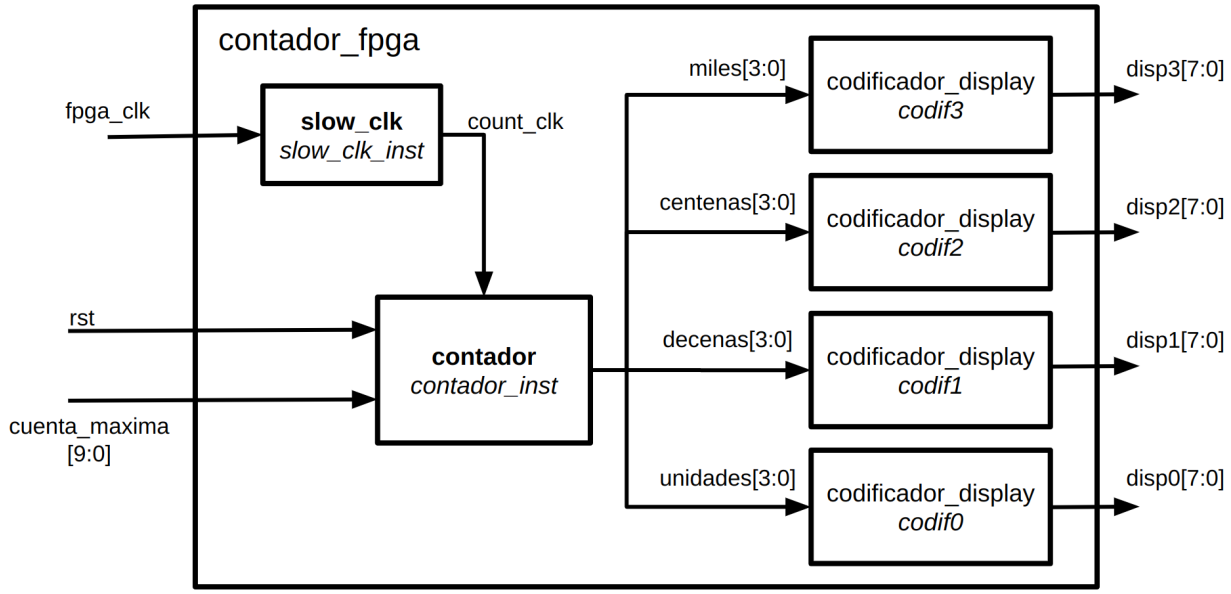

Figura 1: Topología del contador que se implementará en la FPGA.

3.3 Módulo `slow_clk`

El contador deberá incrementar la cuenta de manera periódica, por lo que se utilizará una señal de reloj para sincronizar el aumento de la cuenta en alguno de sus flancos.

En la página 24 del **Manual de Usuario de la tarjeta de desarrollo**, se especifican las señales de reloj conectadas a la FPGA, de las cuales se definen como de uso para lógica del usuario las señales `MAX10_CLK1_50` y `MAX10_CLK2_50`, en el resto del documento, esta señales de reloj se denotarán como `fpga_clk`.

Los flancos postivos de `fpga_clk` se podrían utilizar para coordinar el incremento de la cuenta, sin embargo es importante considerar que la frecuencia de las señales es de 50MHz, es decir, un período de 20ns. Si se sincronizara el incremento de la cuenta a los flancos de este reloj, el valor desplegado sería imperceptible para el ojo humano, pues cambiaría demasiado rápido. Ante esto, se utilizará la señal del reloj de la FPGA (`fpga_clk`) para controlar una señal que produzca flancos de una forma más lenta (`count_clk`).

Para lograr esto, se utilizará `fpga_clk` para realizar una cuenta intermedia (`int_count`) que se incrementará en cada flanco de `fpga_clk` y que, cuando alcance un valor específico (`MAX_CNT`) hará que la señal lenta (`count_clk`) se active. La señal `count_clk` será la que se utilice para el aumento de la cuenta que será desplegada. La señal será regresada a un estado bajo después de un ciclo de `fpga_clk`.

El valor de `MAX_CNT` se deberá elegir cuidadosamente para que los valores mostrados en los *displays* permanezcan visibles por un tiempo prudente para que puedan ser visualizados por el ojo humano.

La Figura 2 muestra un diagrama de ondas de cómo se debería comportar el circuito. Note que la señal `int_count` aumenta ante cada flanco del reloj, hasta que llega a `MAX_CNT`. Una vez que está en dicho valor, la señal `count_clk` se activa durante un ciclo, y el conteo se reinicia. Es importante aclarar que los valores de las señales se leen al momento del flanco, por lo que `count_clk` se activa **el ciclo posterior** a que se alcanzó el valor de `MAX_CNT`, no en el mismo ciclo.

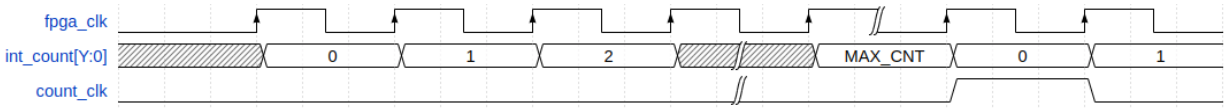

Figura 2: Diagrama de ondas para el `slow_clk`

La estructura descrita se ve ejemplificada en el siguiente módulo de Verilog:

slow_clk.v

```
//Declaracion del modulo slow_clk
module slow_clk (fpga_clk, count_clk);
  //La entrada es el clk rapido, que viene de la tarjeta
  //Debe maperse a MAX10_CLK1_50 o MAX10_CLK2_50
  input fpga_clk;
  //La salida es el clk del contador, el cual es mas lento
  output reg count_clk;

  //Se define el parametro MAX_CLK
  //como el valor maximo al que hay que reiniciar la cuenta
  //Usted debe modificar el valor "X" por uno apropiado
  parameter MAX_CLK = X;
  //Mediante un log2 se calcula la cantidad de bits
  //necesarios para almacenar la cuenta
  parameter Y = $clog2(MAX_CLK);
  //Se declara el conteo intermedio
  reg [Y:0] int_count = 0;

  //En cada flanco positivo del reloj
  always @(posedge fpga_clk) begin
    //Si la cuenta ya llego a MAX_CLK
    if (int_count == MAX_CLK) begin
      //Reinicie la cuenta
      int_count <= 0;
      //Y active el clk lento
      count_clk <= 1;
    end
    //Si aun no llega a MAX_CLK
    else begin
      //Aumente la cuenta
      int_count <= int_count + 1;
      //Y mantenga (o ponga) el clk lento en 0
      count_clk <= 0;
    end
  //Termina always
  end
//Termina modulo
endmodule
```

3.4 Módulo `contador`

La señal `count_clk` generada en el módulo `slow_clk` es utilizada como entrada de este módulo, y se utiliza para controlar el contador que maneja los displays, el cual se genera en este módulo. La cuenta se almacenará y se irá incrementando en la señal `count`.

Los flancos positivos de la señal de reloj (`count_clk`) se utilizan para incrementar la cuenta (`count`). La cuenta deberá seguir incrementando a menos de que ocurra un reset (utilizando `Key0`) o se llegue al valor máximo deseado (`cuenta_maxima`), el cual se configura por el usuario utilizando los switches `SW9-SW0`.

Para finalizar, se deben obtener los dígitos de cada posición de significancia, y para esto se utiliza el cociente entero.

A continuación se muestra el código de Verilog para implementar el módulo descrito anteriormente:

counter.v

```
//Declaracion del modulo
module contador (rst, cuenta_maxima, count_clk,
            unidades, decenas, centenas, miles);
  //Se definen las entradas:
  //El reset (debe maperse a Key0)
  //El clk de conteo (viene de slow_clk)
  input rst, count_clk;
  //El valor maximo a desplegar (SW9-SW0)
  input [9:0] cuenta_maxima;
  //Se definen las salidas:
   //Los registros para enviar a los displays
  //Cada uno contiene una cifra de significancia
  output reg  [3:0] unidades, decenas, centenas, miles;

 //El conteo a desplegar
  reg  [9:0] count;

  //En cada flanco positivo del reloj
  always @(posedge count_clk) begin
    //Si hay un reset, regrese a 0 el conteo
    if (rst == 0) count <= 0;
    //Si se alcanzo la cuenta maxima
    //regrese a 0 el conteo
    else if (count == cuenta_maxima) count <= 0;
    //Sino, siga contando normalmente
    else count <= count + 1;

    //Se calcula cada cifra
    //La parte entera de dividir el conteo entre 1000
    //da las unidades de millar
    miles <= count/1000;
    //Si se quitan las UM (mediante el residuo)
    //Y se divide por 100, obtenemos centenas
    centenas <= (count%1000)/100;
    //Si se quitan las centenas (mediante el residuo)
    //Y se divide por 10, obtenemos decenas
    decenas <= ((count%1000)%100)/10;
    //Si se quitan las decenas (mediante el residuo)
    //obtenemos unidades
    unidades <= ((count%1000)%100)%10;
  end
endmodule
```

3.5 Módulo `codificador_display`

Cada valor posicional de la cuenta se deberá mostrar en uno de los *displays* de siete segmentos de la tarjeta de desarrollo. Por ejemplo, si se quisiera mostrar el número '529' se debería mostrar un '9' en el *display* con posición menos significativa, un '2' en el siguiente *display*, un '5' en el siguiente y un '0' en el *display* más significativo.

Para mostrar un dígito en los *displays* de siete segmentos se deben encender los segmentos correspondientes a ese dígito y mantener apagados los demás. En la Figura 3 se muestra cómo se mapean los segmentos del display HEX0. Para revisar los mapeos adecuados para cada display utilce las guías de usuario (**Terasic** o

Programa Académico de FPGAs de Intel). Recuerde que los segmentos son **activos en bajo**.

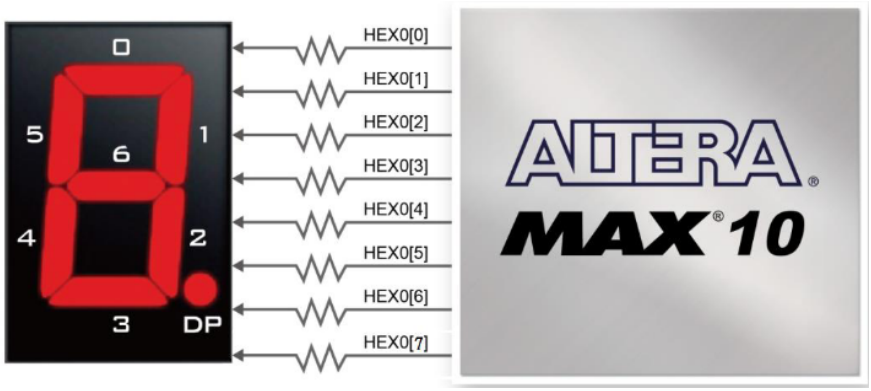

Figura 3: Mapeo de señales del display de 7 segmentos HEX0

Siguiendo el mapeo, para mostrar un '0' se deberían encender (poner en 0) los segmentos 0, 1, 2, 3, 4 y 5). Para no tener que repetir el análisis sobre cómo mostrar cada dígito se definirá un módulo que reciba el número que se desea mostrar (en 4 bits) y produzca la salida correspondiente para desplegar ese número en el *display* y se realizarán instancias de este módulo para cada *display* que se requiera controlar.

A continuación se presenta un fragmento de este módulo:

codificador_display.v

```
//Declaracion del modulo
//Este modulo se instanciara 4 veces
//Una para cada cifra significativa
//Pero solo lo escribimos una vez
//Luego solo lo instanciamos varias veces
//Cada una con entradas disintas
module codificador_display (bin, seg);
  //La entrada es el numero en binario
  //producido por el modulo contador
  input        [3:0] bin;
  //La salida es el bus de 8 bits (7seg + punto decimal)
  //Con cuales segmentos activar (en bajo)
  output reg   [7:0] seg;

  //Cada vez que cambie la entrada
  always @(bin) begin
    //Si la entrada es un 0
    if (bin == 0) seg = 8'b00111111;
    //Si la entrada es un 1
    else if (bin == 1) seg = 8'b00000110;
    //Encarguese usted de hacer los demas
    //...
  //Termina always
  end
//Termina modulo
endmodule
```

Nota: Para el módulo anterior se estableció que los bits de `reg` posean el mismo orden de significancia

que los segmentos del *display* para producir el número indicado dentro del `if`, es decir, se debe asignar `seg[0]` a `HEX[0]` y así sucesivamente, de lo contrario el patrón desplegado será incorrecto.

Módulo `fpga_counter`

Finalmente, el módulo superior se define y configura de acuerdo a lo mostrado en la Figura 1, instanciando los submódulos mencionados en las secciones anteriores e interconectándolos como se muestra a continuación:

fpga_counter.v

```
//Declaracion del top
module contador_fpga (rst, cuenta_maxima, fpga_clk,
                      disp0, disp1, disp2, disp3);
  //Se definen las entradas
  //El fpga_clk
  //Debe maperse a MAX10_CLK1_50 o MAX10_CLK2_50
  //Y el rst (debe mapearse a Key0)
  input rst, fpga_clk;
  //La cuenta maxima (SW9-SW0)
  input [9:0] cuenta_maxima;
  //Las salidas a los displays (HEX3-HEX0)
  output wire  [7:0] disp0, disp1, disp2, disp3;

  //Se declaran interconexiones entre modulos
  wire count_clk;
  wire [3:0] unidades, decenas, centenas, miles;

  //Se instancia el modulo slow_clk
  //Y se conectan las entradas y salidas apropiadas
  slow_clk slow_clk_inst (.fpga_clk(fpga_clk),
           .count_clk(count_clk));

  //Se instancia el modulo contador
  //Y se conectan las entradas y salidas apropiadas
  contador contador_inst (.rst(rst),
           .cuenta_maxima(cuenta_maxima),
           .count_clk(count_clk), .unidades(unidades),
           .decenas(decenas), .centenas(centenas),
           .miles(miles));

  //Se instancian los codificadores a 7seg
  //Se instancia 4, uno por cifra
  //Y se conectan las entradas y salidas apropiadas
  codificador_display codif0 (.bin(unidades),
                              .seg(disp0));
  codificador_display codif1 (.bin(decenas),
                              .seg(disp1));
  codificador_display codif2 (.bin(centenas),
                              .seg(disp2));
  codificador_display codif3 (.bin(miles),
                              .seg(disp3));
endmodule
```

Creación del proyecto en Quartus Prime

Cree un proyecto en Quartus Prime siguiendo la secuencia de pasos presentada en las guías anteriores. Incluya todos los módulos desarrollados en este ejemplo y asegúrese que el módulo `fpga_counter` sea el módulo superior del proyecto. Programe la FPGA de la tarjeta de desarrollo y compruebe su funcionamiento. Utilice la asignación indicada en la Tabla 1.

Señal	Componente
Unidades de Millar	HEX3
Centenas	HEX2
Decenas	HEX1
Unidades	HEX0
Cuenta máxima	SW9-SW0
Reset	Key0

Tabla 1: Asignación de pines para este problema para el contador

4 Trabajo en Clase

Recuerde que el dispositivo que se usará es el Max 10 10M50DAF484C7G.

Diseñe e implemente un circuito que funcione como un temprorizador de tiempo real. Los minutos, segundos y centésimas de segundo deberán desplegarse en los *displays* de siete segmentos de la tarjeta de desarrollo de la manera indicada en la Tabla 2.

El temporizador deberá reiniciarse en la cuenta siguiente a los 59 minutos, 59 segundos y 99 centésimas de segundo. Finalmente, el temporizador deberá detenerse cuando se presione el botón `Key0` y deberá reanudar su funcionamiento cuando se deje de presionar.

Señal	Componente
Minutos	HEX5-HEX4
Segundos	HEX3-HEX2
Centésimas de Segundo	HEX1-HEX0

Tabla 2: Asignación de pines para este problema para el contador

Para la DEMOSTRACIÓN deberá mostrar los diseños descritos funcionando de forma adecuada. Apenas finalice cada diseño, puede solicitar a la persona docente hacer el demo. La calificación se repartirá de la siguiente manera:

- Temporizador: 100%