# Versión 261C.01

**Carrera: INGENIERÍA EN ELECTRÓNICA**

**Asignatura:** 3718 - Lenguajes Descriptivos de Hardware

**Tema:** Diseño Sincrónico de Circuitos Digitales

**Unidad:** 1.0 - 4.0

**Objetivo:** Repasar los fundamentos de los circuitos digitales, entender el diseño y funcionamiento de los circuitos lógicos programables (FPGAs), diseñar circuitos sincrónicos utilizando síntesis mediante lenguajes descriptivos como Verilog

**Competencias a desarrollar:**

- Concepción, diseño y desarrollo de proyectos de ingeniería en electrónica.
- Gestión, planificación, ejecución y control de proyectos de ingeniería en electrónica.
- Utilización de técnicas y herramientas de aplicación en la ingeniería en electrónica.
- Generación de desarrollos tecnológicos y/o innovaciones tecnológicas.
- Desarrollo de una actitud profesional emprendedora.
- Aprendizaje continuo
- Actuación profesional ética y responsable.
- Comunicación efectiva.
- Desempeño en equipos de trabajo.
- Identificación, formulación y resolución de problemas de ingeniería en electrónica

**Descripción de la actividad:**

1. Tiempo estimado de resolución: Cuatrimestral
2. Metodología: Simulador Logisim-Evolution; Xilinx Vivado; Xilinx Vitis
3. Forma de entrega: No obligatoria, desarrollo en laboratorio
4. Metodología de corrección y feedback al alumno: Presencial y por Miel.

# A- Repaso

**Consejo**

Los siguientes ejercicios requieren repasar los temas vistos en las materias previas de técnicas digitales. Dejamos en MIeL una lista de reproducción con videos de repaso sobre estos temas.

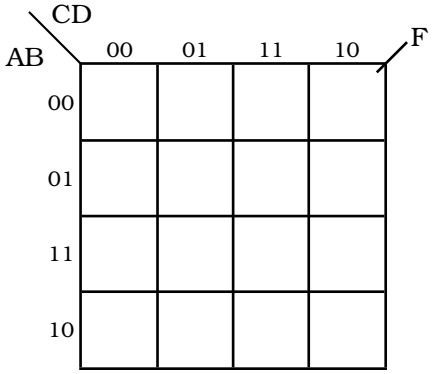
**A.1** Dada la siguiente tabla de verdad para la función F(A,B,C,D)

| A | B | C | D | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | X |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

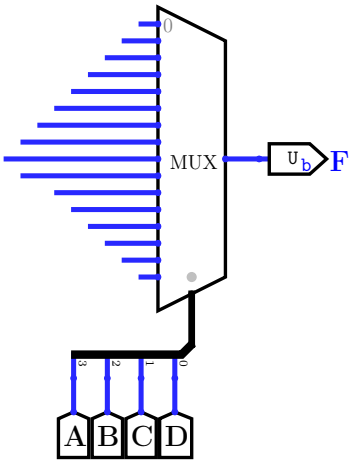
(a) Escriba la función F(A,B,C,D) en primera forma canónica (suma de productos).
**F(A,B,C,D)=**

(b) Escriba la función F(A,B,C,D) en segunda forma canónica (producto de sumas).
**F(A,B,C,D)=**

(c) Simplifique F (minitérminos) utilizando el mapa de Karnaugh y luego implemente el circuito simplificado utilizando solo compuertas NAND.



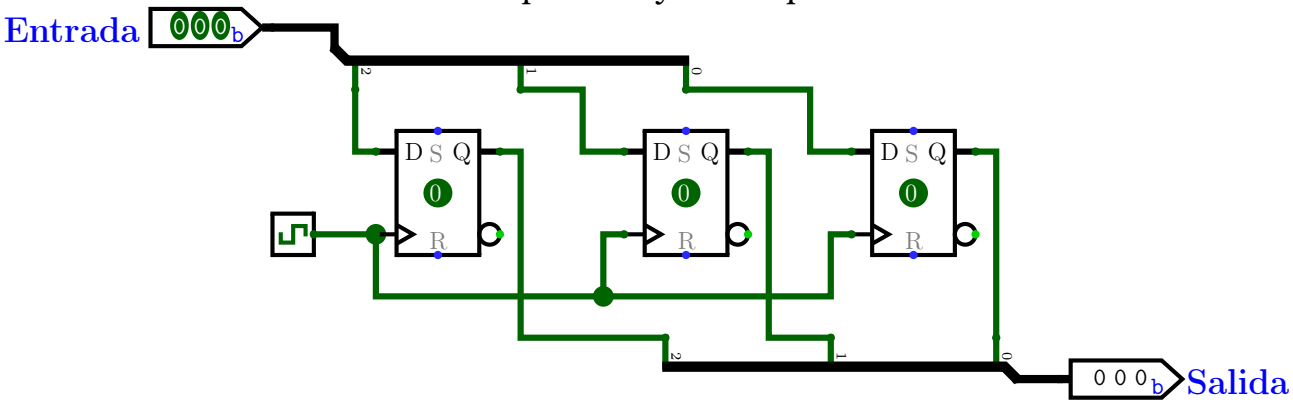

(d) Implemente el circuito utilizando un MUX de 4 entradas de selección.



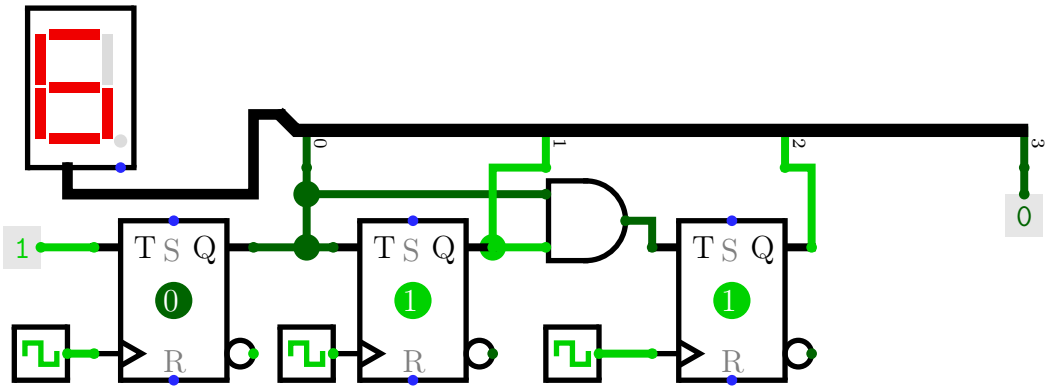

(e) Para los dos circuitos previos (simplificado y MUX) simule ambos en logisim-evolution y verifique que la salida F genera los mismos valores en todos los casos.

**A.2** Utilizando logisim-evolution diseñe un conversor BCD a 7 segmentos. Puede implementar el mismo de la forma que crea mas conveniente (simplificando 7 funciones, con MUXes, con una ROM de LookUp Table, etc).


**A.3** Utilizando 3 FlipFlops tipo D, implemente en logisim-evolution el siguiente registro Latch de 3 bits con entrada paralelo y salida paralelo.

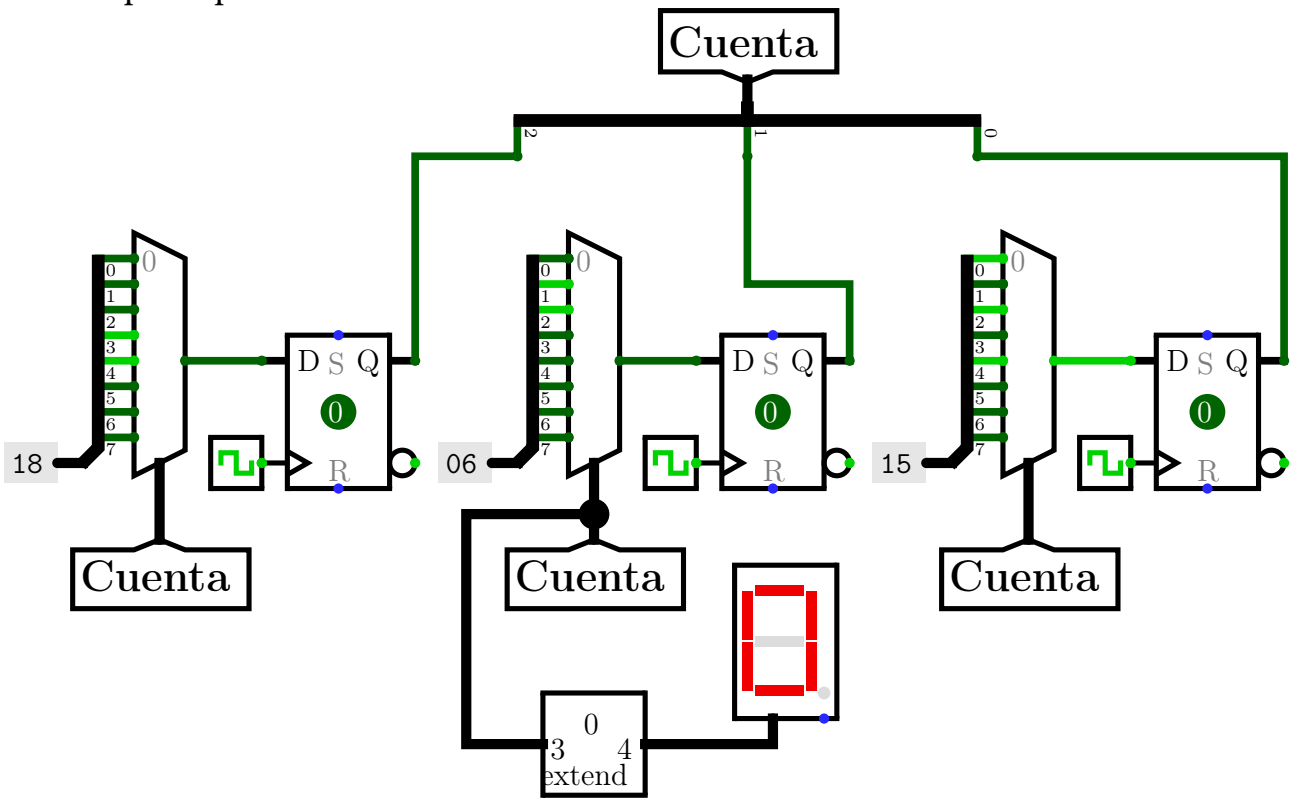



**A.4** Utilizando 3 FlipFlops tipo T, implemente en logisim-evolution el siguiente contador binario sincrónico ascendente módulo 8 (cuenta de cero a siete).




**A.5** Modifique el circuito anterior para que el módulo de cuenta sea 6 (es decir cuenta desde cero hasta cinco).

**A.6** Implemente el siguiente circuito en logisim-evolution. Indique el módulo de cuenta del mismo y luego modifique los valores en las entradas de datos de los MUXes para que el módulo de cuenta sea 7.




**A.7** Modifique los valores en las entradas de los MUXes del punto anterior para que genere como secuencia el código de Gray en 3 bits (000;001;011;010;110;111;101;100). Recomendamos escribir la tabla de verdad con todos los valores posibles de 3 bits como entrada y como salida el siguiente valor de la secuencia. Ej: Cuando la entrada es 011 el siguiente valor es 010, cuando la entrada es 111 el siguiente valor es 101, y asi.

# B- FPGA Didáctica

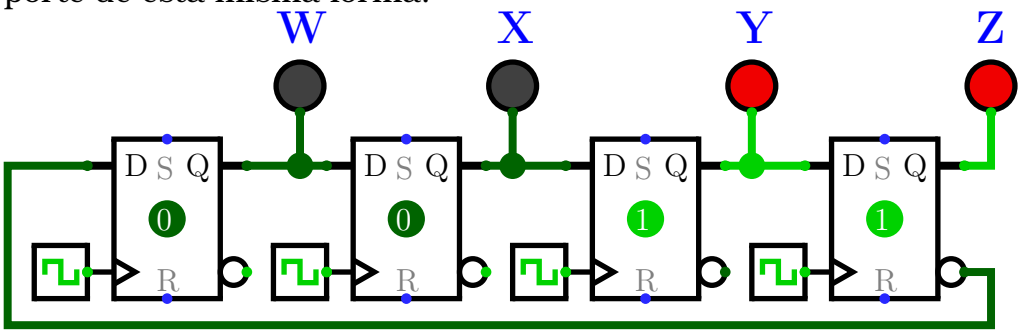
**Aclaración**

Los siguientes ejercicios deben ser implementados en el modelo de FPGA Didáctica para losigim-evolution (FPGAGHO.circ) provisto en https://github.com/edgardogho/CursoVerilog

**B.1** Utilizando como referencia el diseño FPGAGHO_MAYORIA.circ cree un nuevo circuito llamado FPGAGHO_NOT.circ que implemente una simple compuerta NOT, utilizando como entrada A y como salida el LED W. Luego complete la tabla con los valores del bitstream.

**B.2** Utilizando como base el circuito del punto anterior, cree un circuito que implemente la función XOR sobre las 4 entradas (A,B,C,D) y represente el valor en el LED W.

**B.3** Implemente un contador Johnson de 4 bits sobre los LEDs de salida (W,X,Y,Z). Puede ver el funcionamiento esperado en:
https://www.youtube.com/watch?v=6GqrWavMfec
Como referencia tiene a continuación un contador Johnson utilizando FlipFlops tipo D. Debe encontrar los valores correctos en la FPGAGHO para que se comporte de esta misma forma.




**B.4** Implemente un contador en anillo (One-Hot) de 4 bits (1000,0100,0010,0001) sobre los LEDs de salida. Tenga en cuenta que otros valores (ej: 1100) son invalidos. Puede ver el funcionamiento esperado en:
https://www.youtube.com/watch?v=c5yZDTOcpRE
Utilice esta tabla como referencia. Tenga en cuenta que cuando el circuito se enciende el reset pone todos los FF en cero, por ende debe tener una instancia en donde pasa de 0000 a 1000, y luego no vuelve a pasar por 0000. La solución es similar a los ejercicios **A.6** y **A.7**.

| Clock | bin | $C_3$ | $C_2$ | $C_1$ | $C_0$ | $C'_3$ | $C'_2$ | $C'_1$ | $C'_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 8 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 4 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| | – | X | X | X | X | X | X | X | X |

**B.5** Implemente un contador binario ascendente de 4 bits. Recomendamos analizar primero la siguiente tabla:

| Bin | $C_3$ | $C_2$ | $C_1$ | $C_0$ | $C'_3$ | $C'_2$ | $C'_1$ | $C'_0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 10 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 13 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Conecte un display hexadecimal en paralelo a las salidas de los LEDs para ver el dígito correspondiente.

**B.6** Modifique el circuito anterior para limitar el módulo de cuenta a 10, es decir, cuenta de 0 a 9.

**B.7** Modifique el circuito anterior para lograr que se comporte como un contador binario descendente módulo 10, es decir, cuenta de 9 a 0.

# C- Asignaciones Booleanas

**Entorno de desarrollo**

Los siguientes ejercicios deben ser implementados en proyectos de Vivado. Recuerde apagar la síntesis incremental en cada nuevo proyecto. No utilice espacios en el directorio del proyecto ni caracteres raros, solo letras y números.

**C.1** Defina los siguientes módulos en Verilog. En cada caso los nombres de las entradas son A y B, y la salida como F.

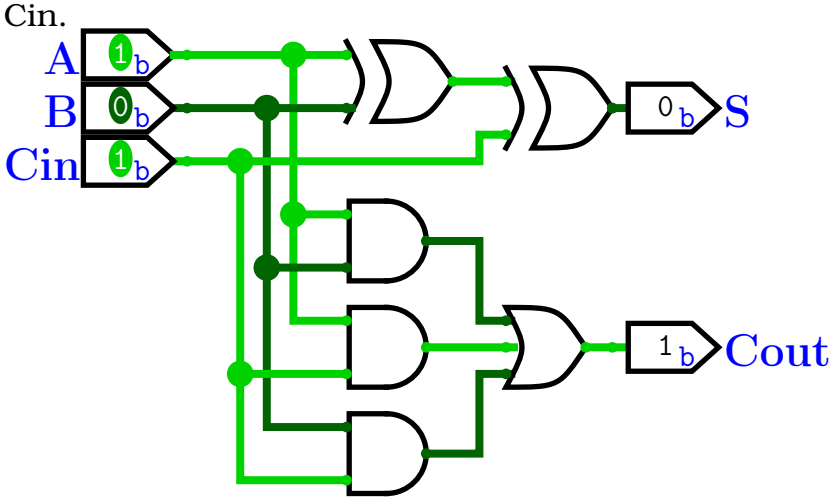
(a) ModuloAND

(b) ModuloOR

(c) ModuloXOR

Para cada módulo tiene que definir un nuevo archivo (ej: ModuloAND.v). Escriba luego un archivo de Testbench (ej: ModuloANDTB.v) donde pruebe todas las combinaciones de entrada para A y B.

**C.2** Defina un nuevo módulo en Verilog llamado FullAdderComp (archivo: FullAdderComp.v) cuyas entradas sean A, B y Cin , y sus salidas sean S y Cout. Para resolver el módulo, tiene que instanciar las compuertas de forma similar al ejemplo de multiplexor instanciando compuertas. Escriba un archivo de Testbench (FullAdderCompTB.v) donde pruebe todos los valores posibles de A,B y Cin.




**C.3** Utilizando la tabla de verdad del FullAdder, defina un nuevo módulo en Verilog llamado FullAdderDef (archivo: FullAdderDef.v) cuyas entradas sean A, B y Cin, y sus salidas sean S y Cout. Para resolver el módulo, tiene que implementar cada salida definida en forma canónica (suma de productos o producto de sumas). Ej: en el caso que S=1 cuando Cin=0, A=0, B=1 .. debe incluir entonces el término (!Cin & !A & B). Esta implementación es similar al ejemplo multiplexor definido en un *assign*. Utilice un archivo de Testbench (FullAdderDefTB.v).

| $Cin$ | $A$ | $B$ | $Cout$ | $S$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**C.4** Defina un nuevo módulo llamado FullAdder2Bits donde tenga como entradas A[1:0] y B[1:0] y como salidas S[1:0] y Cout. Debe comportarse como un sumador de 2 bits con salida de carry. Ej: si las entradas A=01 y B=11 , entonces la salida es S=00 y Cout=1. Utilice la forma que crea más conveniente, pero recomendamos instanciar módulos ya definidos. Genere un archivo de Testbench.

**C.5** Utilizando **asignaciones condicionales** implemente un nuevo módulo llamado Decoder.v, el mismo tiene como entrada A[1:0] y como salida S[3:0] y se comporta como un decoder. Ejemplo: si A=10 entonces S=0100, si A=00 entonces S=0001. Escriba un Testbench.

**C.6** Utilizando la placa asignada en el laboratorio (Zybo), genere un nuevo proyecto de Vivado basado en esta placa. Luego **importe** los archivos definidos en los puntos anteriores. Tiene que poder implementar en la FPGA los dos puntos anteriores. Utilice como entrada los sliders y como salida 4 leds.

## D- Contadores

**Clocks Lentos**

Los siguientes ejercicios donde se encadenan contadores modifican la señal de clock. Esto se conoce como "clock lento" y es un mal diseño. En la siguiente sección convertimos los mismos a un diseño sincrónico correcto, pero comenzamos de esta forma para trazar un paralelo con lo visto en materia previas.

**D.1** Defina un módulo llamado FFD que funcione como un FlipFlop tipo D con los siguientes puertos:

- Entradas:
  - Clock
  - Reset
  - D
- Salidas:
  - Q

Realice un Testbench en donde compruebe el funcionamiento del mismo.

**D.2** Defina un módulo llamado FFT que funcione como un FlipFlop tipo T con los siguientes puertos:

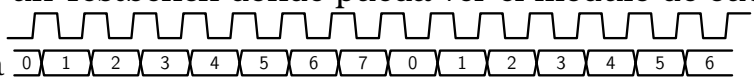
- Entradas:
  - Clock
  - Reset
  - T
- Salidas:
  - Q

Realice un Testbench en donde compruebe el funcionamiento del mismo.

**D.3** Utilizando el módulo FFD implemente un registro Latch similar al visto en el punto **A.3** en la página 3. Escriba un Testbench para probar el mismo. Este ejercicio **NO** puede ser probado en hardware ya que la señal de clock debe conectarse a un oscilador y no a un botón de entrada GPIO.

**D.4** Implemente un registro latch de 3 bits sin utilizar el FFD. Debe realizar el mismo con registros de Verilog al estilo de *'reg [2:0] dato'*. Nuevamente realice un Testbench y compare con el punto anterior. ¿Cual de los dos es mas fácil de modificar para agregar bits?

**D.5** Utilizando el módulo FFT implemente un contador binario sincrónico ascendente de 3 bits (módulo 8) idéntico al visto en el ejercicio **A.4** en la página 3. Realice un Testbench donde pueda ver el módulo de cuenta completo. Ej:




**D.6** Implemente el contador del punto anterior sin utilizar FFD pero utilizando registros de Verilog al estilo de *'reg [2:0] contador'*. Utilice el Testbench anterior para probar el mismo. Luego modifique el diseño para que el módulo de cuenta sea de 16 (4 bits). Compare la relación entre la frecuencia de salida del bit mas significativo contra el menos significativo (divisor de frecuencia).

**D.7** Implemente un contador binario sincrónico ascendente módulo 10 (cuenta desde 0 a 9). Debe realizar un Testbench para probar el funcionamiento del mismo.

**D.8** Utilizando el contador del punto anterior, encadene otro contador módulo 10 para que se genere un módulo conjunto de 100, desde 00 hasta 99. Realice un Testbench para ver el módulo generado, que debe estar compuesto de dos salidas de 4 bits, una para el dígito menos significativo y otra para el más significativo.

**D.9** Diseñe un contador binario ascendente con registro de 27 bits. El mismo tendrá como módulo $2^{27} = 128Mega$. Luego implemente un contador binario ascendente de 4 bits. Este ultimo tiene como clock el bit más significativo (26) del contador anterior. De esta forma cada 128Mega cuentas del primer contador se produce un pulso completo que es tomado como pulso de clock para el segundo contador. Utilizando la placa Zybo conecte 4 LEDs a la salida del segundo contador y tome como entrada de clock del primer contador el oscilador de 125MHz. Pruebe lo que ocurre si modifica el primer contador para que tenga 28 bits y también para que tenga 26 bits.

**D.10** Diseñe un circuito que posea una salida llamada LED que se mantenga en uno durante 10 pulsos de clock y luego se mantenga en cero los siguientes 10 pulsos de clock (exactos). Pruebe este circuito en un Testbench. Recomendamos utilizar un FlipFlop tipo T al cual su clock recibe un pulso cuando un contador módulo 10 pasa a cero.

**D.11** Modifique el circuito anterior para que se genere el cambio en el LED cuando pasan tantos pulsos como la frecuencia del oscilador de 125MHz de la placa Zybo. El resultado debería ser que el LED se enciende un segundo, luego se apaga un segundo. Luego modifique el circuito para que el período sea de 500ms.

**D.12** Teniendo en cuenta los diseños anteriores, realice un contador de 4 bits utilizando los LEDs de la placa Zybo y su oscilador de 125MHz en donde cada elemento de cuenta se mantiene por un segundo. Al cabo de 16 segundos el contador debe volver a cero.

**D.13** Utilizando el decoder visto en el punto **C.5** de la página 8, implemente un contador en anillo conectando a la entrada del decoder un contador de 2 bits. Simule el mismo en un Testbench

**D.14** Adapte el diseño del punto anterior para que se pueda utilizar en la placa Zybo con el oscilador de 125MHz y 4 LEDs de salida los cuales deben cambiar una vez por segundo.

## E- Debouncer

**Diseño sincrónico**

En esta sección no se permiten clocks encadenados. Toda señal de clock proviene del oscilador y es la unica referencia a todos los circuitos secuenciales. Los retardos se realizan con señales del tipo CE (Clock Enable).

**E.1** Defina un módulo llamado FFDCE que funcione como un Flip Flop tipo D con los siguientes puertos:

- Entradas:
  - Clock
  - Reset
  - CE
  - D
- Salidas:
  - Q

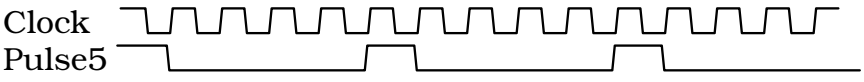
Realice un Testbench en donde compruebe el funcionamiento del mismo. La entrada de CE es la que indica cuando el valor de la entrada D es capturado.

**E.2** Defina un módulo llamado FFTCE que funcione como un Flip Flop tipo T con los siguientes puertos:

- Entradas:
  - Clock
  - Reset
  - CE
  - T
- Salidas:
  - Q

Realice un Testbench en donde compruebe el funcionamiento del mismo. La entrada de CE es la que indica cuando el valor de T es considerado.

**E.3** Implemente un circuito que tenga una entrada llamada Clock y una salida llamada Pulse5 que genere un pulso cada 5 pulsos del clock de entrada. Debe funcionar con flanco ascendente. Recuerde que todo circuito secuencial lleva también una señal de reset.




Simule este circuito con un Testbench.

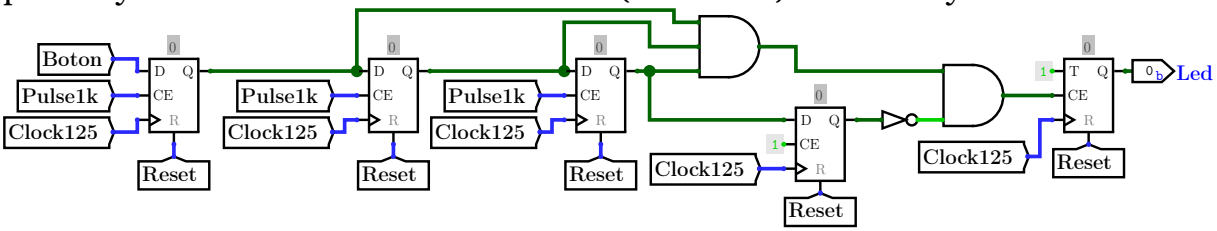
**E.4** Modifique el circuito anterior renombrando la salida a Pulse125M. Esta salida debe generar el pulso cada 125 millones de pulsos del clock de entrada. Luego utilizando un FlipFlop tipo T, conecte esta salida al CE del FlipFlop T, dejando la entrada T del mismo en 1 permanente. Conecte la salida del FlipFlop T a un LED. Corra este proyecto en una placa Zybo utilizando el oscilador de 125MHz como entrada. Indique lo que ocurre con el LED.

**E.5** Utilizando como base los circuitos anteriores genere un contador de 4 bits cuya salida sean los LEDs de la placa Zybo de forma tal que el periodo de cuenta sea un segundo.

**E.6** Utilizando como base los circuitos anteriores genere un contador de 4 bits con entrada de CE conectada a un botón. El contador solo debe contar cuando el botón se encuentra pulsado. ¿Puede realizar incrementos de a uno?

**E.7** Diseñe un generador de pulso similar al anterior pero que el período entre pulsos sea 1ms. Nombre esta salida Pulse1k. Asumiendo que la frecuencia del oscilador es 125.000.000Hz, entonces debe generar un pulso cada (125000000/1000) 125.000 pulsos de clock.

**E.8** Implemente el siguiente circuito utilizando como entrada Pulse1k la salida de un módulo instanciado del punto anterior. Implemente el mismo en la placa Zybo con el oscilador de 125MHz (Clock125) un botón y un LED.




**E.9** Modifique el circuito anterior quitando el FlipFlop T y conecte la salida de la última AND a una señal de salida llamada PulseBoton. Esta salida debe generar un pulso cada vez que se aprieta el botón (post debouncing). Verifique el mismo con el osciloscopio en modo trigger one shot detectando el pulso.

**E.10** Implemente un contador de 4 bits ascendente. El mismo solo debe contar cuando se presiona un botón, generando solo un incremento por cada pulso del botón.

**E.11** Implemente un contador UpDown de 4 bits. El mismo va a poseer un CE para UP (CEUP) y un CE para DOWN (CEDOWN). En el caso de ambos valer uno va a tomar prioridad el UP. Luego con dos botones en la placa Zybo utilice instancias del debouncer para controlar el valor del contador.

## F- Componentes internos

**Parametros**

En estos ejercicios el objetivo principal es manejar los parámetros que permiten instanciar diseños con valores ajustables.

**F.1** Implemente un módulo llamado LUTCOMB3 (LookUp Table Combinacional 3 entradas). Posee 3 entradas y una salida. Debe instanciar el componente interno de la FPGA llamado LUT3.
https://docs.xilinx.com/r/en-US/ug953-vivado-7series-libraries/LUT3
Debe poder cambiar el valor de INIT en cada instancia. Luego implemente una compuerta AND de 3 entradas instanciando LUTCOMB3 con el valor correspondiente en INIT.

**F.2** Implemente una restador de 3 bits (un bit de minuendo A, un bit de sustraendo B y un bit de borrow de entrada). El mismo debe tener como salida un bit de Resta (R) y un bit de Borrow de salida (Bout). A tal fin instancie dos módulos LUTCOMB3 con sus valores de INIT correspondientes a la siguiente tabla de verdad.

| $A$ | $B$ | $B_{in}$ | $B_{out}$ | $R$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Escriba un Testbench donde pueda probar este restador.

**F.3** Utilizando la sintaxis de Generate, implemente un restador de dos números de cuatro bits. Realice un Testbench que realice distintas restas y verifique el resultado.

**F.4** Implemente un contador sincrónico ascendente de 4 bits donde el módulo de cuenta pueda ser definido mediante un parámetro. Ej: si el parámetro dice 12, el contador debe ser módulo 12.

**F.5** Implemente un contador sincrónico módulo $2^N$ donde el valor de N es configurado por un parámetro. Ej: si N es 5 entonces el contador es módulo 32.

## G- Core IP y Memorias

**Aclaración**

Estos ejercicios utilizan componentes internos de la FPGA. Los que utilizan memorias ROM o RAM pueden ser simulados, pero aquellos que utilizan Clocking Wizard deben probarse sobre las placas. En el laboratorio disponemos de placas Zybo, Nexys 4 DDR y Basis 3.

**G.1** Instance un módulo Clocking Wizard, en donde la entrada sea el oscilador de la placa (100MHz o 125Mhz segun la placa) y la salida sea siempre un clock a 8,3886MHz

**G.2** Tomando como Clock la salida del módulo del punto 1, implemente un contador de 23 bits. Genere una salida de pulso cuando el contador se encuentra en 23'd0. Con esta salida genere el CE de un Flip Flop T cuya salida vaya a un LED.

**G.3** Implemente una memoria ROM single port utilizando el módulo Block Memory Generator. En el puerto A debe Width 8 y Depth 128 (7 bits de address, cada posición de 8 bits). Utilizando un archivo COE defina una secuencia de bytes aleatoria para cada posición de memoria. Luego incluya esta memoria ROM en un módulo que posea un contador de 7 bits (alimentando el address) que se incremente cada un segundo. La salida de datos de la memoria ROM maneja LEDS. Puede simular este circuito con un Testbench o utilizar una de las placas del laboratorio.

**G.4** Implemente una memoria RAM single port (palabras de 8 bits, 16 palabras en total). La entrada de datos proviene de 8 switches. Utilice un contador de 4 bits alimentado con un circuito debouncer conectado a un boton. Esos 4 bits son el address de la RAM. Luego un switch adicional se usa para definir Read o Write en la RAM. Otro circuito debouncer con otro botón se utiliza de CE para la memoria RAM. La salida de la memoria RAM se cablea a 8 leds (pueden ser los 7 segmentos o leds). Escriba las 16 palabras de memoria con una secuencia y luego reproduzca la misma utilizando los botones. Puede simular este ejercicio en una placa Basis 3 o Nexys 4 DDR (ya que la Zybo no tiene los sliders suficientes). También puede simularlo en un Testbench.

## H- CPU Microblaze

**Xilinx Vitis**

Estos ejercicios requieren utilizar Vitis y una placa que soporte el Microblaze RISC-V. Las unicas disponibles son Basis, Nexys y Arty. No puede utilizarse la placa Zybo ya que contiene su propio procesador hardware integrado.

**H.1** Implemente un diseño basado en el procesador Microblaze. Debe utilizar un periférico AXI_aurtlite como UART y AXI_GPIO con alguna salida y/o entrada GPIO. Esto puede ser para cualquiera de las placas Digilent disponibles (Nexys 4 DDR, Basys 3 , Arty S7). Exporte el diseño como un archivo XSA (Hardware file) para que pueda ser utilizado en Vitis.

**H.2** Utilizando el siguiente programa de prueba
https://raw.githubusercontent.com/Xilinx/embeddedsw/master/XilinxProcessorIPLib/drivers/gpio/examples/xgpio_example.c
haga titilar un LED y envie un mensaje por puerto serie. A tal fin genere el código en Vitis que utilice la plataforma definida en el punto anterior.