ELO212: Laboratorio de Sistemas Digitales
Semestre I-2020
Guía de Actividades Sesión 4

2, 3, 5 de Junio de 2020

1. Objetivos

Los objetivos para esta sesión son:

- Familiarizarse con los conceptos prácticos básicos sobre frecuencia y período de reloj y su manipulación.
- Experimentar con los conceptos de aritmética en circuitos digitales.
- Diseñar, implementar, y verificar un esqueleto de una Unidad Aritmética y Lógica (ALU) básica.

2. Trabajo previo

El trabajo planteado en esta sección está diseñado para que sirva de preparación para el trabajo asociado a las actividades que si serán evaluadas durante la sesión. Este trabajo no será evaluado directamente, pero el no realizarlo con la debida atención puede afectar su rendimiento en la sesión. La nota asociada a la sesión será completamente basada en la evaluación de las actividades prácticas indicadas en la siguiente sección.

(A) Cuando se crea un nuevo proyecto en Vivado, se ofrece la opción de crear un diseño basado en RTL. ¿Qué significa RTL en el contexto de diseño de sistemas digitales?

(B) Analice cuidadosamente el siguiente código en SystemVerilog que describe un divisor de frecuencia de reloj, donde `COUNTER_MAX` es un parámetro que permite al usuario setear un número entero positivo arbitrario. Como se ha visto en las sesiones anteriores, el uso de parámetros permite aumentar la regularidad y reutilización de sus módulos. Se recomienda revisar el uso de parámetros en SystemVerilog, ya que será clave para su trabajo durante el resto del semestre.

```
1
2 module clock_divider
3 #(parameter COUNTER_MAX = PUT_NUMBER_HERE)
4 ( input  logic clk_in,
5   input  logic reset,
```

```
  output logic clk_out );

   localparam DELAY_WIDTH = $clog2(COUNTER_MAX);
   logic [DELAY_WIDTH-1:0] counter = 'd0;

   // Resetea el contador e invierte el valor del reloj de salida
   // cada vez que el contador llega a su valor maximo.

  always_ff @(posedge clk_in) begin
    if (reset == 1'b1) begin
      // Reset sincronico, setea el contador y la salida a un valor conocido
      counter <= 'd0;
      clk_out <= 0;
    end else if (counter == COUNTER_MAX-1) begin
      // Se resetea el contador y se invierte la salida
      counter <= 'd0;
      clk_out <= ~clk_out;
    end else begin
      // Se incrementa el contador y se mantiene la salida con su valor
    anterior
      counter <= counter + 'd1;
      clk_out <= clk_out;
    end
  end

endmodule
```

También puede analizar la siguiente versión del `clock_devider`, la cual tiene explícitamente separadas las etapas combinacionales y secuenciales:

```
module clock_divider_separated
  #(parameter COUNTER_MAX = PUT_NUMBER_HERE)
  (
  input  logic clk_in,
  input  logic reset,
  output logic clk_out );

   localparam DELAY_WIDTH = $clog2(COUNTER_MAX);
   logic [DELAY_WIDTH-1:0] counter = 'd0;


  logic [DELAY_WIDTH-1:0] counter_next;
  logic clk_out_next;

  //Logica combinacional para describir counter_next
  always_comb
  begin
      // Resetea el contador e invierte el valor del reloj de salida
      // cada vez que el contador llega a su valor maximo.
    if (counter == COUNTER_MAX - 1)
    begin
      counter_next = 'd0;
    end
    else
    begin
      counter_next = counter + 'd1;
    end
```

```
28    end
29
30    //Logica combinacional para describir clk_out
31    always_comb
32    begin
33      if(counter == COUNTER_MAX - 1)
34      begin
35        clk_out_next = ~clk_out;
36      end
37      else
38      begin
39        clk_out_next = clk_out;
40      end
41    end
42
43    //Logica secuencial para descibir counter y clk_out
44    always_ff @(posedge clk_in)
45    begin
46      if(reset == 1'b1)
47      begin
48        counter <= 'd0;
49        clk_out <= 0;
50      end
51      else
52      begin
53        counter <= counter_next;
54        clk_out <= clk_out_next;
55      end
56    end
57  endmodule
```

Una vez entendido el código, derive una expresión general que permita obtener la frecuencia del reloj de salida `clk_out` como función de la frecuencia del reloj de entrada `clk_in` y el parámetro `COUNTER_MAX`. Utilizando la expresión obtenida, indique el valor de `COUNTER_MAX` para obtener un reloj de salida `clk_out` con frecuencia de 30 Hz a partir de un reloj de entrada `clk_in` con período de 10 ns. ¿Cuántos bits necesitaría como mínimo para representar correctamente este número?

(C) Algunos archivos fuente de SystemVerilog contienen al inicio una directiva de compilación del tipo `` `timescale UU/PP ``, donde `UU` y `PP` son unidades de tiempo. ¿Que función cumple esta directiva y que significan sus argumentos? ¿Que sucede si no se incluye esta directiva de compilación? [1]

(D) Explique la diferencia entre una señal de reset sincrónica y una señal de reset asincrónica en un circuito secuencial basado en flip-flops. Escriba un snippet de código en SystemVerilog que describa la funcionalidad de un flip-flop con reset sincrónico y otro con reset asincrónico. ¿Que tipo de reset debería utilizar para las FPGAs que se utilizan en el curso?

(E) Una regla de oro asociada a buenos modales y prácticas en el diseño digital con FPGAs dice que hay que evitar a toda costa utilizar *latches*. Como se ha mencionado, **en este curso está**

[1] La directiva `` `timescale `` es heredada de Verilog clásico. SystemVerilog introdujo las keywords `timescale` y `timeprecision`, las cuales aplican a cada módulo en forma local. Para diseños nuevos, se recomienda usar `timescale` y `timeprecision`.

totalmente prohibido utilizar latches en sus diseños. Si su diseño final contiene latches, ni siquiera será revisado. La mayoría de las veces los latches aparecen por errores involuntarios en la descripción del diseño asociados a omisiones o errores de tipeo en el código.

Explique que es un *latch* y como se diferencia de un flip-flop. Investigue y de ejemplos de la o las razones más comunes para que la herramienta de síntesis lógica infiera latches a partir de su descripción de hardware. ¿Por qué los latches se consideran inadecuados para implementaciones en FPGA? ¿Por qué existen los latches como componentes de circuito si en general no se recomienda su uso en este curso? Discuta, justifique, y reporte claramente sus fuentes de información.

3. Actividades evaluadas.

Las siguientes actividades serán revisadas durante la sesión de laboratorio. Cuando complete una actividad, debe avisar al profesor o ayudante para mostrarala funcionando, responder preguntas asociadas a su diseño, y recibir la calificación por la actividad presentada.

Se recomienda encarecidamente que adelante su trabajo y llegue al menos con las partes fundamentales probadas. La estadística histórica indica que no alcanzará a terminar el trabajo si no llega con algo avanzado a la sesión.

Esta demás repetir a esta altura del semestre que todo problema de diseño debe empezar con un plan y una descripción grupal sobre que es lo que deben implementar. En el contexto de la asignatura, esto implica realizar un diagrama de alto nivel de los módulos que necesitarán (cajitas), y como se conectarán entre ellos (flechitas que conectan las cajitas). No toque la herramienta de diseño sin haber realizado este paso previo. Se recuerda que las actividades no se revisarán y tampoco se responderán consultas si no tiene un diagrama de bloques de su diseño. El pensar que *"tengo todo en mi cabeza, solo necesito unos minutos para escribirlo"*, es un salto directo al fracaso.

Durante la evaluación se pueden hacer preguntas aleatorias sobre detalles técnicos de su diseño y las decisiones que tomaron.

Para el proceso de revisión consideren lo siguiente:

- La revisión de cada actividad es en formato **todo o nada**. No hay puntaje parcial por tener partes de la actividad o algo que este *"casi funcionando"* o *"que le falte poquito"*.
- A parte de verificar la funcionalidad, durante la revisión de la actividad se le puede solicitar mostrar los reportes de los procesos de síntesis lógica e implementación. Debe entender los mensajes entregados por la herramienta para estas etapas.
- Mostrar el diagrama de bloques y no tener latches en su diseño son requisitos para todas las actividades. La actividad no se revisará si no se cumple lo anterior.
- Las actividades son revisadas en el orden que están planteadas. No se asignará puntaje a una actividad si no se ha completado la anterior.
- Deben seguir las especificaciones de diseño. Si se dice que las entradas son P y Q, no entregue su diseño con entradas A y B. Para los requerimientos que no están explícitamente indicados

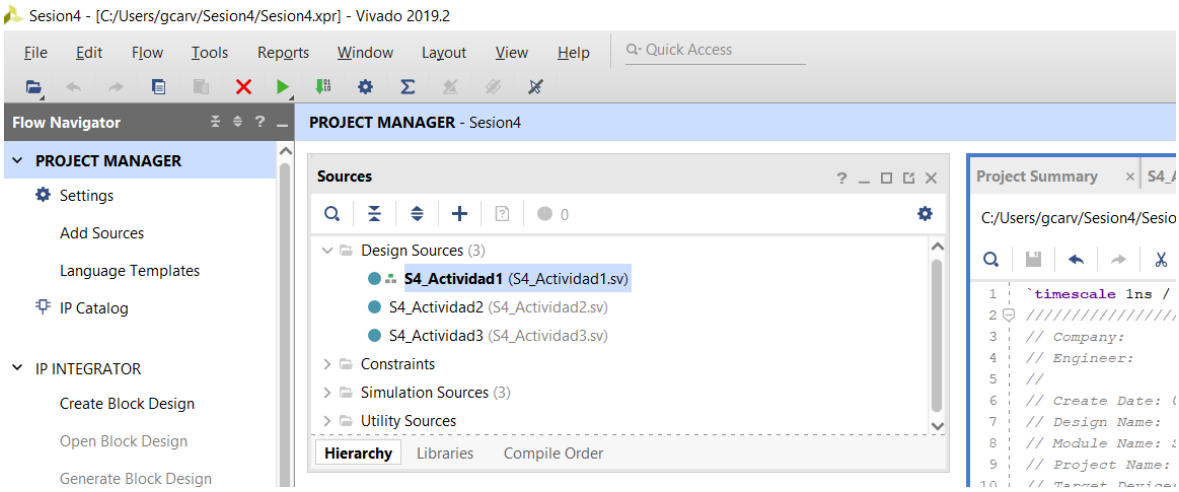

Figura 1: Estructura del proyecto para entrega.

en el enunciado, puede tomar las decisiones que estime conveniente. Estas decisiones deben quedar documentadas como comentarios en el código, y deben ser consecuentes con ellas.

- Durante la sesión deberá mostrar al staff de su paralelo las actividades funcionando mediante resultados de simulación y deberá responder preguntas sobre ello. Cualquier integrante del grupo debe ser capaz de responder las preguntas. Si las respuestas no son satisfactorias, no se revisará la actividad.

- La revisión realizada durante la sesión cumple un caracter preliminar. Una vez que considere su diseño terminado y mostrada la funcionalidad básica durante la sesión, debe enviar los archivos fuente de su proyecto al profesor del paralelo. La evaluación final de cada actividad estará sujeta a una verificación funcional exhaustiva posterior a la sesión.

- Para el envío de sus archivo de diseño, genere un solo proyecto de Vivado con nombre `GPNN`, donde `P` es el número de su paralelo y `NN` es el número de su grupo (por ejemplo, para el grupo 3 del paralelo 1, el nombre del proyecto será `G103-Sesion4`). El proyecto debe incluir un *top module* independiente por cada actividad. Nombre los archivos y el *top module* como `S4-Actividad1`, `S4-Actividad2` y `S4-Actividad3`, según corresponda. La Figura 1 muestra un ejemplo de como debe lucir la estructura de su proyecto con los archivos fuente base. Dentro de cada archivo fuente base puede agregar todos los modulos y archivos que considere necesarios.

- Si alguna actividad no pasa la verificación exhaustiva, se considerará como no entregada a tiempo. En este caso se le avisará al grupo para que entreguen una versión corregida durante la sesión siguiente. En este caso, aplican los descuentos por atraso como está definidos en el reglamento. Es su responsabilidad hacer todas las verificaciones y validaciones necesarias para asegurarse que el diseño entregado cumple con los requerimientos.

- Más detalles sobre el proceso de evaluación de experiencias se pueden ver en el reglamento de evaluación de la asignatura.

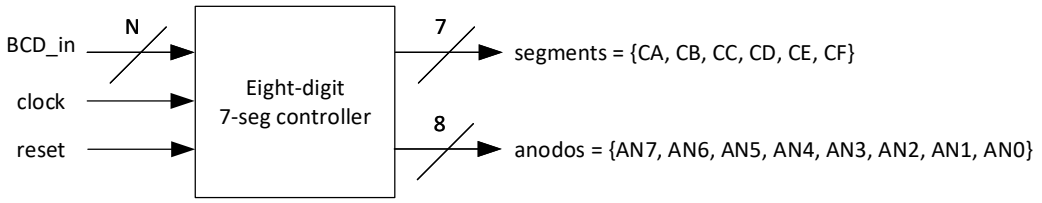

Figura 2: Diagrama de alto nivel para el controlador de display de 7 segmentos incluido en la tarjeta Nexys4DDR/NexysA7. Las señales de salida concuerdan con las señales descritas en el datasheet de la tarjeta.

3.1. Multiplexación temporal para controlador de multiples displays. (25 puntos)

Describa, sintetice, y verifique la funcionalidad de un módulo que permita controlar el display de 7 segmentos incluido en la tarjeta Nexys4DDR/NexysA7 para mostrar el valor en hexadecimal de un número de N bits. La Figura 2 muestra un diagrama de entrada-salida para el módulo a implementar. Asuma que el módulo recibe como entrada de datos un número codificado en BCD de N bits (por simplicidad, considere $N = 32$). Las salidas del módulo corresponden a las señales necesarias para mostrar el valor hexadecimal del número de entrada en los displays disponibles en la tarjeta. Debe utilizar el concepto de *Time Division Multiplexing* (TDM) para poder compartir las señales de salida entre todos los displays.

Su módulo para esta actividad debe contener las señales de entrada y salida indicadas en la Figura 2, **utilizando exactamente los mismos nombres de señales y el mismo orden en la declaración de las señales multibit**. Si considera necesario agregar señales adicionales, agréguelas y explíquelas en su código. Asuma que el reloj de entrada viene ajustado a la frecuencia necesaria para que el número se vea estático en los displays al iterar la activación de los ánodos en cada ciclo de reloj.

Revise cuidadosamente el datasheet para determinar la polaridad necesaria en las señales para encender los displays (notar que no siempre un 1 significará que algo está encendido).

3.2. Generación de relojes de distinta frecuencia a partir de un reloj base. (25 puntos)

El módulo para división de reloj entregado en la sección anterior es solo una posible implementación para generación de relojes con distinta frecuencia a partir de una frecuencia base. Notar que el módulo solo permite generar frecuencias aproximadas en función del parámetro de entrada. Además, presenta limitantes en términos de usabilidad debido a que el usuario necesita hacer cálculos manualmente para poder ingresar los parámetros correctos.

Modifique el archivo base del divisor de reloj para que en lugar de un valor máximo de cuentas, el usuario ingrese la frecuencia del reloj base de entradas en MHz y la frecuencia deseada del reloj de salida en MHz. En base a estos dos parámetros, el módulo debe calcular internamente los valores necesarios para obtener una frecuencia de salida aproximada a la indicada en el parámetro.

Asumiendo que el reloj de entrada es de 100 MHz, describa un modulo que instancie multiples copias del modulo base, utilizando en cada instancia los parámetros para generar relojes de 50, 30, 10, y 1 MHz. La Figura 3 muestra un diagrama de alto nivel del módulo que se pide diseñar.

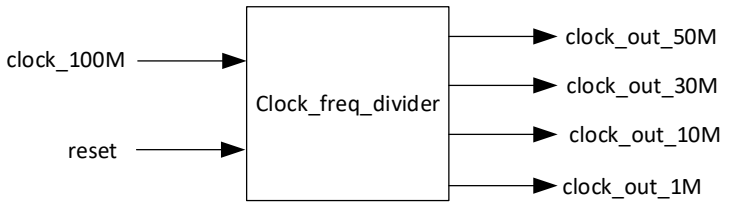

Figura 3: Diagrama de alto nivel para el módulo generador de frecuencias de reloj.

3.3. Unidad Aritmética y Lógica (ALU) básica. (50 puntos)

Las ALU [2] son circuitos combinacionales que permiten realizar distintas operaciones aritméticas enteras y lógicas sobre operandos de entrada. Las ALU son una parte fundamental de cualquier arquitectura de microprocesadores modernos, y trabajarán con ella en diversos cursos de especialidad.

La Figura 4 muestra un diagrama de alto nivel de una ALU genérica, con sus principales entradas y salidas. Una ALU básica tiene como entradas dos operandos de N bits y una señal de selección que indica la operación a realizar sobre los operandos ingresados (el número M de bits de la señal de selección dependerá del número de operaciones distintas que puede realizar la ALU). La salida de la ALU es el resultado de la operación, que tiene el mismo número de bits que los operandos. Además, normalmente se incluyen bits de *status flags* que entregan información de excepciones para la última operación realizada de acuerdo a la siguiente descripción (para mayores detalles, puede revisar el link anterior, cualquier libro de sistemas digitales o arquitectura de computadores, o hacer una búsqueda simple en la web):

- N es 1 si el resultado es negativo.
- Z es 1 si el resultado es cero.
- C es 1 si la última operación generó un carry-out.
- V es 1 si la última operación generó un overflow.

En esta actividad debe implementar un core de una ALU básica, con ancho de bits de datos parametrizable, de acuerdo a las siguientes especificaciones:

- La ALU tiene dos operandos de entrada A y B con ancho de bits configurables por medio de un parámetro al momento de instanciar el módulo. El ancho por defecto es de 8 bits.
- Las operaciones a ejecutar sobre los operandos A y B se configurarán por medio de la señal OpCode de 2 bits, de acuerdo a lo siguiente:

  con los botones de la tarjeta Nexys4 DDR de acuerdo a lo siguiente (revise alguna referencia de SystemVerilog para conocer más sobre los operadores aritméticos y lógicos):

  – Si OpCode=00, la ALU ejecutará la suma de A y B.

[2] https://en.wikipedia.org/wiki/Arithmetic_logic_unit

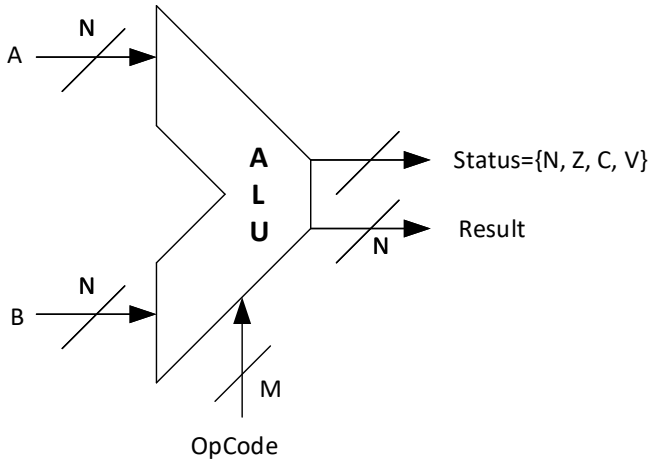

Figura 4: Diagrama de alto nivel de una ALU. Los operandos y resultado son de N bits. La señal `OpCode` especifica la operación a realizar sobre los operandos, la cual viene especificada por medio de una tabla (el número de bits M depende del número de operaciones que la ALU puede realizar. La ALU también tiene una salida de bits de status que indica propiedades de la operación (por ejemplo, si el resultado es cero, si hubo overflow, etc.).

- Si OpCode=01, la ALU ejecutará la resta de A y B.
- Si OpCode=10, la ALU ejecutará el OR bit a bit entre A y B (A|B).
- Si OpCode=11, la ALU ejecutará el AND bit a bit entre A y B (A&B).
- El resultado de todas las operaciones realizadas por la ALU debe tener el mismo número de bits que los operandos.
- La ALU es totalmente combinacional, es decir, debe reevaluar instantáneamente el resultado ante cualquier cambio en alguna de sus señales de entrada.

Verifique el funcionamiento de la ALU realizando operaciones de referencia. Busque ejemplos de operaciones que generen condiciones de excepción para verificar el funcionamiento de los flags. La interrogación asociada a la revisión considerará preguntas sobre estas operaciones [3].

[3]Debe manejar los conceptos básicos de aritmética binaria con número finito de bits y representaciones de números con signo y sin signo.