ELO212: Laboratorio de Sistemas Digitales
Guía de Actividades Sesión 3

4, 5, 7 de mayo de 2021

1. Requisitos de entrada.

Para las actividades de esta sesión debe cumplir con los siguientes requerimientos:

- Haber **completado y entendido** las actividades de las sesiones anteriores.
- Haber revisado previo a la sesión el video *capsula_sesion_3.mp4*, disponible en el repositorio de clases. Este video contiene información crítica para adelantar actividades de esta guía.
- Entender la diferencia entre bloques procedurales `always_comb` para lógica combinacional y `always_ff` para lógica secuencial sincrónica.
- Entender el funcionamiento e implementación de contadores sincrónicos simples.
- Entender la descripción y uso de módulos parametrizados.

2. Objetivos.

Los objetivos para esta sesión son:

- Entender el funcionamiento y modelamiento de multiplexores y decodificadores binarios.
- Experimentar con la generación de relojes con distinta frecuencia a partir de un reloj base.
- Descripción de sistemas basados en *Time Division Multiplexing (TDM)* para compartir recursos limitados.

3. Trabajo previo a la sesión.

Analice con sus compañeros de grupo y responda las siguientes preguntas. Entender en detalle las cosas que se indican será necesario para poder avanzar en esta y sesiones posteriores. Al inicio de la sesión de laboratorio se responderán dudas asociadas a estos tópicos (solo se responderán dudas específicas, no se hará clase).

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

- 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 ejemplo mínimo en SystemVerilog que describa la funcionalidad de un flip-flop con reset sincrónico y otro con reset asincrónico (un `always_ff` por cada uno), y compare sus resultados mediante simulación. ¿Que tipo de reset debería utilizar para las FPGAs que se utilizan en el curso?

- Una regla de oro asociada al diseño digital con FPGAs dice que hay que evitar a toda costa utilizar *latches*. Como se ha mencionado, **en este curso está 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 y se estudian los latches como componentes de circuito si en general no se recomienda su uso en este curso? ¿Hay alguna forma de describir el comportamiento de un latch en SystemVerilog? Discuta, justifique, e indique claramente sus fuentes de información.

4. Actividades Guiadas.

4.1. Decodificador binario.

Un decodificador binario es un circuito de N entradas y 2^N salidas, en donde el valor en las entradas especifican cual de las salidas tendrá un valor alto, mientras que todas las demás salidas se mantienen en bajo (configuración one-hot)[2].

Escriba un código en SystemVerilog basado en descripción procedural usando `always_comb` que describa la funcionalidad un decodificador binario. Prepare un testbench que permita realizar una simulación exhaustiva para verificar la funcionalidad. Muestre sus resultados y discuta con el staff.

4.2. Divisor de frecuencias de reloj simple.

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

[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 y resuelve varias ambigüedades del formato antiguo. Para diseños nuevos, se recomienda usar `timescale` y `timeprecision`, aunque Xilinx sigue usando en sus templates autogenerados el formato antiguo.

[2] Pueden revisar el libro guia de Harris o la descripción en Wikipedia como referencia rápida.

entero positivo arbitrario al momento de instanciar un módulo. Como se ha visto en las sesiones anteriores, el uso de parámetros permite aumentar la regularidad y reutilización de sus módulos. Se recomienda estudiar y entender muy bien el uso de parámetros en SystemVerilog, ya que será clave para su trabajo durante el resto del semestre en términos de reutilización de módulos previamente diseñados y probados.

```
1
2  module clock_divider
3  #(parameter COUNTER_MAX = PUT_NUMBER_HERE)
4  ( input  logic clk_in,
5    input  logic reset,
6    output logic clk_out );
7
8  /* localparam permite definir constantes internas al modulo mediante expresiones
       matematicas complejas. El signo $ especifica una llamada al sistema (
       computador en que corre la herramienta) y estas expresiones no son
       sintetizables, sino que se evaluan numericamente al momento de la sintesis
       logica para obtener una constante. Es decir, al momento de la sintesis esta
       expresion se convierte en una constante que se puede usar para algun
       parametro de diseno interno, como por ejemplo, el ancho de bits de un bus en
       base a un parametro del modulo. No es que se este haciendo un circuito capaz
       de calcular un logaritmo (esto ser a bastante complejo). Estos parametros
       son internos al modulo y no son visibles hacia el exterior.
9  */
10   localparam DELAY_WIDTH = $clog2(COUNTER_MAX);
11   logic [DELAY_WIDTH-1:0] counter = 'd0;
12
13   // Resetea el contador e invierte el valor del reloj de salida
14   // cada vez que el contador llega a su valor maximo.
15
16   always_ff @(posedge clk_in) begin
17     if (reset == 1'b1) begin
18       // Reset sincronico, setea el contador y la salida a un valor conocido
19       counter <= 'd0;
20       clk_out <= 0;
21     end else if (counter == COUNTER_MAX-1) begin
22       // Se resetea el contador y se invierte la salida
23       counter <= 'd0;
24       clk_out <= ~clk_out;
25     end else begin
26       // Se incrementa el contador y se mantiene la salida con su valor anterior
27       counter <= counter + 'd1;
28       clk_out <= clk_out;
29     end
30   end
31
32 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?

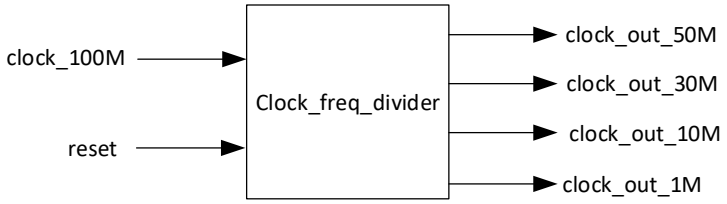

Figura 1: Diagrama de alto nivel para el módulo generador de frecuencias de reloj.

4.3. Implementación de divisor de frecuencias de reloj simple.

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, el módulo presenta limitantes en términos de usabilidad debido a que el usuario necesita conocer la estructura interna del código y hacer cálculos manualmente para poder ingresar los parámetros correctos. **Un objetivo general para el diseño de módulos reutilizables será hacer los códigos tan fáciles de usar y autoexplicativos como sea posible.**

Modifique el archivo base del divisor de reloj utilizado en la actividad anterior para que en lugar de un valor máximo de cuentas, el usuario ingrese al momento de instanciar el módulo la frecuencia del reloj base de entrada 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 un reloj de entrada es de 100 MHz, describa un módulo que instancie múltiples copias del módulo base, utilizando en cada instancia los parámetros para generar relojes de 50, 30, 10, y 1 MHz, respectivamente. La Figura 1 muestra un diagrama de alto nivel del módulo que se pide diseñar.

Siéntase libre de modificar el código base para hacerlo más preciso en la generación de las frecuencias requeridas. Hay muchas formas de hacerlo, unas mejores que otras bajo ciertos contextos. En la práctica, la generación de distintas frecuencias con alta precisión a partir de un reloj base es algo bastante utilizado, y existe hardware especializado para estas tareas. Este proceso se conoce como *frequency synthesis* y se puede implementar mediante phase-locked loop (PLL), Digital Clock Managers (DCM), etc.

4.4. Multiplexación temporal.

Revise en el datasheet de la tarjeta Nexys4DDR/NexysA7 los detalles de configuración y operación de los displays de 7 segmentos.

Una vez entendido el detalle del funcionamiento, diseñe y describa un sistema basado en un contador de 16 bits que muestre en los displays el valor instantáneo del contador en base hexadecimal. Notar que un número de 16 bits se traduce en cuatro símbolos hexadecimales, uno por cada grupo de cuatro bits, para lo cual debe usar cuatro displays. Considere que el reloj del sistema es de 100 MHz y proviene de un módulo externo.

Para lograr el objetivo, debe investigar sobre el principio de *Time Division Multiplexing* (TDM). Utilice como base el video *capsula_sesion_3* disponible en el repositorio de videos del curso. Una vez entienda el concepto, haga un diagrama de alto nivel en *papel y lápiz* que muestre las componentes principales de su sistema y como estarían interconectadas. Como base, su sistema necesita multiplexores, decodificadores, y contadores. Muestre y discuta el diagrama con el staff, explicando lo que asumen y las consideraciones de diseño para su propuesta. Una vez que su diseño haya sido aprobado por alguien del staff, proceda a describir su circuito y a verificar su comportamiento funcional.