ELO212: Laboratorio de Sistemas Digitales
Guía de Actividades Sesión 1

20, 21, 23 de abril de 2021

1. Requisitos de entrada.

- Al menos un integrante de cada grupo debe tener instalada en su computador personal la versión Webpack de Vivado 2020.2. Esta herramienta se usará por el resto del semestre.
- Haber completado las actividades de la sesión anterior y dominar lenguaje técnico mínimo asociado a diseño de sistemas digitales.
- Tener a mano las slides de referencia de SystemVerilog disponibles en Piazza. Idealmente haberlas estudiado y complementado con material adicional.
- **Haber revisado el video complementario *capsula_sesion_1* disponible en el repositorio de clases. Este video contiene información crítica para adelantar actividades básicas de esta guía.**

2. Objetivos.

- Verificar la instalación de la herramienta Vivado de Xilinx para diseño de sistemas digitales basados en FPGA.
- Obtener una primer acercamiento al diseño moderno de sistemas digitales utilizando SystemVerilog.
- Introducir los procesos de descripción y simulación de sistemas digitales utilizando HDLs.
- Proveer una base teórica y práctica para el trabajo de las próximas sesiones a realizarse durante el resto del semestre.

3. Actividades guiadas.

Durante esta sesión revisaremos las primeras etapas del proceso de diseño moderno de circuitos digitales utilizando Hardware Description Languages (HDLs), enfocándonos en lo específico del lenguaje SystemVerilog. Estas etapas consisten en el diseño, descripción, análisis y simulación funcional lógica de circuitos, que son los pasos previos para pasar a las etapas de implementación (*placing &*

routing), generación del archivo de configuración de la FPGA (*bitstream*), y verificación de la implementación física.

Para ilustrar los conceptos fundamentales del proceso de diseño y la operación de las herramientas disponibles, empezaremos describiendo circuitos muy simples, e iremos agregando complejidad en forma incremental para ilustrar el uso de buenas prácticas de diseño. Por muy sencillos que parezcan los diseños básicos, estos permitirán ilustrar los pasos que se harán cada vez más necesarios a medida que vayamos incrementando la complejidad de los sistemas a implementar.

Para todo el resto del curso, tengan en cuenta que los códigos en HDL nos permiten describir la funcionalidad de un sistema en un lenguaje más cercano al humano comparado con una captura esquemática, dejando que una herramienta de diseño, como Vivado, se encargue de elegir el como se implementará esta funcionalidad en base a ciertas restricciones. **Un código en HDL describe un sistema a implementar, no se ejecuta como un programa tradicional. No podemos cambiar un circuito una vez que está implementado**[1]**.**

Durante la sesión nos enfocaremos en obtener una visión intuitiva de la descripción utilizando HDLs y en revisar el uso de la herramienta Vivado para los procesos de visualización preliminar del diseño y simulación. Detalles técnicos acerca de la sintáxis y formalidades de SystemVerilog se revisarán en detalle en las cátedras y en las próximas sesiones. La información entregada en esta sesión será crucial para el trabajo del resto del semestre, por lo que se recomienda reportar y consultar cualquier duda que tenga o problema que detecte en el proceso.

3.1. Introducción a Vivado y creación de un proyecto de diseño.

Siga las indicaciones del video de referencia para la sesión para crear un nuevo proyecto de diseño basado en descripciones del tipo *Register Transfer Level* (RTL) y el chip XC7A100TCSG324-1. No agregue archivos de diseño ni constraints en esta etapa.

Asumiendo que revisó el video previamente a la sesión, el profesor hará un breve repaso sobre el entorno de desarrollo en Vivado. Realice todas las consultas que tengan sobre este simple paso.

3.2. Introducción a descripción de hardware utilizando SystemVerilog.

El profesor dará una breve cátedra para explicar aspectos básicos sobre la descripción de hardware mediante SystemVerilog (tenga a mano las slides asociadas disponibles en Piazza). En esta parte nos enfocaremos en aspectos básicos que permitirán ilustrar el uso de la herramienta de diseño con ejemplos mínimos. Aspectos formales y más avanzados se cubrirán en las cátedras y en las siguientes sesiones.

Es muy importante que los conceptos explicados queden claros. Haga todas las preguntas que considere necesarias. No hay dudas demasiado simples o tontas. Recuerde que los efectos de arrastrar dudas básicas hacia las próximas sesiones puede ser desastrosos.

[1] Existen técnicas avanzadas que permiten reconfigurar partes del circuito *on-the-fly*, lo que se conoce como *partial reconfiguration*. Sin embargo, este es un tópico avanzado que escapa totalmente de este curso.

3.3. Conectando cables.

Agregue al proyecto creado un nuevo archivo de diseño en formato SystemVerilog. Cree un archivo vacío con extensión .sv, y typee el siguiente código [2]:

```
module cable (
    input  logic A,
    output logic B
    );

    assign B = A;
endmodule
```

Siga las indicaciones del profesor para sintetizar el diseño. En esta actividad, el proceso de síntesis lógica solo se realizará para comprobar el correcto funcionamiento de Vivado. Más detalles sobre este proceso se verán en las siguientes sesiones.

Interprete y haga un diagrama de alto nivel en *papel y lápiz* del circuito que se está describiendo. Siga las indicaciones para ver el circuito que la herramienta de diseño infiere a partir de su descripción utilizando la vista de *Elaborated Design*, y compare lo que entrega la herramienta con lo que ustedes infirieron.

3.4. Circuitos combinacionales simples.

Agregue al proyecto creado un nuevo archivo de diseño en formato SystemVerilog. Cree un archivo vacío con extensión .sv, y **tipee** el siguiente código:

```
module logica_simple (
    input  logic A, B, C,
    output logic X, Y, Z
    );

    assign X = A;
    assign Y = ~A;
    assign Z = B & C;
endmodule
```

Verifique que su código no tiene errores de sintáxis. Interprete y haga un diagrama de alto nivel en *papel y lápiz* del circuito que se está describiendo. Siga las indicaciones para ver el circuito que la herramienta de diseño infiere a partir de su descripción utilizando la vista de *Elaborated Design*, y compare lo que entrega la herramienta con lo que ustedes infirieron.

3.5. Simulación funcional y diagramas de tiempo.

Revise el video base de la experiencia para estudiar el proceso de realizar una simulación funcional (*behavioral simulation*) de un diseño por medio de testbenches [3]. Siguiendo las indicaciones del

[2] Note también la estructura de los códigos entregados como ejemplo respecto a la organización, uso de comentarios, indentación, etc. Se espera que sigan buenas prácticas asociadas. La legibilidad de los códigos se considerará en las evaluaciones.

[3] Quiz: ¿Por qué se llama simulación funcional?¿Que otro tipo de simulación hay?

video, agregue un archivo de simulación a su proyecto de diseño y tipee en este el código mostrado a continuación:

```
`timescale 1ns / 1ps

module testbench_simple();    // creacion modulo "dummy"

    logic A, B, C;      // definicion de conexiones virtuales
    logic X, Y, Z;

    logica_simple DUT(      // instancia del modulo a testear
        .A (A),
        .B (B),
        .C (C),
        .X (X),
        .Y (Y),
        .Z (Z)
        );

    initial begin             // aca se asignan valores de prueba o estimulos
        A = 1'b0;             // valores iniciales a t=0
        B = 1'b0;
        C = 1'b0;
        #3                    // retardo de 3 unidades de tiempo basadas en
    timescale
        A = 1'b1;
        #6
        B = 1'b1;
        #4
        C = 1'b1;
    end
endmodule
```

Ejecute la simulación, observe y analice detalladamente los resultados. Luego, modifique el código base para generar distintos patrones de estímulos para las entradas y observe los resultados apuntando a realizar una verificación exhaustiva de su diseño, es decir, que se verifiquen que las salidas son correctas para todos los posibles valores de las entradas.

3.6. Diseño y descripción de reconocedor de números fibbinarios de 4 bits.

Nota: *En esta parte se presenta y describe en detalle el proceso de diseño de un circuito combinacional simple mediante el análisis clásico, comenzando con una descripción funcional hasta llegar al diagrama lógico. Este proceso se describe solo con el fin de ilustrar el proceso de diseño tradicional basado en minimización de expresiones lógicas mediante algebra booleana (esto se ve en detalle en la asignatura ELO211) hasta llegar a una captura esquemática en forma explícita. El trabajo a realizar por Ud. en esta actividad se concentrará en describir, simular, y analizar el circuito resultante de la Figura 1 utilizando SystemVerilog.*

Enunciado. Se especifica que un número es **Fibbinario** si en su representación binaria de magnitud sin signo no posee 1s consecutivos. Para más detalles, revise los siguientes enlaces:

- `http://oeis.org/A003714`
- `https://www.geeksforgeeks.org/httpswww-geeksforgeeks-orgfibbinary-numbers/`

Se quiere diseñar un circuito basado en compuertas lógicas que permita identificar con un bit de salida en alto (1, TRUE, ON, etc.) si la palabra de entrada de 4 bits (`abcd`) corresponde a un número fibbinario, en caso contrario deberá presentar la salida en bajo (0, FALSE, OFF, etc.).

Ejemplos: $1:0001->1$; $5:0101->1$; $6:0110->0$

Desarrollo. Para llegar a una expresión lógica que se pueda implementar directamente con primitivas lógicas básicas [4], se han de realizar los siguientes pasos:

1. Completar la tabla de verdad según los números que se desea identificar en notación binaria, considerando "*a*" como el bit más significativo hasta "*d*" como el bit menos significativo, logrando el siguiente resultado:

a	b	c	d	Fibbinario
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Cuadro 1: Tabla de Verdad para Números Fibbinarios de 4 bits.

2. Generar un mapa de Karnaugh vacío para 4 bits (notar que los mapas se ordenan utilizando codificación Gray):

[4] Técnicamente, hay muchísimas formas de implementar un circuito que cumpla esta función mediante distintas componentes

ab \ cd	00	01	11	10
00				
01				
11				
10				

Cuadro 2: Mapa de Karnaugh vacío para 4 bits.

3. Completar el mapa de Karnaugh con los valores de la tabla de verdad (notar que el mapa contiene exactamente la misma información de la tabla de verdad, solo que ordenada de una forma conveniente que es adecuada para encontrar patrones visuales):

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	0
11	0	0	0	0
10	1	1	0	1

Cuadro 3: Mapa de Karnaugh para Números Fibbinarios de 4 bits.

4. Agrupar los “1” para generar los “mintérminos”:

ab \ cd	00	01	11	10
00	1	1	0	1
01	1	1	0	0
11	0	0	0	0
10	1	1	0	1

Cuadro 4: Mapa de Karnaugh con mintérminos para Números Fibbinarios de 4 bits.

5. La ecuación reducida (“Suma de Productos” al agrupar los “Mintérminos”) resultante es:

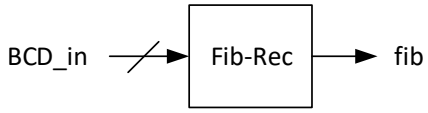

Figura 2: Diagrama de alto nivel de reconocedor de números fibbinarios de 4 bits.

$$f(abcd) = \bar{a}\bar{c} + \bar{b}\bar{c} + \bar{b}\bar{d}$$

6. En base a la expresión obtenida, el circuito que debe armar corresponde al siguiente esquemático:

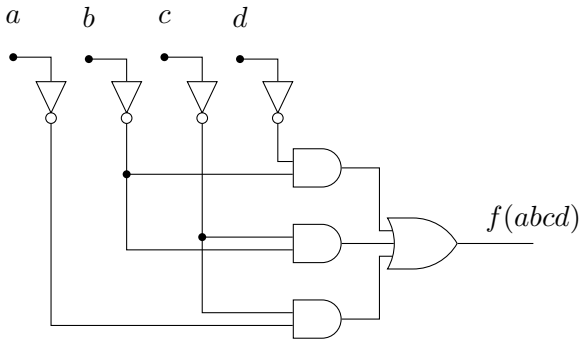

Figura 1: Diagrama Lógico para Números Fibbinarios de 4 bits.

La actividad consiste en generar una descripción en SystemVerilog del diagrama lógico resultante en la Figura 1. Para esto, revise la utilización de operadores lógicos en SystemVerilog (notar que son distintos a los operadores aritméticos).

Utilice el análisis de *Elaborated Design* para comparar lo que la herramienta infiere de su descripción con el diagrama original. Finalmente, genere un testbench para verificar la funcionalidad del circuito descrito mediante una prueba exhaustiva.

3.7. Descripción en base a bloques procedurales.

El profesor dará una breve explicación para ilustrar las capacidades de abstracción que otorgan los bloques procedurales de alto nivel que entrega SystemVerilog. En particular, revisaremos los bloques `always` combinacionales.

Es muy importante que los conceptos explicados queden claros. Haga todas las preguntas que considere necesarias.

3.8. Descripción de alto nivel para reconocedor de números fibbinarios.

Todo proceso de diseño debe partir identificando las señales de entrada y salida del sistema a diseñar, obteniendose como mínimo un diagrama como el mostrado en la Figura 2. El procesamiento a realizar dentro de la *caja negra* lo describiremos usando las decripciones de comportamiento de SystemVerilog.

Utilice el siguiente código para describir la funcionalidad de su circuito (revise el uso de operadores booleanos para evaluar la veracidad del condicional en las slides del curso o en el libro de su preferencia):

```
module fib_rec (
    input  logic [3:0] BCD_in, // BCD 8421 input
    output logic       fib  // High if input is fibbinary
    );

    always_comb begin
        if(BCD_in==4'd0 || BCD_in==4'd1 || BCD_in==4'd2 || BCD_in==4'd4 ||
           BCD_in==4'd5 || BCD_in==4'd8 || BCD_in==4'd9 || BCD_in==4'd10)
            fib = 1;
        else
            fib = 0;
  end
endmodule
```

Siga los pasos anteriores para ver el circuito que la herramienta infiere de su descripción de hardware. Compare el resultado de la descripción de la actividad anterior con la actual. Analice cuidadosamente los resultados y discuta con su grupo y el staff.

Repita el proceso de creación de un testbench y verificación funcional de su circuito utilizando el siguiente código base:

```
`timescale 1ns / 1ps

module test_simple();   // creacion modulo "dummy"

    logic [3:0] BCD_in;     // definicion de conexiones virtuales
    logic       fib;

    fib_rec DUT(             // instancia del modulo a testear
        .BCD_in (BCD_in),
        .fib (fib));

    initial begin            // aca se asignan valores de prueba o estimulos
        BCD_in = 4'b0000;    // 4'b0000 es equivalente a 4'd0
        #3                   // retardo de 3 unidades de tiempo basadas en
    timescale
        BCD_in = 4'b0001;
        #3
        BCD_in = 4'b0011;    // 4'b0011 es equivalente a 4'd3
        #3
        BCD_in = 4'b0111;
    end
endmodule
```

3.9. Multiplexores.

Estudie la sintaxis y el uso de las sentencias `case` en SystemVerilog, y describa un módulo que implemente la funcionalidad del circuito de la Figura 3 mediante asignaciones procedurales. **En su descripción, debe usar exactamente los nombres de señales indicados en el esquemático,**

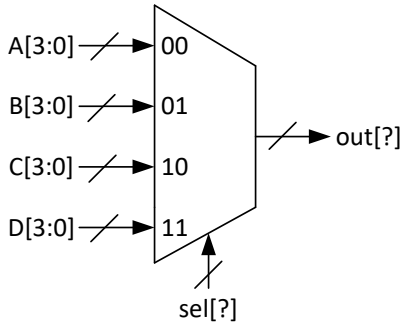

Figura 3: Esquema multiplexor 4:1 con entradas de 4 bits.

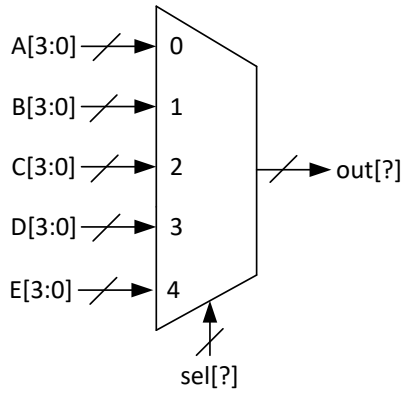

Figura 4: Esquema multiplexor con cinco entradas de 4-bits conectadas a puertos de datos.

incluyendo mayúsculas y minúsculas [5]. Asigne los valores correspondientes a las señales marcadas con "?". Revise el esquemático generado mediante Elaborated Design, y simule el comportamiento del circuito mediante un testbench exhaustivo.

3.10. Más multiplexores.

Repita la actividad anterior para el circuito de la Figura 4. Determine cuidadosamente los tamaños adecuados de las señales, y siga las recomendaciones generales para el uso de las sentencias `case`.

3.11. Descripción de circuito sumador.

En la sesión anterior se le planteó el problema de diseñar un sumador de múltiples bits desde cero, a partir de tablas de verdad. Como (se espera) pudieron concluir, el enfoque clásico de diseño basado en capturas esquemáticas a partir de compuertas lógicas discretas no escala bien al aumentar

[5] Se hace énfasis en que seguir las especificaciones de diseño será parte importante de las evaluaciones. Si su diseño esta funcionalmente correcto (hace lo que se supone debe hacer) pero no sigue las especificaciones (por ejemplo, usa nombres de señales distintos a los solicitados), entonces se considerará incorrecto.

la magnitud de los operandos, y si no se sigue un enfoque modular, el problema se puede volver inmanejable.

Una de las mayores ventajas del uso de HDLs viene de la abstracción que proveen, permitiendo describir sistemas a nivel funcional, y dejando que la herramienta de diseño interprete la descripcion de alto nivel y decida por nosotros la mejor forma de implementarlo, utilizando las primitvas y configuraciones que estime conveniente dadas ciertas restricciones.

Investigue sobre el uso de operadores aritméticos en SystemVerilog, y describa circuitos sumadores de distintos números de bits, y simule su comportamiento para verificar su correcto funcionamiento (se recomienda partir con sumadores simples de 1 o 2 bits). Compare el trabajo asociado al diseño mediante métodos clásicos (como lo hizo en la sesión anterior) y mediante el uso de HDLs.