ELO212: Laboratorio de Sistemas Digitales
Semestre I-2020
Guía de Actividades Sesión 1 (Intro y marcha blanca)

5, 6, 8 de Mayo de 2020

1. Requisitos de entrada

Para las actividades de esta sesión considerar lo siguiente:

- Al menos un integrante de cada grupo debe tener instalada en su computador personal la versión Webpack de Vivado 2019.2. Esta herramienta se usará por el resto del semestre.
- Todos deben haber leído el datasheet de la tarjeta Nexys4DDR o NexysA7 (ver información asociada en Piazza), y tener dominio de representaciones de números en distintas bases, en particular bases binarias y hexadecimal con números fijos de bits.

2. Objetivos

Los objetivos para esta sesión son:

- Verificar la instalación de la herramienta Vivado de Xilinx para diseño de sistemas digitales basados en FPGA.
- Dar un primera aproximación al diseño moderno de sistemas digitales utilizando SystemVerilog.
- Introducir los procesos de síntesis lógica 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), utilizando en específico el 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 de las herramientas disponibles, en esta sesión 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, estos diseños básicos 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, dejando que la herramienta se encargue de elegir el como se implementará esta funcionalidad. **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. 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 cualquier duda que tenga o problema que detecte en el proceso.

3.1. Tarjeta de desarrollo Nexys4 DDR.

El profesor hará un breve introducción al hardware de la tarjeta Nexys4 DDR que se utilizará por el resto del semestre, describiendo algo de nomenclatura, sus capacidades, y periféricos.

Si bien no tendrán acceso físico a la tarjeta hasta que se reinicien las actividades presenciales, todos los diseños estarán orientados a este hardware (es necesario especificar un hardware para trabajar con la herramienta).

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

El profesor dará una introducción sobre el entorno de desarrollo de Vivado y el contexto tecnológico en el que se usa. Luego de la introducción, siga las indicaciones de su profesor y ayudantes 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.

Realice todas las consultas que tengan sobre este simple paso. Una vez creado el proyecto, siga las indicaciones de su profesor para familiarizarse con el entorno de desarrollo.

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:

```
1 module cable (
```

[1] En la práctica esto no es del todo cierto, y existen técnicas avanzadas que permiten reconfigurar partes del circuito *on-the-fly*. Sin embargo, este es un topico avanzado que escapa totalmente de este curso.

```
    input  logic A,
    output logic B
    );

    assign B = A;
endmodule
```

El profesor dará una breve indicación sobre como operan los módulos en SystemVerilog. Posteriormente, siga las indicaciones para sintetizar el diseño y verifique que su código no tiene errores de sintáxis. Interprete y haga un diagrama de alto nivel del circuito que se está describiendo. Siga las indicaciones del profesor/ayudante 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 Ud. infirió.

Agregue al proyecto creado un nuevo archivo de diseño en formato SystemVerilog. Cree un archivo vacío con extensión .sv, y typee 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 del circuito que se está describiendo. Siga las indicaciones del profesor/ayudante 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 Ud. infirió.

3.4. Simulación funcional y diagramas de tiempo.

El profesor dará una indicación general sobre los conceptos y el proceso de simulación funcional (*behavioral simulation*) de un diseño por medio de testbenches. Luego de esta explicación, 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;

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

```

```
17      initial begin                   // aca se asignan valores de prueba o estimulos
18          A = 1'b0;                   // valores iniciales a t=0
19          B = 1'b0;
20          C = 1'b0;
21          #3                          // retardo de 3 unidades de tiempo basadas en
        timescale
22          A = 1'b1;
23          #6
24          B = 1'b1;
25          #4
26          C = 1'b1;
27      end
28  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.

3.5. 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, 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). El trabajo para 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 no posee 1's consecutivos. Para más detalles se recomienda revisar los siguientes enlaces:

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

Diseñe un circuito que permita identificar con un bit de salida en alto si la palabra de entrada (`abcd`) de 4 bits corresponde a un número fibbinario, en caso contrario deberá presentar la salida en bajo.

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

Implemente el circuito considerando que los números fibbinarios que se pueden representar con 4 bits son:

$$0, 1, 2, 4, 5, 8, 9, 10.$$

Desarrollo. Para llegar a una expresión lógica que se pueda implementar directamente con los componentes disponibles, se ha de realizar el siguiente análisis:

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:

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:

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:

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

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

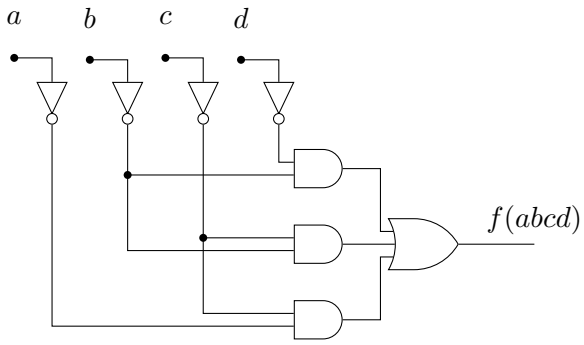

Figura 1: Diagrama Lógico para Números Fibbinarios de 4 bits.

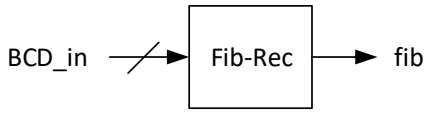

Figura 2: Diagrama de alto nivel de reconocedor de números fibbinarios de 4 bits.

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.

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

Para ilustrar las capacidades de abstracción que otorga el uso de HDLs, en esta actividad describiremos y simularemos el mismo circuito reconocedor de números fibbinarios de 4 bits de la actividad anterior, pero esta vez sacando ventaja de los bloques de alto nivel que entrega SystemVerilog.

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:

```
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. Analice cuidadosamente los resultados y discuta con su profesor y compañeros.

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

```
`timescale 1ns / 1ps

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

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

Figura 3: Diagrama de alto nivel de conversor BCD a 7 segmentos.

```
7
8       fib_rec DUT(                // instancia del modulo a testear
9           .BCD_in (BCD_in),
10          .fib (fib));
11
12      initial begin               // aca se asignan valores de prueba o estimulos
13          BCD_in = 4'b0000;       // 4'b0000 es equivalente a 4'd0
14          #3                      // retardo de 3 unidades de tiempo basadas en
        timescale
15          BCD_in = 4'b0001;
16          #3
17          BCD_in = 4'b0011;       // 4'b0011 es equivalente a 4'd3
18          #3
19          BCD_in = 4'b0111;
20      end
21  endmodule
```

4. Actividades adicionales.

4.1. BCD y display 7-segmentos

Lea el datasheet de la tarjeta de desarrollo para determinar como operar uno de los displays de 7 segmentos. Luego de enteder el funcionamiento, describa la funcionalidad de un circuito que reciba como entrada un número de 4 bits codificado en BCD y muestre en el display su símbolo equivalente en decimal.

El diagrama de entradas y salidas de alto nivel se muestra en la Figura 3, y un snippet de código en SystemVerilog se muestra a continuación:

```
1   module BCD_to_sevenSeg (
2       input  logic [3:0] BCD_in,
3       output logic [6:0] sevenSeg
4       );
5
6       always_comb begin
7           case(BCD_in)
8               4'd0:    sevenSeg = 7'b1111110; // output is abcdefg
9               4'd1:    sevenSeg = 7'b0110000;
10              default: sevenSeg = 7'b0000000;
11          endcase
12      end
13  endmodule
```

Complete el código anterior para que se pueda visualizar en el display los números decimales ingresados en codificación BCD 8421, y siga los pasos para ver lo que infiere la herramienta mediante

el *Elaborated Design*. Elabore un testbench que permita una verificación funcional exhaustiva de su diseño.

5. Comentarios finales

Recuerden que SystemVerilog es un estándar que cumple dos propósitos: **verificación y descripción para síntesis lógica**. En la simulación o testbenches se provee un entorno virtual para instanciar y **simular** la funcionalidad del sistema descrito. El entorno de simulación se ejecuta en el computador y permite usar gran parte del lenguaje. Sin embargo, la descripción de hardware para síntesis lógica es mucho más restrictiva. Se discutirán las diferencias conceptuales entre ambos objetivos en las siguientes cátedras.