Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ingeniería Eléctrica
IEE2753 - Diseño de Circuitos Integrados Digitales

Tarea 2

10 de abril de 2020

Agustín Campeny

1. Preguntas cortas

1. ¿Por qué ejecutamos DRC en el diseño? ¿Cómo se relaciona esto con LVS?

   El DRC es un proceso fundamental al utilizar cualquier software CAD para diseñar un dispositivo que luego debe ser fabricado. Este proceso se encarga de analizar el diseño actual, específicamente dimensiones, separaciones, posiciones, densidades, etc, y compararlas con las especificaciones de un proceso de fabricación. Es un método automatizado que asiste en el diseño, y asegura la factibilidad de este al momento de ser enviado a la fabricación.

   El LVS es una sigla para *Layout Vs Schematic*, y es un chequeo automático para verificar si el layout calza efectivamente con lo que se quiere diseñar basado en la planificación, en este caso el esquemático. Tanto DRC como LVS son herramientas automatizadas que asisten y agilizan en el diseño del layout.

2. Al diseñar el diseño, los NMOS se agrupan y los PMOS se agrupan. ¿Cuál es la razón detrás de esto?

   En la mayoría de los circuitos lógicos CMOS, se busca que la salida de estos sea completamente binaria, es decir o V_{DD} o V_{SS}, y para lograr esto hay que tener ciertas precauciones con el efecto que tiene el cambio en las entradas. Ya que NMOS y CMOS son complementarios en su funcionamiento, es decir que para una misma entrada se genera el efecto contrario en cada uno (si uno se enciende el otro se apaga), el diseño debe procurar que si un camino se abre hacia uno de los rieles, se cierre el camino hacia el riel opuesto, de lo contrario existe la posibilidad de dejar la salida flotando o de generar una conexión entre rieles (cortocircuito).

   Otro motivo es debido al mecanismo de funcionamiento de los componentes. El criterio de encendido o apagado de los transistores MOS depende del voltaje entre gate y source (V_{GS}), y del voltaje entre drain y source (V_{DS}). Un transistor apagado quiere decir que este se encuentra en región de corte, la que se logra cuando $V_{GS} < V_{Th}$, y un transistor encendido es aquel que se encuentra en región de triodo, lo que se logra cuando $V_{GS} > V_{Th}$ y $V_{DS} < V_{GS} - V_{Th}$. Es por este motivo que se busca conectar el terminal source de los NMOS al voltaje más bajo posible (el mas alto posible en PMOS), y de esta forma asegurar un comportamiento predecible con respecto a su voltaje en gate, y evitar que entre en región activa (mucha disipación de potencia).

3. Hemos dicho que enrute cada capa de metal en una sola dirección (horizontal o vertical). Explica por qué es una buena idea en diseños grandes y complejos.

   A medida que los circuito se hacen mas densos y complejos, la interconexión entre celdas se hace más complicada y requiere mucho más trabajo por parte del diseñador, o la herramienta de routing. Para facilitar este proceso una buena práctica es mantener una sola dirección para el ruteo en cada capa de metal, y de esta forma es mas sencillo evitar colisiones entre conexiones, al costo de generar rutas menos óptimas y más largas.

4. Las celdas estándar son puertas que tienen diseños con la misma altura vertical, un riel de tierra que atraviesa la parte inferior de la celda y un riel de alimentación que atraviesa la parte superior de la celda. La ventaja de usar diseños de celdas estándar es que su diseño se puede dividir en filas de celdas estándar y enrutar alimentación / tierra a lo largo del diseño en una cuadrícula. En estos diseños, las filas de celdas alternas se reflejan (voltean verticalmente). ¿Cuál es la ventaja de hacer un diseño como este, en lugar de no reflejar filas alternas?

   La ventaja de invertir verticalmente filas consecutivas es que se puede compartir el mismo riel de alimentación, en lugar de necesitar dos por cada fila, reduciendose de esta manera la altura total del circuito y el área total utilizada.

2. Compuerta misteriosa

1. El circuito lógico de la figura 1 corresponde a un *Full Adder*. El procedimiento utilizado para determinar esto fue primero la realización de una tabla de verdad, que se presenta a continuación:

X	Y	Z	F	G
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Cuadro 1: Tabla de verdad de circuito en la figura 1

   Se puede notar que las entradas X, Y corresponden a los sumandos, Z corresponde al carry in, la salida F corresponde al resultado de la suma y G corresponde al carry out.

2. Se dibujan los diagramas de palo para la puerta lógica **XOR** y **NAND**. Se utiliza el software *xcircuit*, que si bien está hecho para hacer esquemáticos, es también útil para dibujos sencillos. Se utiliza el color azul para representar Metal 1, morado para Metal 2, puntos para contactos entre capas y cuadrados con una equis para los pines de la celda.

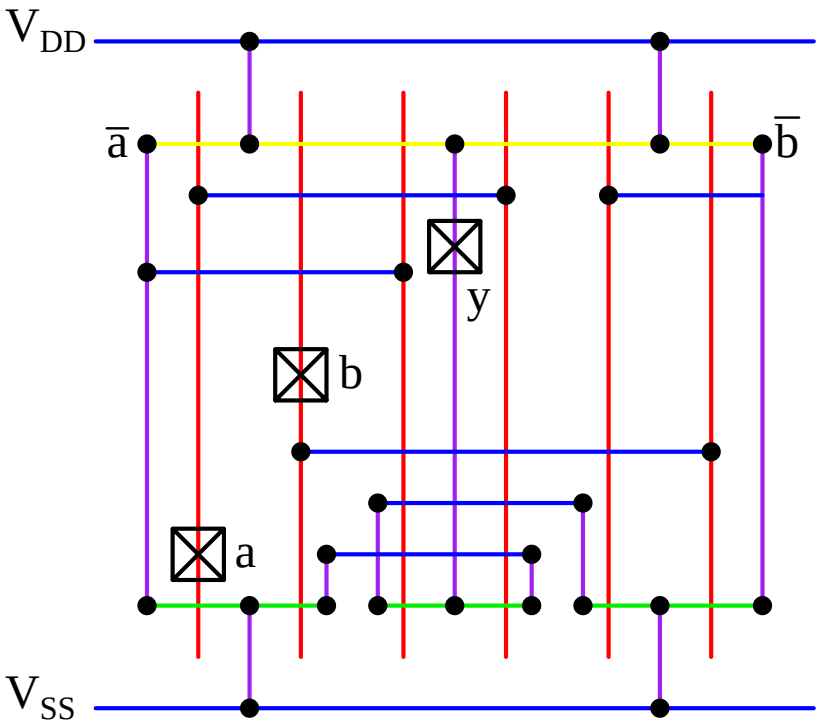

Figura 1: Diagrama de palo para la puerta XOR

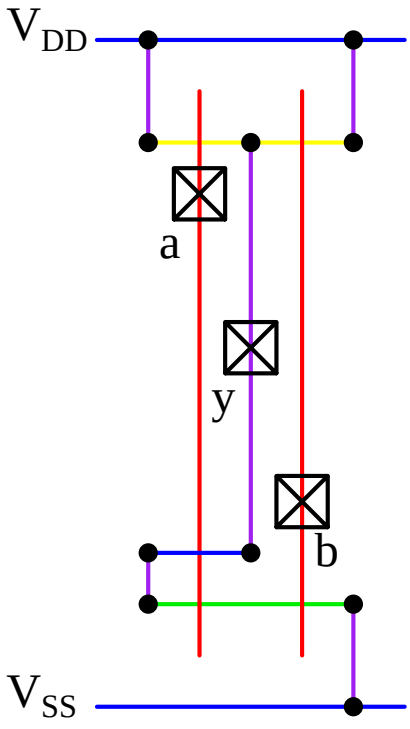

Figura 2: Diagrama de palo para la puerta NAND

Considerando que el alto del sustrato P es de 16λ, y del sustrato N es de 8λ, con los rieles superior e inferior y 6 metales horizontales el alto total de la celda es de 96λ, asumiendo 4λ entre metales y sustratos.

3. Para el ruteo del circuito completo se utiliza Metal 2 (morado) y Metal 3 (naranjo). La celdas son las mismas del item anterior, pero se le remueven la conexiones internas y se disponen de forma horizontal o vertical para el placement de estas.

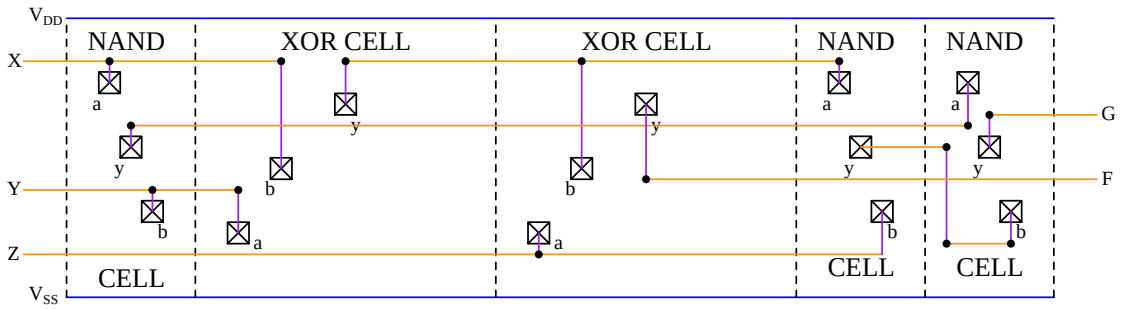

Figura 3: Vista abstracta de circuito completo

4. Para poder darle mérito al sintetizador y que esta no sea trivial, se implementa un código verilog correspondiente al circuito a nivel de comportamiento, es decir, un full adder combinacional. El modulo fue llamado `f_addr` y se encuentra en las carpetas de la tarea junto al verilog sintetizado. A continuación se presenta un bosquejo del resultado de la síntesis.

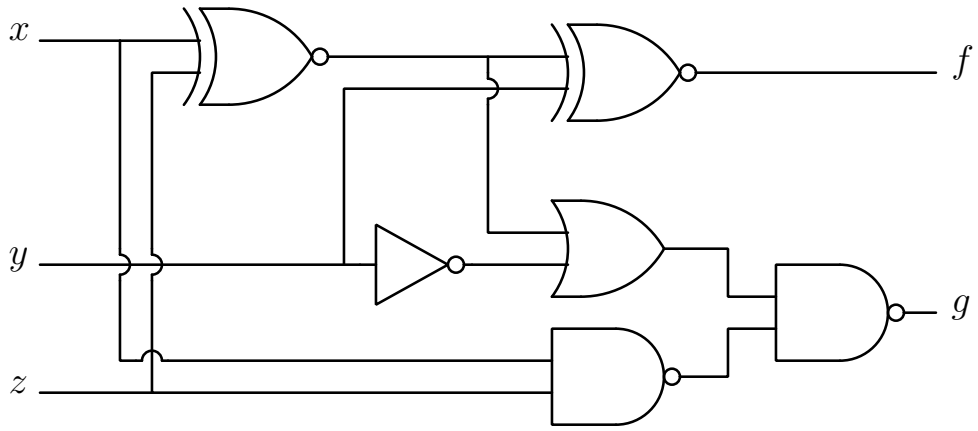

Figura 4: Resultado de la síntesis del circuito

Si bien el circuito resultante no posee la misma implementación que en el enunciado, es un circuito equivalente.

3. Síntesis

Se realiza la síntesis de los módulos rtl creados en la tarea 1 utilizando el software libre *yosys*, y las celdas estándar `osu_05`, provistas por *Oklahoma State University*. Luego de realizar la síntesis, se chequea de manera breve si el resultado es consistente con lo esperado con respecto al módulo rtl, y se comenta al respecto.

Se utilizó el siguiente script para yosys:

```
# read design
read_verilog ../../rtl/lfsr16/lfsr16.v

# generic synthesis
synth -top lfsr16

# mapping to osu05_stdcells.lib
dfflibmap -liberty osu05_stdcells.lib
abc -liberty osu05_stdcells.lib
clean

# write synthesized design
write_verilog synth.v
```

3.1. lbshifter

La síntesis de este módulo se realizó sin problemas, entregando las siguientes estadísticas:

```
=== lbshifter ===

   Number of wires:                 2
   Number of wire bits:            16
   Number of public wires:          2
   Number of public wire bits:     16
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 0
```

Por como está implementado el módulo, que es simplemente un shift circular hacia la izquierda, solo son necesarios wires para la entrada y salida, sin necesidad de compuertas ni registros.

3.2. asr8

La síntesis de este módulo pasó sin problemas, con la siguiente estadística:

```
=== asr8 ===

   Number of wires:                119
   Number of wire bits:            194
   Number of public wires:          13
   Number of public wire bits:      88
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                178
     $_ANDNOT_                      12
     $_AND_                          1
     $_AOI4_                         8
     $_DFF_PN0_                     64
     $_MUX_                          8
     $_NAND_                         3
     $_NOR_                          1
     $_NOT_                         33
     $_OAI4_                        16
     $_ORNOT_                       10
     $_OR_                          22
```

Se puede notar que se generaron 64 flip flops, que corresponden a 8 bits por registro por 8 registros que es el valor de largo por defecto. El resto de las celdas forman parte de la logica para el reset de los registros y la salida con respecto al address en la entrada.

3.3. lfsr16

El proceso de síntesis se realizó sin problemas, dando la siguiente estadística como resultado:

```
=== lfsr16 ===

   Number of wires:                40
   Number of wire bits:            85
   Number of public wires:          6
   Number of public wire bits:     51
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                51
     $_ANDNOT_                     16
     $_DFFSR_PPP_                  16
     $_NOR_                        16
     $_XNOR_                        2
     $_XOR_                         1
```

A simple vista la síntesis pareciera haber dado resultados correctos en cuanto al número y tipo de compuertas, apreciándose 16 registros que conforman el shift register. Al revisar el archivo verilog de salida, se puede notar que las compuertas NAND y NOR son utilizadas para asignar el valor del seed al hacer reset, y las compuertas XOR y XNOR son utilizadas en los taps para realizar el feedback. Si bien se esperarían 3 compuertas XOR para que calce con el diseño, utilizar 2 XNOR y 1 XOR es un circuito equivalente.

3.4. mac8

Al intentar la síntesis de este módulo, apareció el error `ERROR: Multiple edge sensitive events found for this signal!` que impidió la síntesis de este. Investigando un poco[1], se encontró que el siguiente fragmento de código no era sintetizable debido a que el if es gatillado tanto por una señal síncrona como una asíncrona:

```
if (~rst || counter >= 7) begin
  counter <= 0;
  accumulator <= 0;
end
```

La solución fue separar el bloque anterior en un bloque `if-else if`. A continuación se presenta la estadística de la síntesis:

```
=== mac8 ===

   Number of wires:                401
   Number of wire bits:            466
   Number of public wires:           7
   Number of public wire bits:      54
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                432
     $_ANDNOT_                      88
     $_AND_                         66
     $_AOI3_                        39
     $_DFF_PN0_                     20
     $_NAND_                        22
     $_NOR_                         21
     $_NOT_                          2
     $_OAI3_                         5
     $_ORNOT_                        6
     $_OR_                          14
     $_XNOR_                        65
     $_XOR_                         84
```

[1] https://www.reddit.com/r/yosys/comments/5cnvzz/error_with_synthesizing_some_verilog_code/

A simple vista el resultado parece correcto, considerando que hay 20 registros; 16 para el acumulador y 4 para el contador.