Instalación de las herramientas para el curso.

Gabriel Fernando Araya Mora B80525

Escuela de Ingeniería Eléctrica, Universidad de Costa Rica

Circuitos Digitales II (IE-0523)

---

1. Tiempo de ejecución de la tarea:

- **Buscar información**: Siempre antes de empezar cualquier trabajo es importante tomarse un tiempo prudente para buscar información e investigar acerca del tema. Esto es especialmente cierto cuando se trata de programar. Para la búsqueda de información, revisé la documentación de verilog acerca de multiplexores y flip-flops. Revisé el proyecto de verilog del curso anterior para ver si se podía reusar código de alguna u otra forma. Aproximadamente 1 hora me tomó la búsqueda de información.
- **Usando la información**: Como se pide la descripción conductual del módulo solicitado, basta con saber cómo se comporta el multiplexor para poder implementarlo en el código. (Al ser conductual, se espera lógica de if o else). Para implementar el flip-flop se utiliza el reloj y asignaciones no bloqueantes. Desarrollar e implementar esta lógica en verilog me tomó aproximadamente media hora. Mientras que el probador me tomó al menos 2 horas, ya que no entendía muy bien la lógica al escribir el código, y además buscar que fuera igual al propuesto por el profe me tomó más tiempo.
- **Elaboración del reporte**: La elaboración del reporte me tomó aproximadamente una hora.
- **Total:** De lo anterior se saca que el tiempo total para la realización de la tarea es de 4.5h.

2. Descripción arquitectónica o diagrama del circuito.

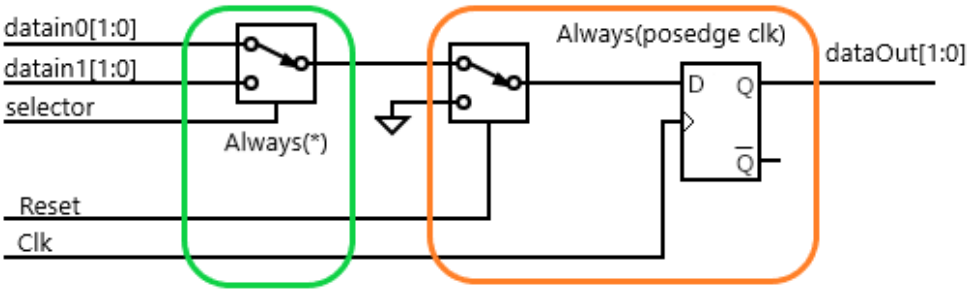

Figura 1: Esquemático de Diseño. (Creación Propia)

Aquí se implementa un único módulo en el cual se incluyen dos instrucciones always, una se encarga de modelar la lógica combinacional en el multiplexor, mientras que el otro always se encarga de modelar la lógica secuencial del flip-flop tipo D con reset. Todo esto se hace de esta manera y usando comandos if/else ya que se pide un modelado conductual.

```
module Mux2x1(
    output reg [1:0]dataOut,
    input [1:0]dataIn0,
    input [1:0]dataIn1,
    input selector,
    input clk,
    input reset);

    reg [1:0]out;

    //Mux de entrada el que escoge entre dataIn1 y dataIn0 a partir del selector
    always @(*)
        begin
            if(selector==0)
                out=dataIn0;
            else
                out=dataIn1;
        end
    // Flop que guarda lo que sea que este ee el cable out que conecta el mux con el flop
    always @(posedge clk)
        begin
            if(reset == 1)
                dataOut <= out;
            else
                dataOut <= 2'b0;
        end
endmodule
```

Figura 2: Verilog del módulo solicitado. (Creación Propia)

3. Plan de pruebas

Para esta parte es necesario idear pruebas de tal forma que se logre comprobar que el circuito diseñado sirva como dicen los requerimientos ante diversas entrada. Para esto se trata de seguir el patrón mostrado por el profesor en el enunciado de la tarea. Primeramente el reset se encontrará en cero con el fin de que el flip-flop devuelva cero. Una vez cambia la señal del reset de bajo a alto se espera que el flip-flop empiece a arrojar los datos anteriores dependiendo del selector en el multiplexor, para probar esto entonces se varía el valor del selector y de los buses de entrada de manera que se cubran varios casos al mismo tiempo. De esto se esperera que cuando el reset esté en alto, y el selector en cero se tenga una salida en el tiempo anterior de lo que había en el bus de datos **dataIn0**, mientras que cuando el reset se encuentra en alo y el selector en alto se da el otro caso donde en la salida se refleja **dataIn1**. Cabe destacar que cuando se da el flanco positivo del reloj se va a ver reflejado lo que había antes en los buses de datos y según indique el selector, esto es ideal ya que así operan los flip-flops.

4. Istrucciones para ejecutar el programa.

Para ejecutar el programa, basta con hacer ***make*** en la consola de linux, con el cuidado de estar en el directorio src de la entrega de la tarea.

```
build: operador1 operador2 operador3 gtkwave1
        echo fin
operador1:
        iverilog BancodePruebasMuxCond.v
operador2:
        ./a.out
operador3:
        rm a.out
gtkwave1:
        gtkwave --dump=tarea02.vcd
```

Después de haber usado el comando bash anteriormente mencionado se abrirá el programa con gtkwave abriendo el archivo .vcd con la simulación de la tarea. Lo único que restaría hacer es agregar las señales al gtkwave.

5. Ejemplos de los Resultados.

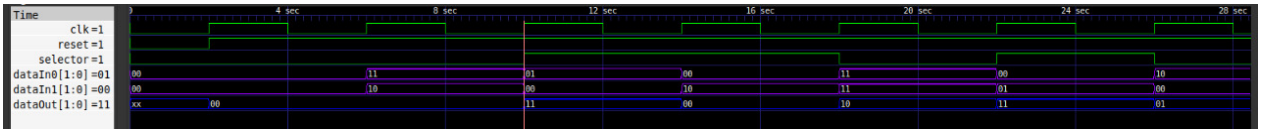

Figura 3: Pruebas ejecutadas para la tarea. (Creación Propia)

En la figura anterior se ve cómo la señal es prácticamente igual a la mostrada por el profesor en la guía para la tarea. Repliqué esta prueba ya que me parece que maneja todos los posibles escenarios a los cuales se podría ver sometido el circuito.

En la figura también se muestra como en cada flanco positivo de reloj la salida cambia dependiendo del selector del multiplexor, y el efecto de retardo en la señal que genera el flip-flop. Adentrándose un poco más

en la simulación se ve que cuando el reset está en alto y se da el segundo flanco positivo del reloj la salida será cero independientemente del selector ya que ambos buses de datos comienzan en 00 y en el flip-flop se refleja el estado anterior. Por el otro lado cuando se da el tercer flanco positivo del reloj el selector se encuentra en alto; sin embargo la salida después del tercer flanco está en 11, esto es correcto debido a que el flip-flop almacena el estado anterior y para el estado anterior el selector se encontraba en cero y por ende en la salida se ve reflejado lo que haya en el bus de datos de la entrada 0. Este comportamiento se repite hasta que se acaba la simulación.

6. Análisis y conclusiones:

De la sección anterior se puede concluir que la tarea se terminó con éxito, debido a que el circuito se comporta de manera esperada y además al seguir el set de pruebas del profesor se obtiene la misma salida. Para evitar errores a la hora de instanciar los módulos se usó la funcionalidad explorada en la tarea anterior para hacer esto de forma automática ($/ * AUTOINST * /$). Lo más complicado de esta tarea fue sin duda alguna el probador, ya que llegar a la respuesta obtenida por el profesor requirió de mucho trabajo y además me costó entender la lógica detras de asignar las entradas para que el módulo las reconociera.