UNIVERSIDAD DE COSTA RICA
Escuela de Ingeniería Eléctrica
IE0523 – Circuitos Digitales II

Tarea 2

Yeison Rodríguez Pacheco, B56074

28/03/19

Resumen

En esta tarea se realiza el diseño conductual y simulación de un demultiplexor de 1:2 con la especialidad de que la salida que no se esté atendiento en ese momento debe almacenar el dato que tuvo anteriormente. Además el demultiplexor debe funcionar instantáneamente cuando la señal de reset se pone en alto.

1. Contabilización del tiempo

Tabla 1: Contabilización del tiempo

Sesiones	Sesión 1	Sesión 2	Sesión 3
Búsqueda y estudio de información	20 min	10 min	0 min
Diseño inicial del circuito	25 min	50 min	0 min
Programación del código	2h	2.5h	45min
Ejecución y pruebas	0min	20 min	35 min
Reporte	0 min	0 min	4h

2. Descripción arquitectónica del circuito

Se debe diseñar un demultiplexor 1:2 con capacidad para buses de 4 datos,y que además guarde el valor que tuvo anteriormente la salida que no se está utilizando (ya que en ese momento se estaría utilizando la otra salida). Para lograr que el circuito cumpla con lo especificado se sabe que se necesitarán Flip Flop, esto se deduce porque es necesario guardar valores anteriores, y esto se logra con este tipo de etructuras, además de que se debe generar una señal selector, la cuál está sincronizada con el reloj, por lo que también se debe utilizar un FF en este segmento de la estructura. En la figura 1 vemos un diagrama estructural del circuito diseñado para a continuación explicar el diseño.

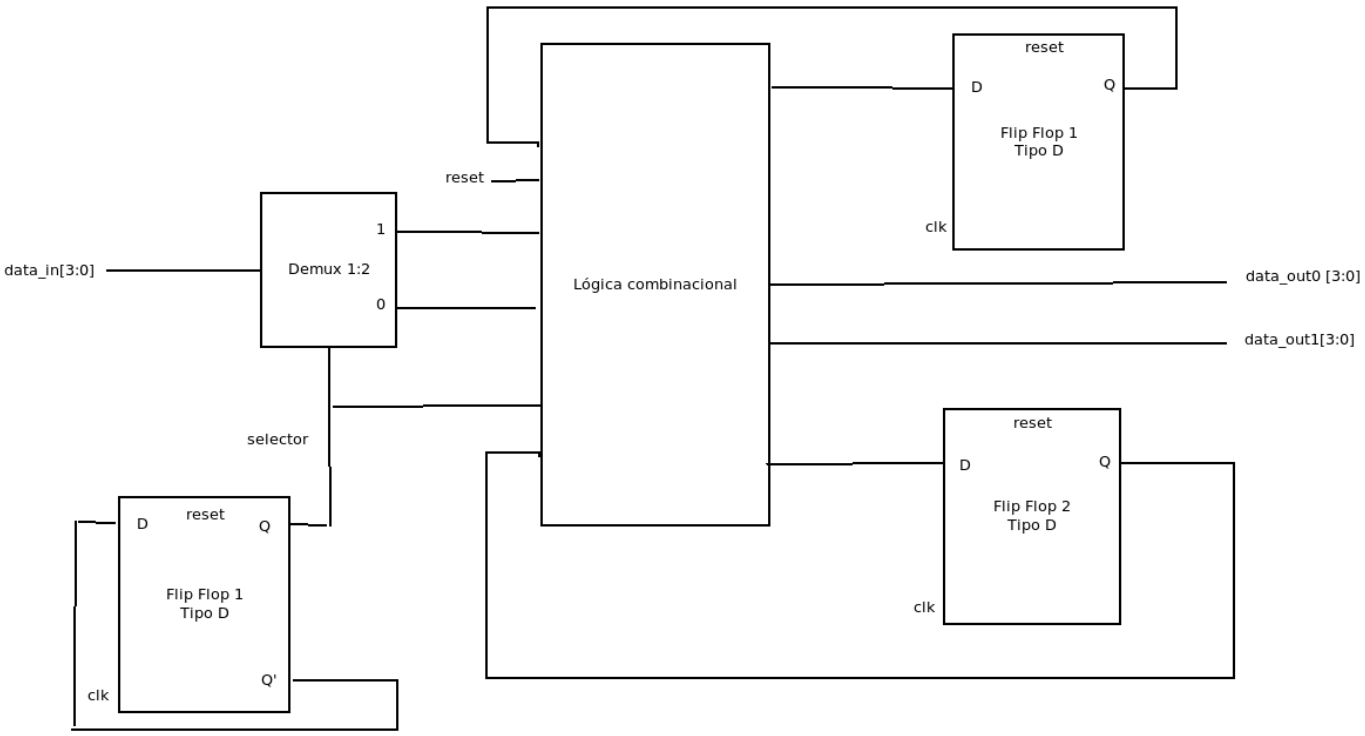

Figura 1: Diagrama arquitectónico del circuito diseñado. Hecho en Dia

Vemos que los datos ingresan por data_in[3:0] que es un bus de datos de 4 bits, estos datos son seleccionados mediante la señal selector para decidir si pasan a la salida 1 o 0 del demux. Seguidamente

entran a una lógica combinacional junto con las señales reset, selector y las salidas de los FF. Dicha lógica combinacional se encarga de generar los datos que llegarán a las entradas de los FFs y además de generar las salidas data_out0[3:0] y data_out1[3:0]. Notamos que las salidas son de lógica combinacional, esto ocurre porque el reset está sincronizado con el reloj y justo cuando la señal reset se levanta, la salida debe cambiar inmediatamente.

La lógica combinacional también se encarga de que lleguen los datos correspondientes a los FF para que los 'almacenen' y así utilizarlos cuando sea necesario. Dicha lógica también funciona como un selector para saber cuales datos reflejar en cada una de sus salidas, esto dependiendo del valor de la señal selector, ya que esta es la que regula si se debe mantener en la salida lo guardado anteriormente por los FF o cambiar a la generada por el Demux.

La señal selector es generada con ayuda de un flip flop ya que esta debe oscilar entre 0 y 1 un ciclo de reloj después de que el reset se ponga en alto. Por lo que cuando reset se activa, selector es 0 y el FF se realimenta con su salida negada, por lo que en el siguiente ciclo select será 1 y esto se repite mientras reset esté activo.

3. Plan de pruebas

Para el diseño realizado se procedió a realizar pruebas mediante el archivo banco_pruebas_conductual.v , el cuál es el que realiza la lógca para que el archivo de probador y demux_conductual se comporten correctamente.

Se realizaron tres archivos de prueba (probador.v, probador1.v, probador2.v), el primero de estos corresponde a la simulación que se brindó en el enunciado de la tarea, en la sección de resultados veremos que la salida es exactamente igual a la esperada. El archivo probador1.v se le realizó un cambio para que el reset se pusiera en estado bajo cuando selector estuviera en alto (a diferencia de la prueba inicial). Por último en probador2.v se cambian todas las entradas, y se hace encender y apagar repetidas veces el reset, para observar comportamientos en distintos escenarios.

Se realizaron múltiples pruebas al Makefile para verificar que funcionara correctamente.

4. Instrucciones de utilización de la simulación

Para ejecutar las simulaciones, abra su terminal de linux y diríjase a la carpeta llamada src, en esta se encuentra un Makefile, el cuál es utilizado para compilar, ejecutar y simular todas las pruebas. En la tabla 2 se presentan todos los comandos del Makefile. A continuación se muestra un ejemplo de comandos para la correcta ejecución.

```
make prueba0
make gtk
```

Para ejecutar las demás pruebas se utiliza el mismo procedimiento. En caso de querer abrir el pdf del informe, puede utilizar el siguiente comando, o ir a la carpeta informe.

```
make pdf
```

Tabla 2: Comandos de Makefile

Comandos de Makefile	
make	Compila y ejecuta la prueba0, la cuál es la que se dió en el enunciado original.
make gtk	Abre el GTKwave con los datos de la simulación compilada actualmente
make prueba0	Compila y ejecuta la prueba0, la cuál es la que se dió en el enunciado original.
make prueba1	Compila y ejecuta la prueba1.
make prueba2	Compila y ejecuta la prueba2.
make clean	Borra los archivos binarios y .vcd
make pdf	Abre el pdf del informe.

A continuación ejemplos para ejecutar las otras pruebas:

```
make prueba1
make gtk

make prueba2
make gtk
```

A la hora de ejecutar las simulaciones en GTKwave se deben utilizar las generadas por demux_conduc, ya que estas presentan las señales internas mientras que el probador no las contiene.

5. Ejemplos de los resultados

Vemos en la figura 2 el resultado de la simulación de la prueba 0, la cuál muestra un comportamiento igual a la brindada en el enunciado de esta tarea, por lo que se corrobora el correcto funcionamiento. Principalmente podemos observar que justo cuando reset presenta su estado alto, inmediatamente se refleja la entrada a la salida correspondiente (dependiendo del valor de selector) el resultado ingresado en data_in, también es importante notar que las salidas siempre almacenan el dato anterior, excepto si reset es igual a 0, en este caso inmediatamente las salidas se convierten en 0.

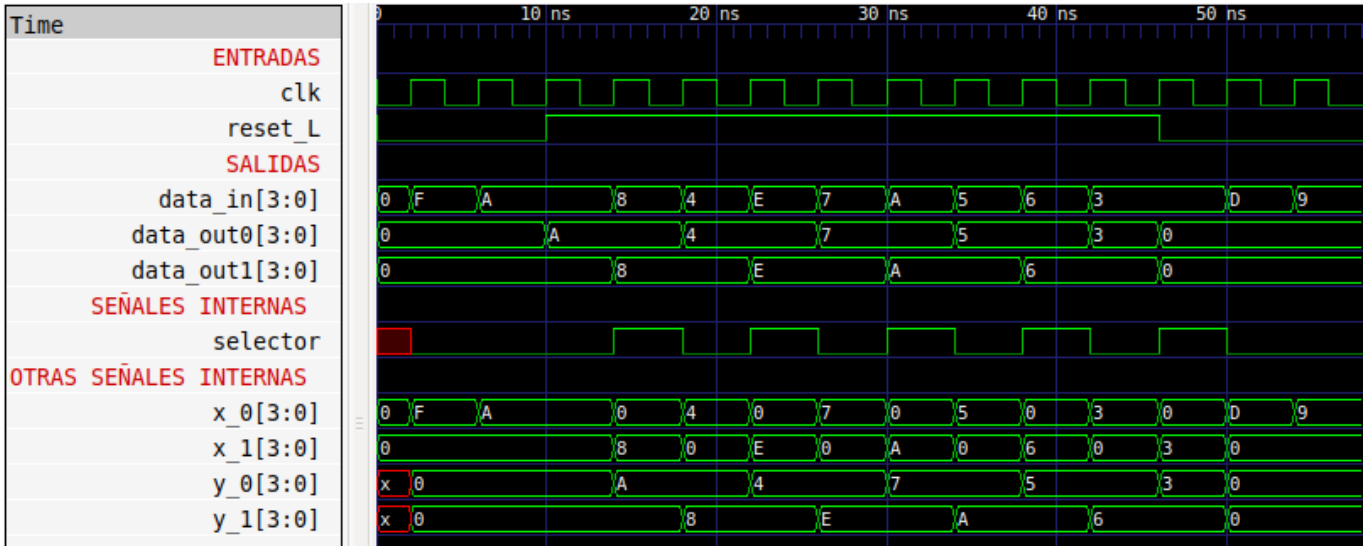

Figura 2: Gráficas de las señales de la prueba 0. Hecho en GTKwave

Para la simulación de la prueba 1 (figura 3) vemos que al levantarse reset se refleja la entrada a data_out0 y se mantiene durante el siguiente ciclo mientras data_out1 refleja la entrada actual, seguidamente reset cae y todas las salidas vuelven a ser 0.

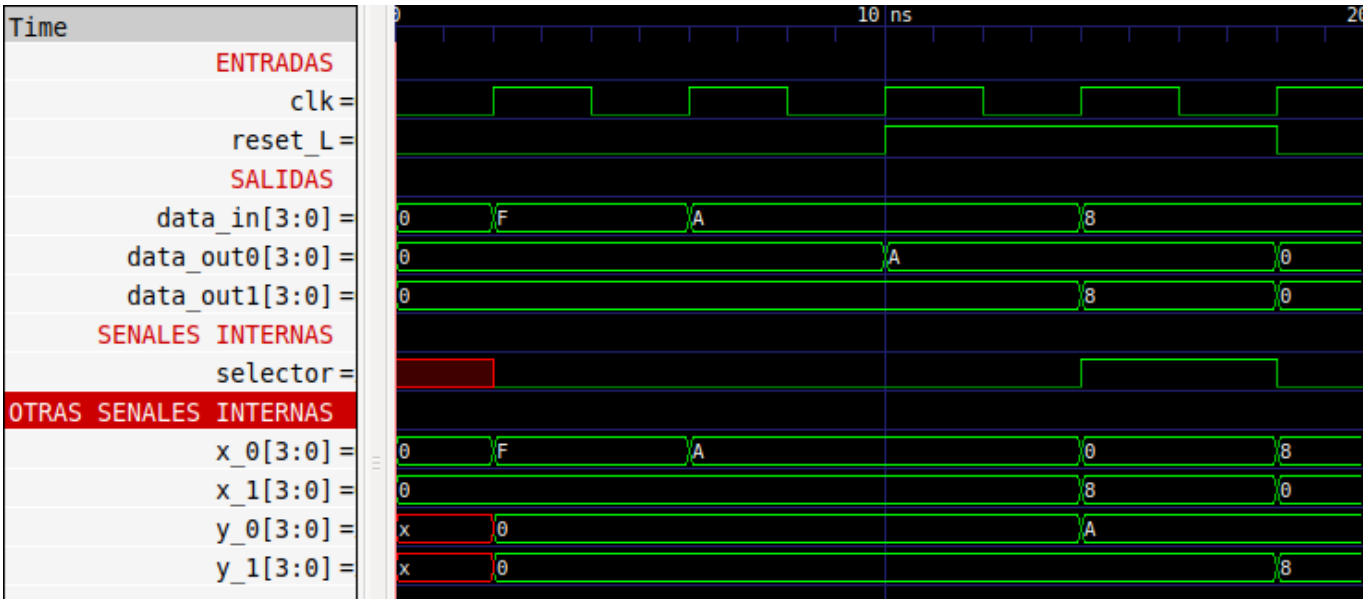

Figura 3: Gráficas de las señales de la prueba 1. Hecho en GTKwave

Para la simulación de la prueba 2 (figura 4) vemos que se cambiaron todas las entradas y se hizo variar irregularmente el reset. Observamos que siempre que reset sea 0, todas las salidas son 0, y que siempre las salidas almacenan su estado anterior, mientras el selector atiende la otra salida. El comportamiento del circuito es el esperado aún con tantas variaciones en reset.

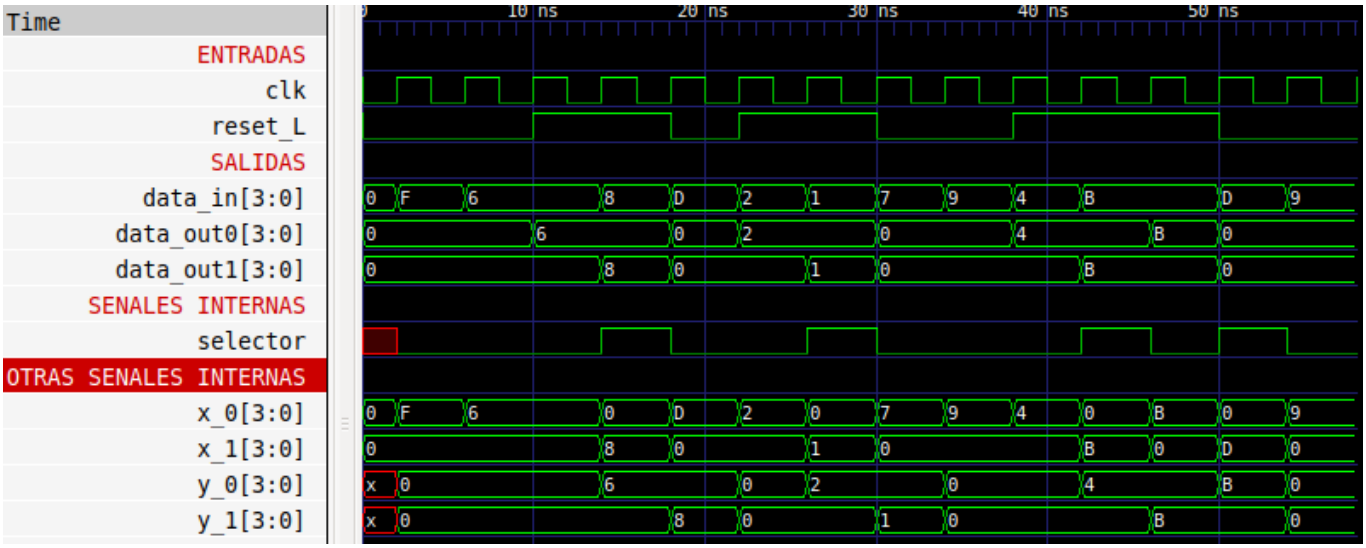

Figura 4: Gráficas de las señales de la prueba 2. Hecho en GTKwave

6. Análisis y conclusiones

6.1. Análisis de resultados

6.1.1. Prueba 0

Al ejecutar la prueba 0 (prueba igual a la del enunciado de la tarea) y simularlas en GTKwave observamos la salida de la figura 2. Notamos que durante toda la simulación los datos en la entrada están cambiando, pero no es hasta que la señal reset se pone en 1 que inmediatamente la salida data_out0 refleja la entrada data_in , como era de esperarse ya que la señal selector está en 0, al selector volverse 1 en el siguiente ciclo de reloj, vemos que ahora la salida data_out1 es la que refleja la entrada de data_in, mientras que data_out0 continúa con el valor que tuvo en su ciclo anterior. Nuevamente ahora selector se pone en 0 y se repite el comportamiento. Vemos que al caer la señal reset a 0, todas las salidas se vuelven 0, sin importar el valor de selector.

Las gráficas de x_0 y x_1 corresponden a señales internas de lógica combinacional. Las señales y_0, y_1 son también señales internas pero son las salidas de los FF, por esto es que cambian su valor hasta 1 ciclo después de que el reset se active. Ambos pares de señales son utilizados para calcular las salidas.

6.1.2. Prueba 1

Al ejecutar la prueba 1 (figura 3) observamos nuevamente que al levantarse reset, inmediatamente se refleja la salida en data_out0, para en el siguiente ciclo seguir manteniendo este valor mientras que data_out1 refleja el nuevo valor de la salida, pero no lo mantiene en el siguiente ciclo ya que la señal reset cae a 0 y todos los valores de la salida se vuelven 0.

6.1.3. Prueba 2

Por último en la prueba 2 (figura 4) se cambian todas las entradas y se hace cambiar varias veces la señal reset de 0 a 1, para buscar por posibles casos esquina. Vemos que al activarse reset se despliegan los

valores en data_out0, el cuál almacena este valor durante el siguiente ciclo mientras data_out1 toma el valor de la entrada.Al reset volver a 0 inmediatamente todas las salidas caen a 0, pero cuando reset vuelve a estado alto, ocurre el mismo comportamiento visto anteriormente, por lo que se tiene consistencia. Al subir reset por última vez a estado alto, data_out0 y data_out1 realizan su comportamiento esperado, reflejando la salida y almacenándola mientras el selector trabaja con la otra salida. y nuevamente caen a 0 instantaneamente con la caída de reset.

6.2. Inconvenientes

Al realizar este diseño se tuvieron algunos inconvenientes, los cuales fueron:

- A la hora de realizar el primer diseño, no se contempló que la salida no podía ser directamente la del FF, ya que el resultado era que las señales de las salidas estaban corridas un ciclo de reloj más adelante, por lo que no se cumplía con lo especificado. Posteriormente este problema fue resuelto con un diseño nuevo.
- Dificultad para manejar la sintaxis de verilog, ya que nunca se habían utilizado buses de datos, en un principio la descripción conductual quedó muy extensa, pero conforme se investigó más se pudo simplificar todo para hacerlo más legible y de igual forma funcional.

6.3. Conclusiones

- Se logró el comportamiento esperado del circuito y se pudo corroborar con el ejemplo brindado en el enunciado de esta tarea.
- La robustez en el diseño se pudo verificar ya que a pesar de las pruebas variando las señales, la descripción conductual siempre respondió como era esperado.
- Fue posible utilizar la herramienta Autoinst para hacer la instanciación correcta en el banco de pruebas.
- Se realizó un Makefile que pudiera automatizar varias pruebas con la ayuda de sed, para facilitar el uso del diseño.
- A pesar de que se tuvo varios inconvenientes, fue posible aprender de ellos para no cometer los mismos errores en futuros proyectos.