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

Tarea 6

Yeison Rodríguez Pacheco, B56074

08/05/19

Resumen

En esta tarea se realizarán tres circuitos. El primero es el demux de las tareas pasadas pero con bits de válido para su entrada y salida, seguidamente se crea el mismo demux pero que funciona a media frecuencia y por último se realizará un demux 1:4 a partir de los creados anteriormente.

1. Contabilización del tiempo

Tabla 1: Contabilización del tiempo

Sesiones	Sesión 1	Sesión 2	Sesión 3
Búsqueda y estudio de información	15 min	10 min	0 min
Mejora del Makefile	35min h	0 min	0 min
Programación del código	3h	2h	1h
Ejecución y pruebas	0min	35 min	40 min
Reporte	0 min	0 min	1.3h

2. Descripción arquitectónica del circuito

En la figura 1 se presenta el diagrama del circuito de la tarea 6, que consiste en un demultiplexor de 1:4 de 4 bits.

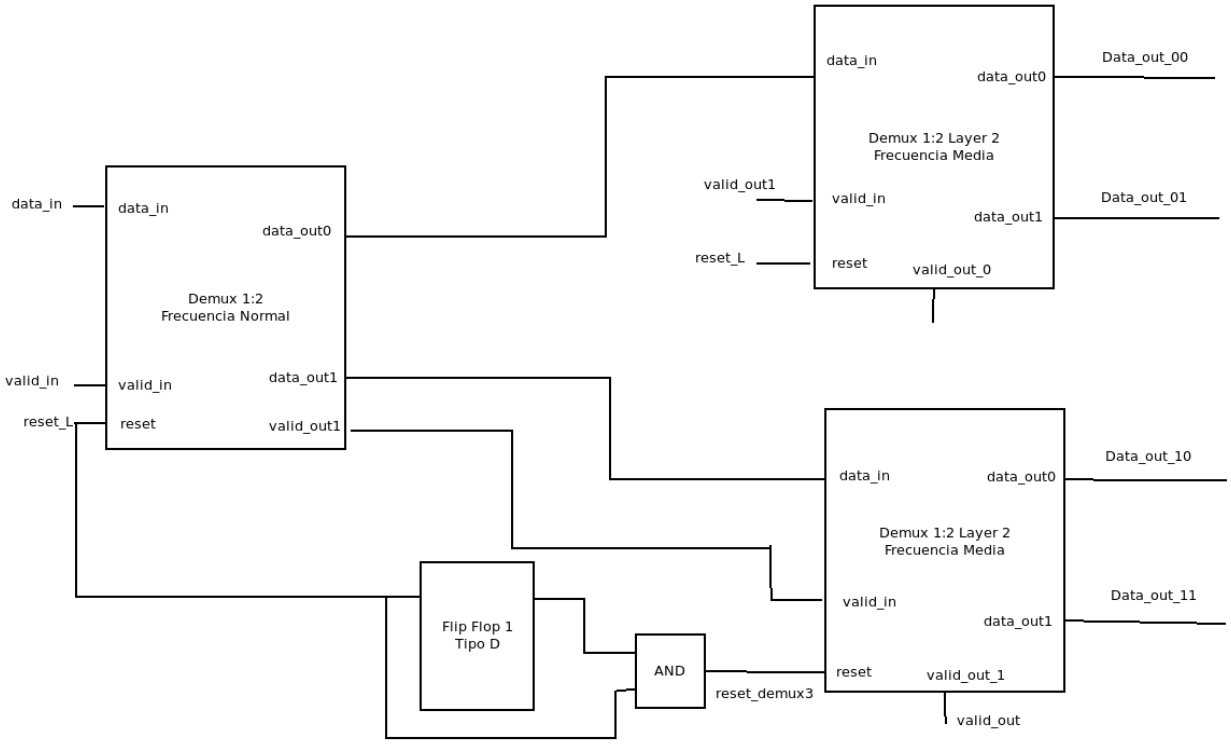

Figura 1: Diagrama arquitectónico del demux 1:4. Hecho en Dia

Leyendo el diagrama de izquierda a derecha, podemos ver que el primer demultiplexor es el mismo de las tareas anteriores a excepción de que se agregó la entrada de valid_in y la salida valid_out, las cuales son utilizadas para saber si el dato de entrada o salida es válido, en caso de no ser válido se mantiene el dato anterior. Las salidas del demux se conectan a la entrada de otros dos demux que funcionan a media frecuencia (su señal interna selector cambia cada dos ciclos de reloj).

Vemos que existe un FF y una compuerta en la señal de reset del tercer demux, esto ocurre ya que este tercer demux debe comenzar a operar un ciclo después que los otros para que se propaguen los datos de manera correcta.

La señal de valido de la salida del primer demux se conecta a la señal de valido de la entrada de los segundos.

3. Plan de pruebas

Las pruebas consisten en probar primeramente el demultiplexor 1:2 de frecuencia normal con las señales de válido tanto para entrada como para salida, con el fin de ver que su comportamiento es el correcto. Este demultiplexor tiene el nombre de demux_layer1.

La segunda prueba consiste en comprobar el correcto funcionamiento de un demultiplexor que funcione a media frecuencia, por lo que debe sostener los datos a su salida por cuatro ciclos de reloj y su señal selector interna hace toggle cada 2 ciclos de reloj. Este demultiplexor también contiene las señales de válido para entrada y salidas. Tiene el nombre de demux_layer2.

La tercer prueba consiste en comprobar que el demultiplexor 1:4 realizado con un demux_layer1 y dos demux_layer2 funcione de manera correcta. Este demultiplexor es llamado demux_layer3.

Cada prueba consiste en una simulación en GTKwave en la cuál se compara tanto el modelo conductual como el estructural. También se cuenta con un comparador en el probador para saber si las salidas conductual y estructural son iguales.

4. Instrucciones de utilización de la simulación

En esta tarea todos los demultiplexores se prueban en el mismo banco de pruebas, por lo que para su ejecución solo es necesario realizar el siguiente comando:

```
make
```

Inmediatamente se presentarán las simulaciones en GTKwave, en la imagen 2 se deja un ejemplo de cómo se verían de manera ordenada, ya que el simulador ordena automáticamente las señales por orden alfabético. Vemos que en primer lugar las señales del demux_layer1 terminan en l1, lo mismo con ocurre con los demás demux. Las señales estructurales se presentan en azul y pueden ser identificadas por la E en su nombre, lo mismo para las conductuales pero con una C.

En caso de que se quiera ver las señales internas de cada uno de los demux_layer se recomienda abrir las señales simuladas de cada una de las instancias creadas en el banco de pruebas. Se encuentran con el nombre demux_layer1_cond_instance o demux_layer1_estr_instance, etc.

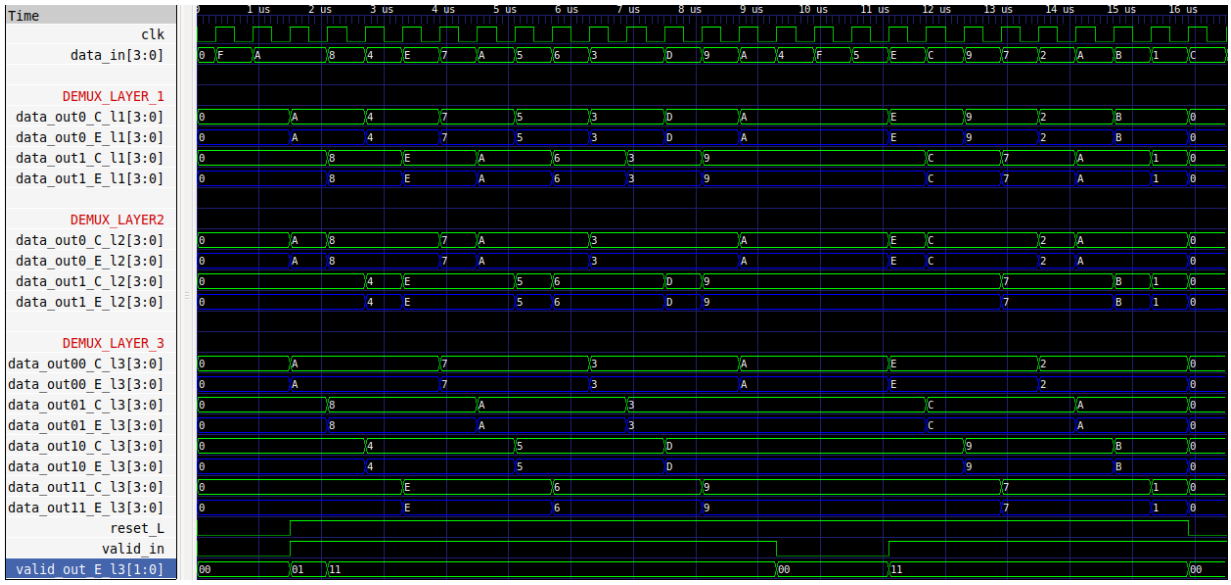

Figura 2: Respuesta de los demultiplexores.

En la tabla 2 se dejará una lista detallada de comandos del makefile, muchos de estos solo se programaron para la creación del laboratorio, y no para las pruebas en sí.

Tabla 2: Comandos Makefile

Comandos del Makefile para pruebas (usuario)	
make	Compila y ejecuta la prueba estandar, también abre el GTKwave con las simulaciones
make gtk	Ejecuta nuevamente el gtkWave, por si necesita abrirlo de nuevo sin compilar
make clean	Remueve todos los archivos .o y .vcd
make pdf	Abre el pdf del informe
Comandos de Makefile para facilitar generación de código (diseñador)	
make generar_archivos	Crea los archivos conductual, probador y banco de prueba según variables de los nombres deseados especificadas en el Makefile (Comando parametrizado)
make generar_archivo_yosys	Genera el archivo con los comandos de Yosys con respecto a los nombres elegidos por el diseñador en las variables del Makefile (Comando parametrizado)
make llenarBancoPrueba	Este comando hace los include e instancias necesarias para que el banco de pruebas funcione correctamente. (Comando parametrizado)
make ejecutar_yosys	Este comando ejecuta el archivo .ys y además se encarga de cambiar el nombre del módulo generado por la síntesis, y hace automaticamente el include del archivo que contiene la librería de compuertas a utilizar (Comando parametrizado)

5. Discusión y Análisis de resultados

5.1. Demultiplexor frecuencia normal, bits de válido (demux_layer1)

En la figura 3 vemos la respuesta estructural y conductual del demultiplexor de las tareas pasadas con señales de válido para la entrada y la salida.

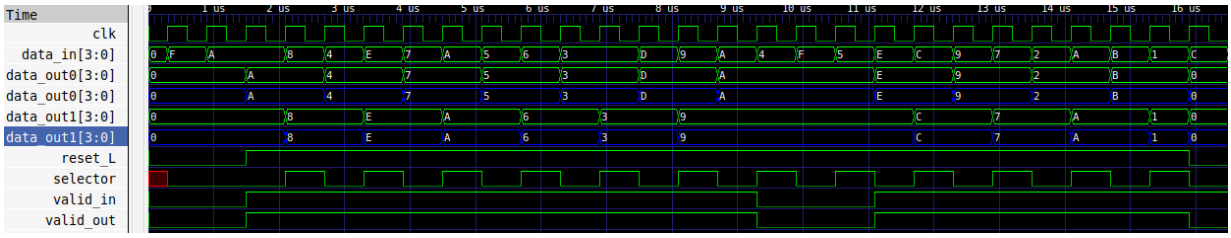

Figura 3: Respuesta conductual y estructural del demultiplexor de frecuencia normal demux_layer1. Hecho en GTKwave

Notamos que al activar la señal de reset inmediatamente se propaga la entrada en la salida 0, ya que la señal selector tiene este valor, además de que se pone en alto la señal de valid out, ya que la salida es válida. Cuando selector es 1 se se propaga el valor de la entrada a la salida 1 mientras la salida 0 mantiene su estado anterior. Hasta este punto el funcionamiento es el ya programado con anterioridad. Pero observamos que al caer la entrada valid_in a 0, las salidas comienzan a tener un único valor, que es el que tenía la entrada la última vez que fue válida. Vemos que cuando valid_in sube a 1, inmediatamente se propaga a la salida correspondiente el dato de la entrada.

5.2. Demultiplexor frecuencia media (demux_layer2)

En la figura 4 vemos el comportamiento del demultiplexor de frecuencia media con señales de válido para su entrada y salidas.

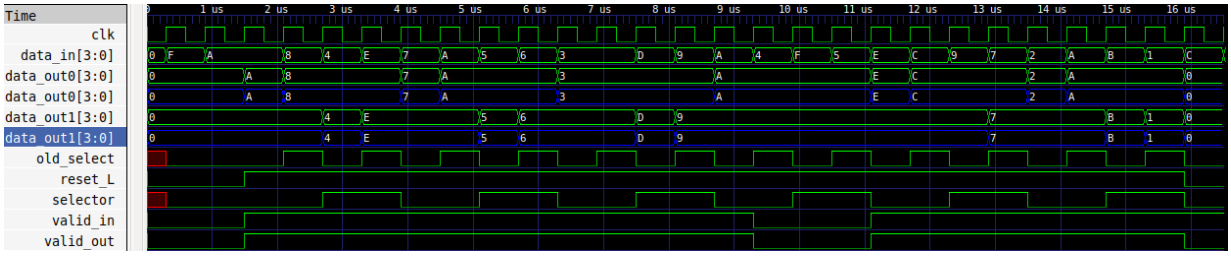

Figura 4: Respuesta conductual y estructural del demultiplexor de frecuencia media demux_layer2. Hecho en GTKwave

Lo primero que debemos notar es que la señal de selector ahora dura dos ciclos de reloj en hacer toogle, por lo que los dos primeros ciclos desde que reset sube a 1 la salida 0 propaga la entrada, cuando la señal de selector cambia a 1 la salida 0 sostiene por los siguientes dos ciclos el último dato que tuvo mientras que la salida 1 propaga los datos de la entrada por los siguientes dos ciclos de reloj. La respuesta de las salidas puede parecer confusa pero es como se espera que trabaje el circuito.

Vemos que las señal de valid in al igual que en la figura anterior hacen que la entrada mantenga su último estado válido hasta que esta señal vuelva a 1, y la señal valid out cuando cae a 0 las salidas mantienen su último valor válido hasta que esta señal vuelva a 1.

5.3. Demultiplexor 1:4 hecho con los demultiplexores anteriores (demux_layer3)

En la figura 5 vemos la respuesta del demultiplexor 1:4 hecho con los demultiplexores anteriores.

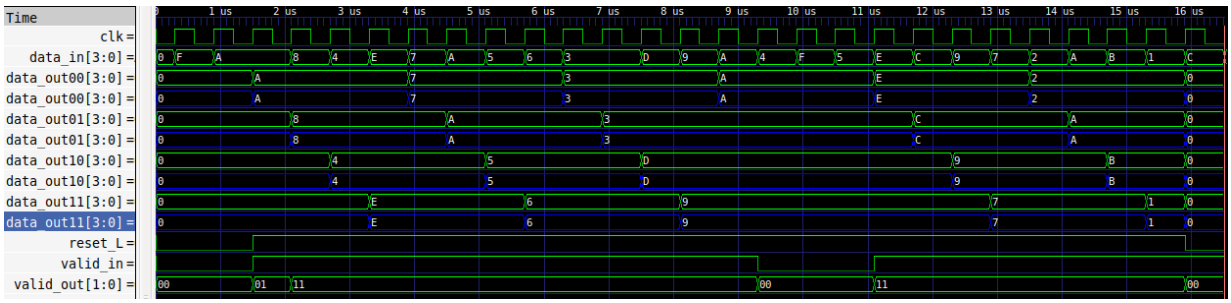

Figura 5: Respuesta conductual y estructural del demultiplexor 1:4 demux_layer3. Hecho en GTKwave

Notamos que ahora se presentan cuatro salidas para el circuito, y cada una de estas salidas mantiene un dato por cuatro ciclos de reloj, mientras las demás salidas se encuentran cambiando. Vemos que el comportamiento es el esperado ya que todos los datos de la entrada se propagan en orden por las diferentes salidas y se sostienen por el tiempo necesario.

Notamos que al caer la señal valid in a cero las señal de valid out también cae para ambas salidas, y se comienzan a sostener los valores que tuvo el circuito anteriormente. Vemos que al ser válido nuevamente la señal valid_in las señales de out también lo son y se propaga inmediatamente la entrada a la salida correspondiente.

Se puede notar que al levantar la señal de reset en el comienzo de la simulación, solo una de las salidas es válida, esto ocurre porque el reset de uno de los demux de media frecuencia llega hasta un ciclo de reloj después, esto se hizo así para que el circuito funcione correctamente, pero también es correcto asumir que la salida del segundo demux es válida hasta el segundo ciclo de reloj, ya que hasta ese momento propaga su primer dato.

6. Conclusiones

- Fue posible implementar bits de válido tanto para la entrada como para las salidas, y mantener el último dato válido en lugar de cambiar a cero los valores, esto con el fin de gastar menos potencia.
- Se pudo comprobrar que es posible crear un demultiplexor que funcione a frecuencia media con el mismo comportamiento del visto en tareas anteriores.
- Se logró utilizar dos módulos de distintos demultiplexores para realizar un circuito más completo a partir de conexiones entre los mismos (demux 1:4).
- Es importante tener un buen Makefile para automatizar tareas.