Múltiples Multiplexores con Bit de Valid

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 yosys acerca de la síntesis y cuales son las mejores prácticas para obtener las descripciones estructurales para los diseños solicitados y buscar si era posible reutilizar algún código de las tareas anteriores. Aproximadamente hora me tomó la búsqueda de información.
- **Usando la información**: El problema de esta tarea no es diseñar nada ya que se cuenta con el módulo conductual hecho para la tarea #2; sin embargo, entender la implementación del Valid fue algo bastante complicado. Este proceso de acomodo del código y la obtención de la síntesis hecha por yosys me tomó aproximadamente 2 horas.
- **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 4h.

2. Descripción arquitectónica o diagrama del circuito.

2.1. Síntesis Estructural:

Para esta tarea se tienen varios multiplexores de diferentes tipos los cuales tienen como base el mux logrado para la tarea #2, pero ahora con un dispositivo extra, el cual se encarga de validar los datos de entrada y salida.

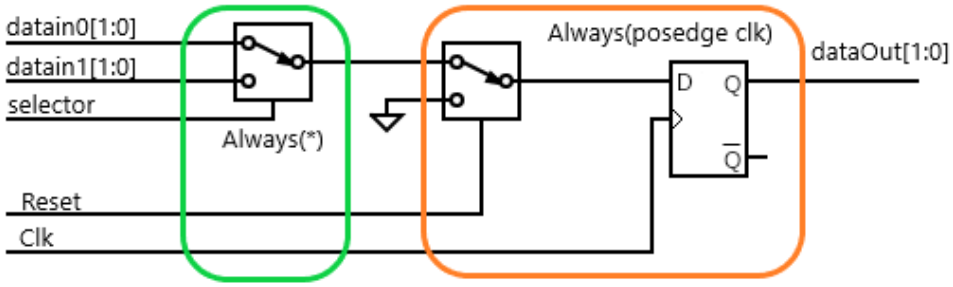

Figura 1: Esquemático de Diseño para la tarea #2 ahora de forma conductual. (Creación Propia)

Para la construcción del mux 2:1 de cuatro bits se colocan dos multiplexores base 2:1 de 2 bits en paralelo para que su salida forme una de 4 bits.

Para la construcción del mux 4:1 de cuatro bits se colocan 3 muxes 2:1 de 4 bits contruidos en el punto B, se colocan 2 en paralelo y uno en serie de forma que un selector escoge datos para los muxes en paralelo y el otro selector escoge los datos para la salida.

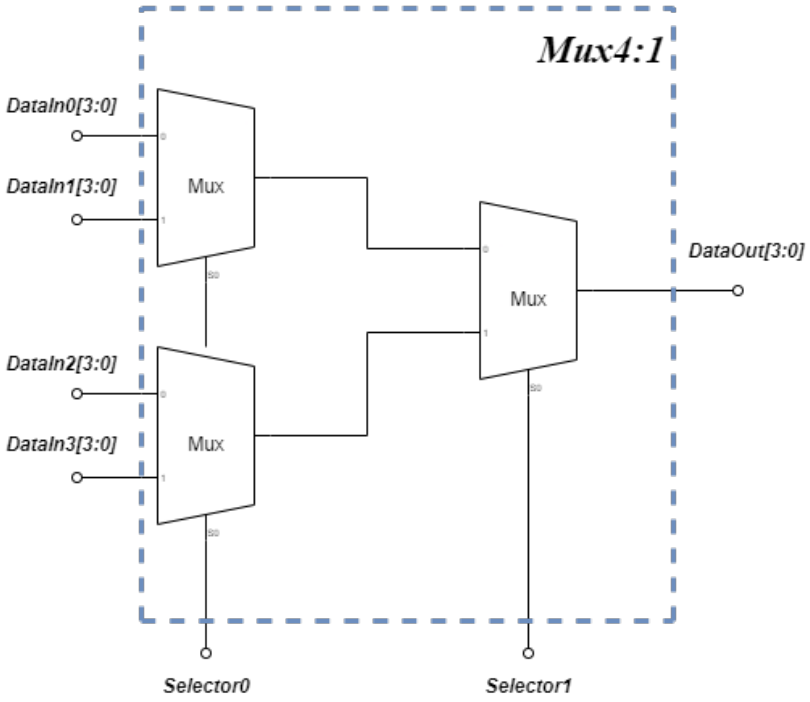

Figura 2: Esquemático de Diseño para el mux 4:1. (Creación Propia)

Todos las síntesis fueron hechas utilizando yosys, junto con la librería de arquitectura CMOS.

3. Plan de pruebas

Todos los cambios en las señales de entrada se hicieron con base en el archivo mostrado en la tarea #2 dada por el profesor, esto debido a que hace muchos casos posibles y combinaciones de los mismos, lo cual asegura que las descripciones funcionen de manera correcta; sin embargo, la única diferencia en este es que se incluyen bits de valid para validar la salida, entonces la señal solo pasa si el cuando el selector apunta a esa salida el valid también está en un uno, sino el flop se encarga de dar la salida anterior. En esta tarea no hace falta probar entrada por entrada y revisar salida por salida, ya que como se trabaja con resultados obtenidos de la tarea anterior, basta con compararlos y ver que sean iguales.

4. Istrucciones para ejecutar el programa.

Para ejecutar el programa, se tienen dos opciones validas. Estando en el directorio **src** y abriendo una terminal en esta dirección basta con escribir ***make***; sin embargo, esto lo hará de forma secuencial, es decir primero se abrirá el probador1, al cerrar el Gtk wave se abre el siguiente probador y así hasta terminar con los tres.

La segunda opción trata de decirle al make file cual probador se quiere ejecutar por lo tanto se hace: ***make parteA*** para el primer probador y así con los otros. Es necesario recalcar que independientemente de el método seleccionado para correr la tarea, se debe estar dentro del directorio **src**

```
all: parteA parteB parteC
        echo fin
parteA:
        iverilog bancoPrueba.v
        ./a.out
        rm a.out
        gtkwave mux2_2bit.vcd
parteB:
        iverilog bancoPrueba2.v
        ./a.out
        rm a.out
        gtkwave mux2_4bit.vcd
parteC:
        iverilog bancoPrueba3.v
        ./a.out
        rm a.out
        gtkwave mux4_4bit.vcd
```

5. Ejemplos de los Resultados.

5.1. Inciso A:

Para esta primera prueba se mantiene el plan de pruebas utilizado para la tarea #2, solo que ahora con la implementación del bit de valid. Aqui se puede observar como cuando el valid baja la salida se mantiene en la anterior, por lo que se puede decir que tanto diseño como la síntesis se hicieron de manera correcta y se comportan igual.

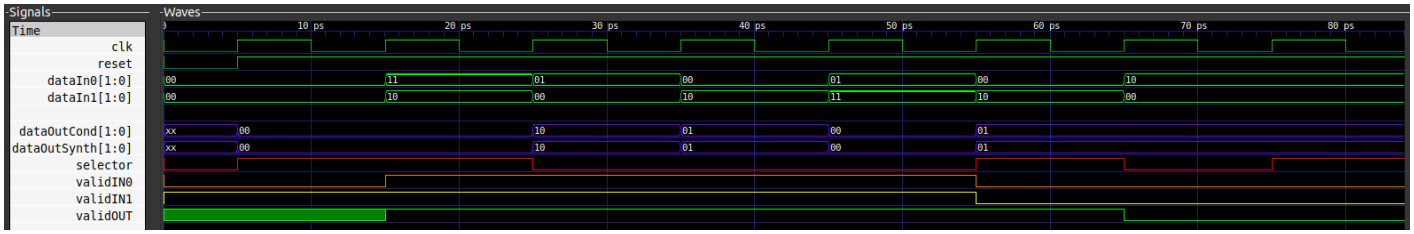

Figura 3: Resultado de las pruebas para el punto A (Creación Propia)

5.2. Inciso B:

El plan de pruebas para este es muy similar al anterior, solo que ahora se tienen entradas de cuatro bits, pero no hay diferencia en la implementación del probador, y se ve claramente como ambas descripciones se comportan igual, y que el bit de valid sirve de manera correcta ya que cuando el valid baja la entrada se mantiene.

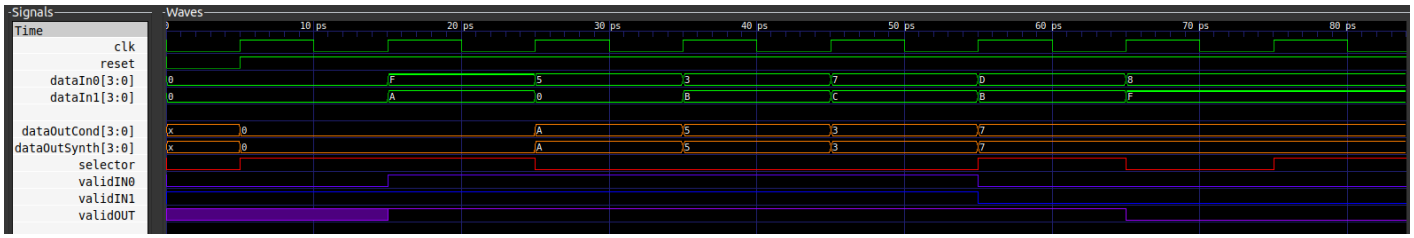

Figura 4: Resultado de las pruebas para el punto B (Creación Propia)

5.3. Inciso C:

En la siguiente figura de color amarillo se ven las salidas de los dos muxes colocados en paralelo con el fin de observar que salidas están entrando al mux final y confirmar que todo esté correcto. Para esta prueba las entradas se van a mantener en una entrada fija de tal forma que se pueda llevar un control total de manera sencilla de que entra y que sale de los muxes. La salida total se ve atrasada un ciclo de rejol respecto a los muxes paralelos debido a que ahora hay dos flops en serie. Se colocan todos los valid en unoi en esta prueba de tal forma que los datos siempre pasan.

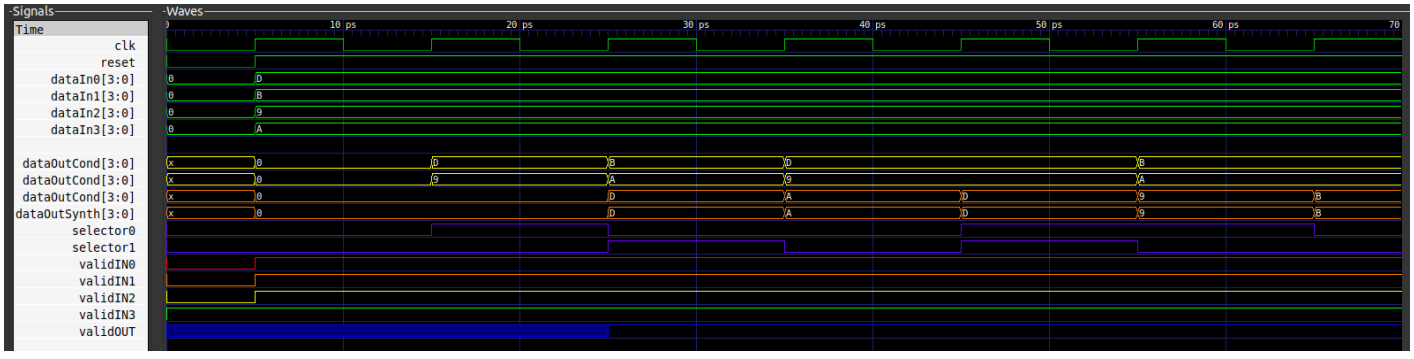

Figura 5: Resultado de las pruebas para el punto C (Creación Propia)

Ahora se hace la misma prueba pero cambiando los valid lo que entonces hace que se sostenga una señal por otro ciclo de reloj o lo que es igual que la salida sea el resultado anterior. Además se nota que ambas descripciones son iguales para ambos casos.

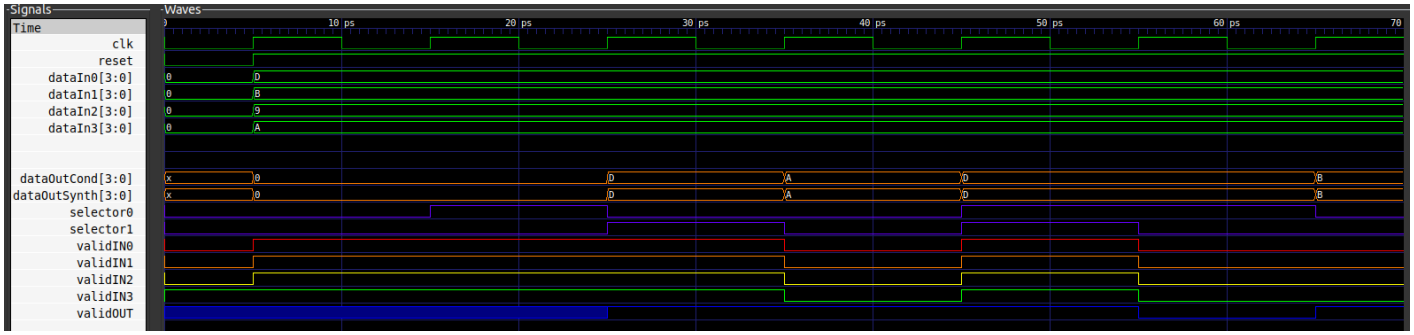

Figura 6: Resultado de las pruebas para el punto C (Creación Propia)

6. Análisis y conclusiones:

Esta tarea pone en evidencia lo poderoso e importante que puede llegar a ser un sofware de síntesis como yosys, esto debido a que se ahorra mucho tiempo si se tiene el modelo conductual correcto. No es necesario ir a buscar todas las compuertas lógicas que forman el sistema, sino que con solo las que se encuentran disponibles en la arquitectura deseada si se quiere trabajar con retardos. Por lo que trabajar con yosys es mas eficiente en cuanto a tiempo invertido.