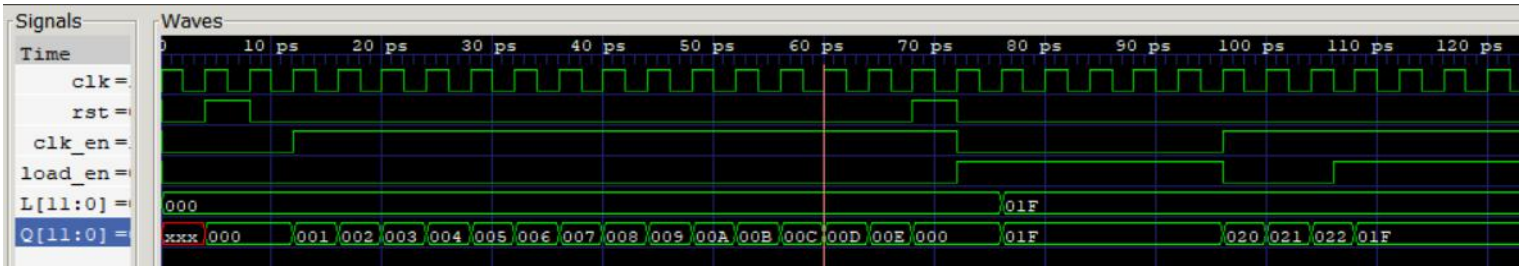
Carlos Cuellar
#19275

Lab #8:
Memoria en Verilog

Repositorio: *https://github.com/Cue19275/DIGITAL1UVG*

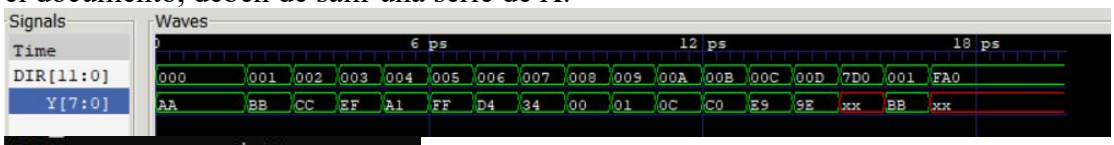
Ejercicio 1:
El modulo del archivo principlal se hace muy similar a como se diseñan los flip flops tipo D. Se necesitan 5 entradas para el diseño del contador requerido. Las entradas son: reloj, enable de la cuenta por medio del reloj, un reset, un enable para habilitar la carga, y un bus para la carga (de 12 bits en este caso). El contador tiene una única salida, la cual es el número de la cuenta. Por medio de condicionales se dictamina el funcionamiento del módulo contador. El primer condicional indica que cuando el reset está en uno, la salida es 0. El segundo (con menor prioridad), indica que la salida será igual a la carga, mientras que el enable de la carga esté en uno. Finalmente el último condicional indica que la salida será una unidad mayor a la misma salida, para cada flanco de reloj, cuando el enable del reloj este en 1. En el testbench se diseño a manera que se contara a partir de 0 en un inicio. Luego se reseteaba el contador, para habilitar la cuenta a partir de una carga. Finalmente se vuelce a resetear, y se deja el enable de la carga en 1 y la salida se encierra en el valor de la carga.



```
lk rst clk_en  load_en     L                    |Q
CD info: dumpfile Contador_tb.vcd opened for output.
   0   0        0       000000000000          |xxxxxxxxxxx
   0   0        0       000000000000          |xxxxxxxxxxx
   1   0        0       000000000000          |000000000000
   1   0        0       000000000000          |000000000000
   0   0        0       000000000000          |000000000000
   0   0        0       000000000000          |000000000000
   0   1        0       000000000000          |000000000001
   0   1        0       000000000000          |000000000001
   0   1        0       000000000000          |000000000010
   0   1        0       000000000000          |000000000010
   0   1        0       000000000000          |000000000011
   0   1        0       000000000000          |000000000011
   0   1        0       000000000000          |000000000100
   0   1        0       000000000000          |000000000100
   0   1        0       000000000000          |000000000101
   0   1        0       000000000000          |000000000101
   0   1        0       000000000000          |000000000110
   0   1        0       000000000000          |000000000110
   0   1        0       000000000000          |000000000111
   0   1        0       000000000000          |000000000111
   0   1        0       000000000000          |000000001000
   0   1        0       000000000000          |000000001000
   0   1        0       000000000000          |000000001001
   0   1        0       000000000000          |000000001001
   0   1        0       000000000000          |000000001010
   0   1        0       000000000000          |000000001010
   0   1        0       000000000000          |000000001011
   0   1        0       000000000000          |000000001011
   0   1        0       000000000000          |000000001100
   0   1        0       000000000000          |000000001100
   0   1        0       000000000000          |000000001101
   0   1        0       000000000000          |000000001101
   0   1        0       000000000000          |000000001110
   0   1        0       000000000000          |000000001110
   1   1        0       000000000000          |000000000000
   1   1        0       000000000000          |000000000000
   0   0        1       000000000000          |000000000000
   0   0        1       000000000000          |000000000000
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
   0   0        1       000000011111          |000000011111
```

Ejercicio 2:
El array se implementa a forma de matriz. Se tiene que definir la amplitud y la profundiad. Primer se asigna el la amplitud, en este caso era de 8 bits y profundidad de 4095. El $readmemb y el $readmemh indica que leera los datos de un archivo, en formato binario o hexadecimal respectivamente. La función toma dos argumentos, el primero corresponde al nombre del archivo y su tipo (.list en este caso) y el segundo argumento indica a aque variable se asignaran dichos valores. En nuestro codigo se cargan los valores de nuestro archivo .list a la matriz.

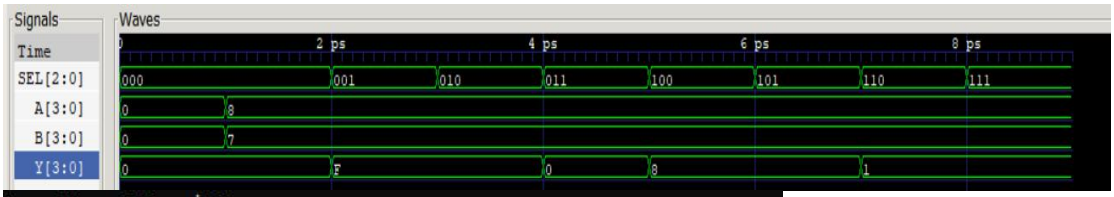
El código consiste en el diseño de un módulo que cuyo input es la driección de la celda de memoria cuyo dato se quiere extraer/leer, y el output siendo dicho dato. Dentro del módulo se tiene que establecer que la salida es igual al contenido de la dirección ingresada. Seguido a eso se establece la matriz de memoria, poniendo la profundiad y la amplitud. Luego de eso se hace el código para leer el archivo de datos en el formato deseado (binario/hex). En el tb, lo que se hace es ir cambiando el input, para que el output vaya vairando. Si se ingresa un input que no tiene valor definido en el documento, deben de salir una serie de X.




```
DIR             | Y
VCD info: dumpfile ROM_tb.vcd
000000000000    | 10101010
000000000001    | 10111011
000000000010    | 11001100
000000000011    | 11101111
000000000100    | 10100001
000000000101    | 11111111
000000000110    | 11010100
000000000111    | 00110100
000000001000    | 00000000
000000001001    | 00000001
000000001010    | 00001100
000000001011    | 11000000
000000001100    | 11101001
000000001101    | 10011110
011111010000    | xxxxxxxx
000000000001    | 10111011
111110100000    | xxxxxxxx
```

Ejercicio 3:

Para el diseño de la ALU se necesitan de 3 inputs: El primer operando, el segundo operando y el selector de operaciones. Los primeros dos deben de contener la misma cantidad de bits, y el último tiene bits según la cantidad de operaciones que se deseen implementar. Como en este caso se quieren realizar 7 operaciones, se usa un selector de 3 bits. La ALU cuenta con una salida única que es también del mismo numero de bits que los operandos, y esta salida es el resultado particular de las distintas operaciones que se pueden realizar. El funcionamiento del modulo consiste en la implementación de cases. Los cases se diseñan para indicar que va a pasar para cada valor que vaya a tener la variable que se observe, en este caso la del selector. De esa forma, se describe que pasa para cada valor y como se esta diseñando una ALU, para cada valor que pueda tomar el selector, se describe el valor de Y a partir de una operacion entre los operandos. Como se usan 7 operaciones y el numero de bits presta a que hayn 8 opciones, se crea un caso de default, el cual indica que pasa si hay un caso no descrito o si no se ha seleccionado nada. En el tb se programa para observar los resultados de Y a medida que cambia el valor del selector.




```
A   B   SEL  | Y
VCD info: dumpfile ALU_tb.vcd opened for output.
0000  0000  000   | 0000
1000  0111  000   | 0000
1000  0111  001   | 1111
1000  0111  010   | 1111
1000  0111  011   | 0000
1000  0111  100   | 1000
1000  0111  101   | 1000
1000  0111  110   | 0001
1000  0111  111   | 0001
gtkwave ALU_tb.vcd ALU_tb.gtkw
```