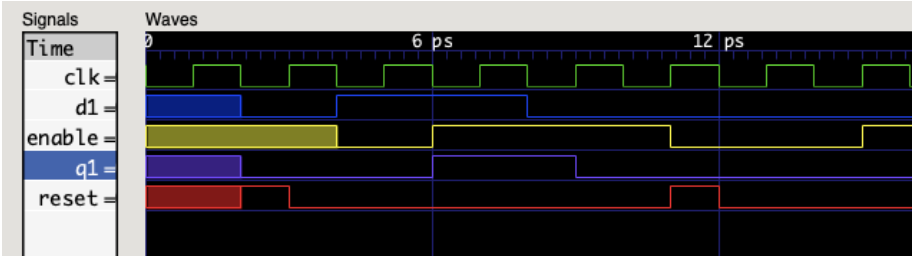
Daniela Morales 19213

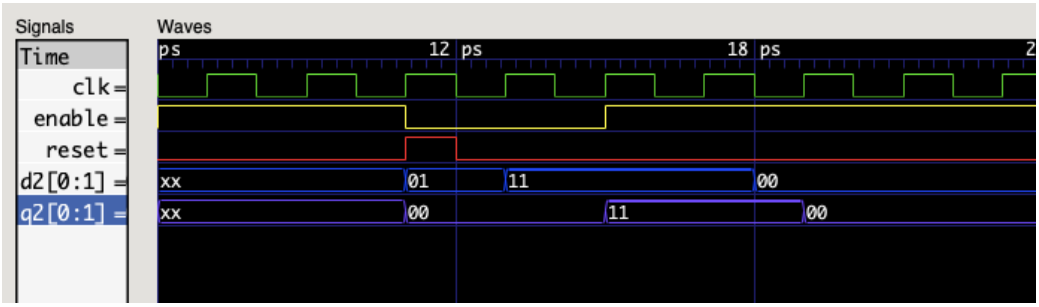
Lab 09

Ejercicio 1

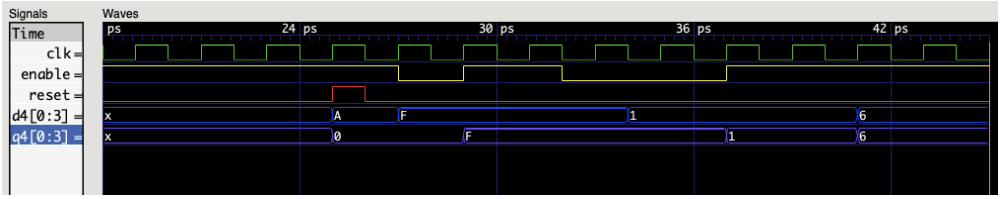
```
flipflop d
clk reset enable  d1 | q1
1     x     x     x  |  x
0     1     x     0  |  0
1     0     x     0  |  0
0     0     0     1  |  0
1     0     0     1  |  0
0     0     1     1  |  1
1     0     1     1  |  1
0     0     1     0  |  1
1     0     1     0  |  0
```




```
flipflop d
2 bits
clk reset enable   d2 | q2
0     0     1     xx  |  xx
1     1     0     01  |  00
0     0     0     01  |  00
1     0     0     11  |  00
0     0     0     11  |  00
1     0     1     11  |  11
0     0     1     11  |  11
1     0     1     11  |  11
0     0     1     00  |  11
1     0     1     00  |  00
0     0     1     00  |  00
1     0     1     00  |  00
0     0     1     00  |  00
1     0     1     00  |  00
```

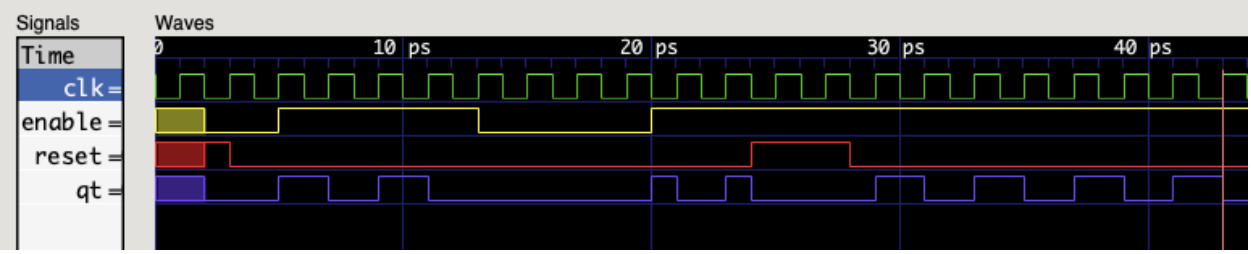



```
flipflop d
4 bits
clk reset enable    d4 | q4
0     0     1     xxxx  |  xxxx
1     1     1     1010  |  0000
0     0     1     1010  |  0000
1     0     0     1111  |  0000
0     0     0     1111  |  0000
1     0     1     1111  |  1111
0     0     1     1111  |  1111
1     0     1     1111  |  1111
0     0     0     1111  |  1111
1     0     0     1111  |  1111
0     0     0     0001  |  1111
1     0     0     0001  |  1111
0     0     0     0001  |  1111
1     0     1     0001  |  0001
0     0     1     0001  |  0001
1     0     1     0001  |  0001
0     0     1     0001  |  0001
1     0     1     0110  |  0110
0     0     1     0110  |  0110
1     0     1     0110  |  0110
0     0     1     0110  |  0110
1     0     1     0110  |  0110
gtkwave ej1_tb.vcd ej1_tb.gtkw
```



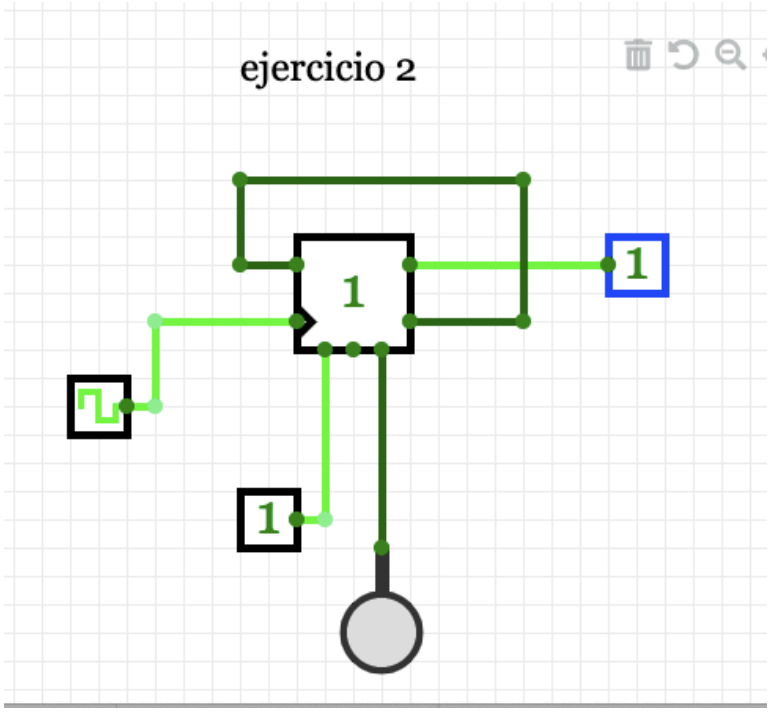
El primer ejercicio era un Flip flop tipo D, pero cada uno de diferente cantidad de bits. Se programo para que cuando el Enable este en 1 y haya un flanco positivo pase el valor de D hacia Q. Pero si el reset esta en 1, la salida es 0. Para cambiar la cantidad de bits, se indico en los inputs y outputs la cantidad deseada.

Ejercicio 2



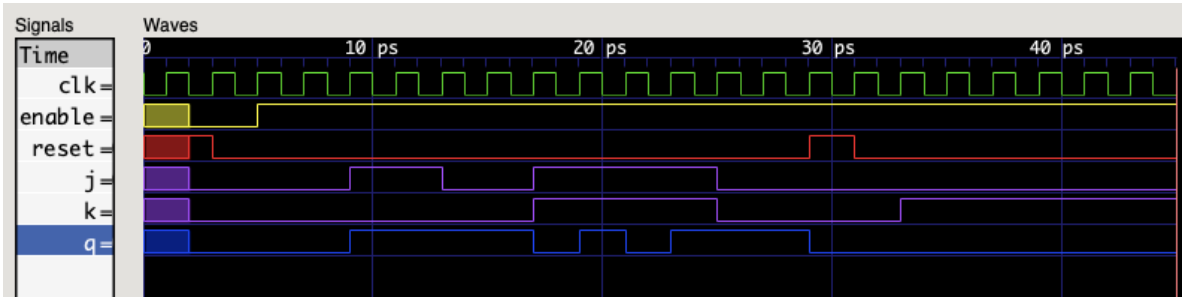
flipflop t

clk	reset	enable	q
1	x	x	x
0	1	0	0
1	0	0	0
0	0	0	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	0	0
0	0	0	0
1	0	0	0
0	0	0	0
1	0	0	0
0	0	0	0
1	0	0	0
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	0
0	1	1	0
1	1	1	0
0	0	1	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1
0	0	1	1
1	0	1	0
0	0	1	0
1	0	1	1

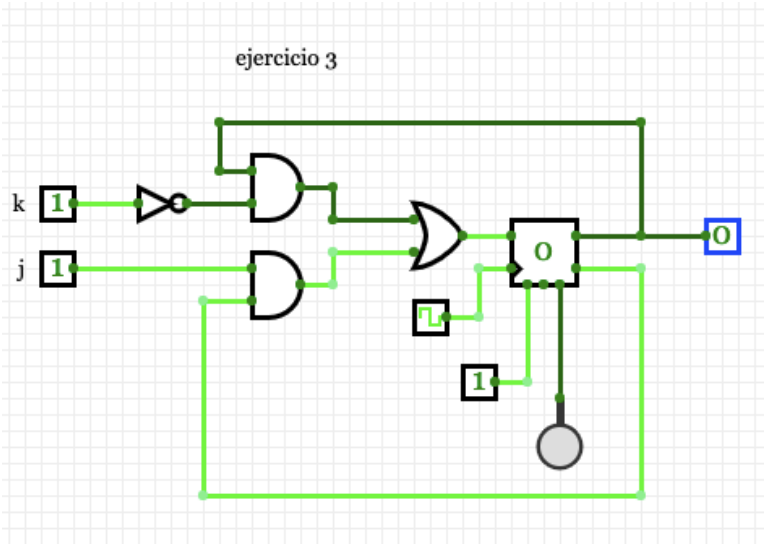


Este flipflop tipo T, se programo para que cuando el Enable este en 1, se invierta la salida Q cada flanco de reloj, pero si el reset esta en 1 la salida es 0. Para hacer esto se uso un flip flop tipo D, y la entrada d del Flip flop se indico con lógica combinacional usando solamente el valor de Q, ya que D es el inverso de Q.

Ejercicio 3

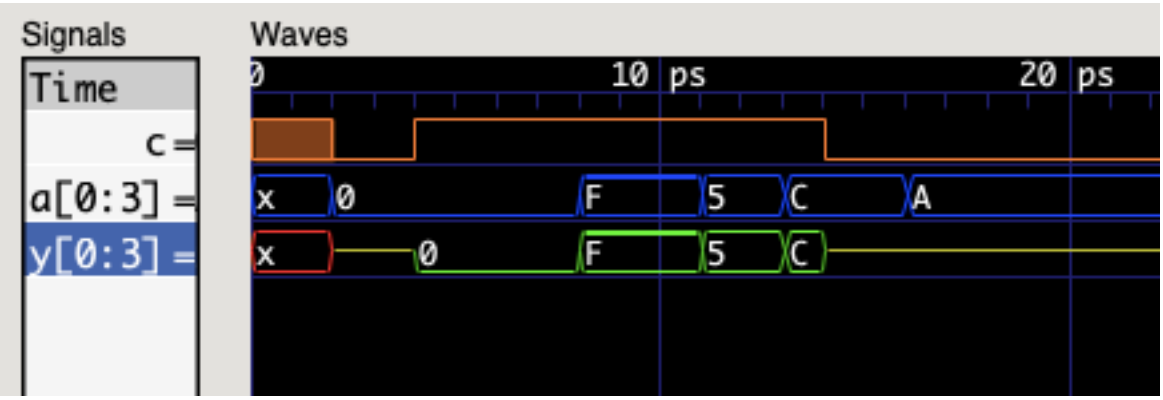



```
flipflop jk
clk reset enable j  k | q
1     x     x    x  x | x
0     1     0    0  0 | 0
1     0     0    0  0 | 0
0     0     0    0  0 | 0
1     0     1    0  0 | 0
0     0     1    0  0 | 0
1     0     1    0  0 | 0
0     0     1    0  0 | 0
1     0     1    1  0 | 1
0     0     1    1  0 | 1
1     0     1    1  0 | 1
0     0     1    1  0 | 1
1     0     1    0  0 | 1
0     0     1    0  0 | 1
1     0     1    0  0 | 1
0     0     1    0  0 | 1
1     0     1    1  1 | 0
0     0     1    1  1 | 0
1     0     1    1  1 | 1
0     0     1    1  1 | 1
1     0     1    1  1 | 0
0     0     1    1  1 | 0
1     0     1    1  1 | 1
0     0     1    1  1 | 1
1     0     1    0  0 | 1
0     0     1    0  0 | 1
1     0     1    0  0 | 1
0     0     1    0  0 | 1
1     1     1    0  0 | 0
0     1     1    0  0 | 0
1     0     1    0  0 | 0
0     0     1    0  0 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
0     0     1    0  1 | 0
1     0     1    0  1 | 0
```



Para hace el Flip flop tipo JK se realizo el mismo procedimiento que para el Flip flop T, se uso un Flip flop tipo d y se indico con lógica cominacional en valor de D. La ecuación necesaria era mas larga, ya que dependía de los valor de j, k y Q.

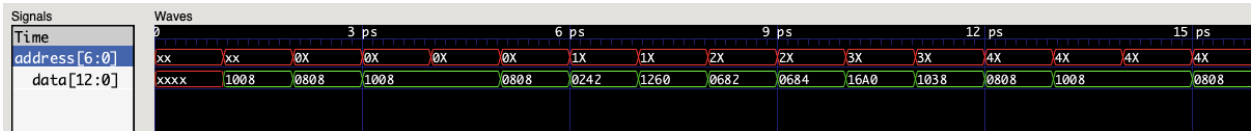
Ejercicio 4




```
buffer triestado
c    a    |  y
x   xxxx | xxxx
0   0000 | zzzz
1   0000 | 0000
1   1111 | 1111
1   0101 | 0101
1   1100 | 1100
0   1100 | zzzz
0   1010 | zzzz
```

En este ejercicio se utilizo un operador en Verilog, ya que si c esta en 0, la salida es alta impudencia, sin importar el valor de a. Pero en el caso de que c es 1, el valor de a pasa a y. Esto también se pudo haber realizado usando condicionales if.

Ejercicio 5




```
ejercicio 5
address |       data
xxxxxx0 | 1000000001000
00001x1 | 0100000001000
00000x1 | 1000000001000
00011x1 | 1000000001000
00010x1 | 0100000001000
0010xx1 | 0001001000010
0011xx1 | 1001001100000
0100xx1 | 0011010000010
0101xx1 | 0011010000100
0110xx1 | 1011010100000
0111xx1 | 1000000111000
1000x11 | 0100000001000
1000x01 | 1000000001000
1001x11 | 1000000001000
1001x01 | 0100000001000
1010xx1 | 0011011000010
1011xx1 | 1011011100000
1100xx1 | 0100000001000
1101xx1 | 0000000001001
1110xx1 | 0011100000010
1111xx1 | 1011100100000
1111110 | 1000000001000
0000111 | 0100000001000
0000001 | 1000000001000
0001111 | 1000000001000
0001011 | 0100000001000
0010001 | 0001001000010
0011101 | 1001001100000
```

Para este ejercicio fue necesario usar casex, para que reconociera que había don't cares en los inputs. Entonces al recibir un valor de alguno de los casos, la salida es indicada en el case y se muestra en la tabla. En las ultimas lineas de la tabla se puede ver que se escribió Address sin don't cares y Verilog si fue capaz de reconocerlas, se colocaron en estos casos los primeros 7 casos, que también se colocaron al inicio pero con don't cares, esto también se puede observar en el diagrama.