

Laboratorio 10

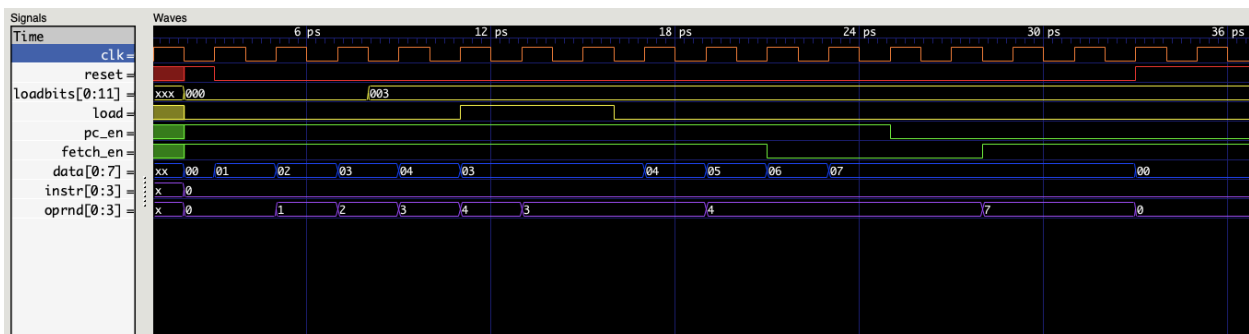
Ejercicio 1

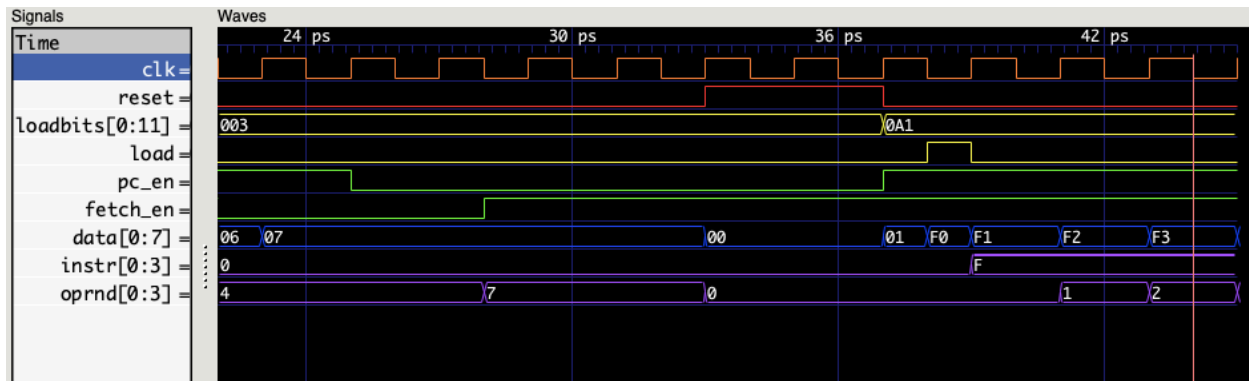
En este ejercicio se unió el program counter, la program ROM y el fetch, para realizar un bloque del Nibbler. El funcionamiento del bloque es que el program counter va contando e incrementando un bit cada flanco de reloj, también se le puede ingresar un valor de 12 bits y mientras la variable “load” este en 1, el valor de la salida del program counter será igual a la entrada de 12 bits. Este valor de 12 bits se envía directamente a la ROM la cual usando \$readmemh accede a la lista donde, por medio de los 12 bits de dirección que tiene como input, regresa el valor de 8 bits en la lista. Estos 8 bits son el program byte, que se colocó como salida del bloque y también es la entrada del fetch. El fetch es un flipflop D, que tiene como salida la instrucción (4 bits más significativos del program byte) y el operando (4 bits menos significativos del program byte). Debido a que el fetch es un flipflop, la salida de instrucción y operando es igual a los nibblers del program byte pero con un atraso de 1 flanco de reloj.

El fetch y program counter, ambos cuentan con su propio bit de Enable y usan el mismo bit de Clock y reset.

En la tabla a continuación se observa el funcionamiento del bloque, que inicia en 0 la variable data, que es el program byte y cada flanco de reloj incrementa 1. Se puede observar que también cada flanco de reloj, mientras el Enable del fetch este en 1, el valor del program byte pasa a instr y oprnd.

Cabe mencionar que en la lista el valor va incrementando de 00 a 18 y luego hace un salto a la posición A1 donde incrementa de f0 a fc. Por lo que el program byte, al estar el Enable del program counter en 1, incrementa 1 cada flanco y luego se observa que se pone en 0 porque se prende el reset. Pero al poner 0A1 en loadbits y prender el bit load, el valor del program byte es f0 y comienza nuevamente a incrementar porque en la lista el valor va incrementando.





ejercicio1

clk	e_pc	e_fetch	reset	loadbits	load	program byte	instr	oprnd
1	x	x	x	xxxxxxxxxxxx	x	xxxxxxx	xxxx	xxxx
0	1	1	1	000000000000	0	00000000	0000	0000
1	1	1	0	000000000000	0	00000001	0000	0000
0	1	1	0	000000000000	0	00000001	0000	0000
1	1	1	0	000000000000	0	00000010	0000	0001
0	1	1	0	000000000000	0	00000010	0000	0001
1	1	1	0	000000000000	0	00000011	0000	0010
0	1	1	0	000000000011	0	00000011	0000	0010
1	1	1	0	000000000011	0	00000100	0000	0011
0	1	1	0	000000000011	0	00000100	0000	0011
1	1	1	0	000000000011	1	00000011	0000	0100
0	1	1	0	000000000011	1	00000011	0000	0100
1	1	1	0	000000000011	1	00000011	0000	0011
0	1	1	0	000000000011	1	00000011	0000	0011
1	1	1	0	000000000011	1	00000011	0000	0011
0	1	1	0	000000000011	0	00000011	0000	0011
1	1	1	0	000000000011	0	00000100	0000	0011
0	1	1	0	000000000011	0	00000100	0000	0011
1	1	1	0	000000000011	0	00000101	0000	0100
0	1	1	0	000000000011	0	00000101	0000	0100
1	1	0	0	000000000011	0	00000110	0000	0100
0	1	0	0	000000000011	0	00000110	0000	0100
1	1	0	0	000000000011	0	00000111	0000	0100
0	1	0	0	000000000011	0	00000111	0000	0100
1	0	0	0	000000000011	0	00000111	0000	0100
0	0	0	0	000000000011	0	00000111	0000	0100
1	0	0	0	000000000011	0	00000111	0000	0100
0	0	1	0	000000000011	0	00000111	0000	0111
1	0	1	0	000000000011	0	00000111	0000	0111
0	0	1	0	000000000011	0	00000111	0000	0111
1	0	1	0	000000000011	0	00000111	0000	0111
0	0	1	0	000000000011	0	00000111	0000	0111
1	0	1	1	000000000011	0	00000000	0000	0000
0	0	1	1	000000000011	0	00000000	0000	0000
1	0	1	1	000000000011	0	00000000	0000	0000
0	0	1	1	000000000011	0	00000000	0000	0000
1	1	1	0	000010100001	0	00000001	0000	0000
0	1	1	0	000010100001	1	11110000	0000	0000
1	1	1	0	000010100001	0	11110001	1111	0000
0	1	1	0	000010100001	0	11110001	1111	0000
1	1	1	0	000010100001	0	11110010	1111	0001
0	1	1	0	000010100001	0	11110010	1111	0001
1	1	1	0	000010100001	0	11110011	1111	0010
0	1	1	0	000010100001	0	11110011	1111	0010
1	1	1	0	000010100001	0	11110100	1111	0011

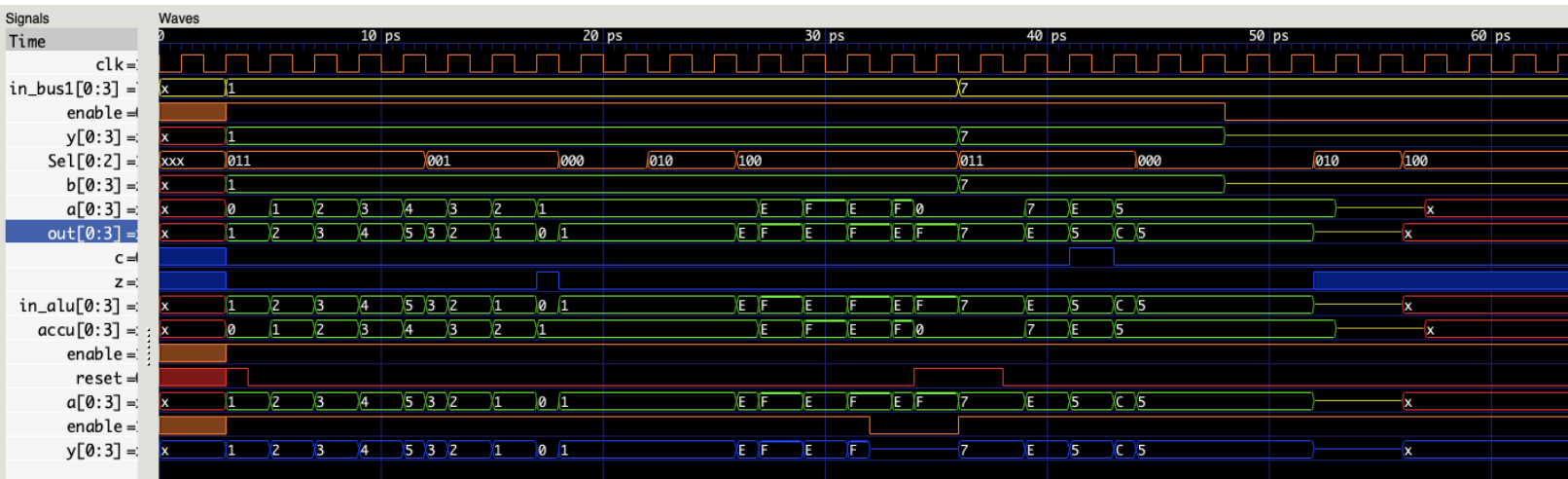
gtdwave ejercicio1_tb.vcd ejercicio1_tb.gtkw

Ejercicio 2

Para el segundo ejercicio se debía de tener una entrada de 4 bits que fuera a un buffer triestado y la salida de esta compuerta fuera una de las entradas de la ALU. La ALU puede realizar 5 operaciones distintas:

- Dejar pasar a
- Dejar pasar b
- Restar a menos b
- Sumar a y b
- Nand entre a y b

La salida de la ALU va a la entrada del acumulador, que es un flipflop de 4 bits que su salida va a la entrada “a” de la alu. Al mismo tiempo, la salida de la alu va a la entrada de un segundo buffer triestado, y la salida de este componente ya son los outputs del bloque.



ejercicio2											
clk	reset	accu_en	bus1_en	bus2_en	in_bus1	Sel	c	z	out		
1	x	x	x	x	xxxx	xxx	x	x	xxxx		
0	x	x	x	x	xxxx	xxx	x	x	xxxx		
1	1	1	1	1	0001	011	0	0	0001		
0	0	1	1	1	0001	011	0	0	0001		
1	0	1	1	1	0001	011	0	0	0010		
0	0	1	1	1	0001	011	0	0	0010		
1	0	1	1	1	0001	011	0	0	0011		
0	0	1	1	1	0001	011	0	0	0011		
1	0	1	1	1	0001	011	0	0	0100		
0	0	1	1	1	0001	011	0	0	0100		
1	0	1	1	1	0001	011	0	0	0101		
0	0	1	1	1	0001	001	0	0	0011		
1	0	1	1	1	0001	001	0	0	0010		
0	0	1	1	1	0001	001	0	0	0010		
1	0	1	1	1	0001	001	0	0	0001		
0	0	1	1	1	0001	001	0	0	0001		
1	0	1	1	1	0001	001	0	1	0000		
0	0	1	1	1	0001	000	0	0	0001		
1	0	1	1	1	0001	000	0	0	0001		
0	0	1	1	1	0001	000	0	0	0001		
1	0	1	1	1	0001	000	0	0	0001		
0	0	1	1	1	0001	010	0	0	0001		
1	0	1	1	1	0001	010	0	0	0001		
0	0	1	1	1	0001	010	0	0	0001		
1	0	1	1	1	0001	010	0	0	0001		
0	0	1	1	1	0001	100	0	0	1110		
1	0	1	1	1	0001	100	0	0	1111		
0	0	1	1	1	0001	100	0	0	1111		
1	0	1	1	1	0001	100	0	0	1110		
0	0	1	1	1	0001	100	0	0	1110		
1	0	1	1	1	0001	100	0	0	1111		
0	0	1	1	0	0001	100	0	0	zzzz		
1	0	1	1	0	0001	100	0	0	zzzz		
0	1	1	1	0	0001	100	0	0	zzzz		
1	1	1	1	0	0001	100	0	0	zzzz		
0	1	1	1	1	0111	011	0	0	0111		
1	1	1	1	1	0111	011	0	0	0111		
0	0	1	1	1	0111	011	0	0	0111		
1	0	1	1	1	0111	011	0	0	1110		
0	0	1	1	1	0111	011	0	0	1110		
1	0	1	1	1	0111	011	1	0	0101		
0	0	1	1	1	0111	011	0	0	1100		
1	0	1	1	1	0111	000	0	0	0101		
0	0	1	1	1	0111	000	0	0	0101		
1	0	1	1	1	0111	000	0	0	0101		
0	0	1	0	1	0111	000	0	0	0101		
1	0	1	0	1	0111	000	0	0	0101		
0	0	1	0	1	0111	000	0	0	0101		
1	0	1	0	1	0111	000	0	0	0101		
0	0	1	0	1	0111	010	0	x	zzzz		
1	0	1	0	1	0111	010	0	x	zzzz		
0	0	1	0	1	0111	010	0	x	zzzz		
1	0	1	0	1	0111	100	0	x	xxxx		
0	0	1	0	1	0111	100	0	x	xxxx		
1	0	1	0	1	0111	100	0	x	xxxx		
0	0	1	0	1	0111	100	0	x	xxxx		
1	0	1	0	1	0111	100	0	x	xxxx		
0	0	1	0	1	0111	100	0	x	xxxx		
1	0	1	0	1	0111	100	0	x	xxxx		
0	0	1	0	1	0111	100	0	x	xxxx		
1	0	1	0	1	0111	100	0	x	xxxx		
0	0	1	0	1	0111	100	0	x	xxxx		
1	0	1	0	1	0111	100	0	x	xxxx		

Como se puede observar en el diagrama de timing y en la tabla, al inicio Sel esta en 011 que es la suma entre a y b, y como la entrada del bus 1 es 1 y la entrada a de la ALU inicia en 0 porque se hace reset, cada flanco de reloj se suma 1 y luego este valor

entra al acumulador y la salida de este, un flanco de reloj después, es la entrada “a” de la ALU. Luego se colocó el Sel 001, que es la resta entre a y b, por lo tanto comienza a decrementar uno ya que la entrada del bus1 es 1, se puede observar que cuando el valor de salida llega a 0, se prende la bandera z.

Luego, Sel se coloca en 000, que es dejar pasar “a” que en ese momento su valor es 1 y luego se coloca Sel en 010 que es dejar pasar “b” que en ese momento su valor también es 1. Luego se hace un nand entre “a” y “b”.

Luego, se apaga el Enable del segundo bus, por lo que la salida del bloque se encuentra en alta impedancia. Después se resetea el acumulador, se pone en 1 el enable, se ingresa el valor “7” a la entrada del primer bus y se asigna 011 a Sel, es decir, la ALU sumará a y b. Por lo tanto, la primera suma entre 7 y 0 es 7, pero luego entre 7 y 7 es E, y al sumar E y 7, da 15, es decir, hay un overflow por lo que se prende la bandera c. Luego se pone en 000 Sel, por lo que solo pasará el valor de a, que en ese momento es 5. Debido a que esta pasando el valor de a, la operación en la ALU no es afectada por el valor de b, entonces al apagar el enable del bus1 no afecta la salida, hasta que Sel es 010, ya que eso es pasar “b”, por lo que la salida del bloque es alta impedancia. Y al poner Sel en 100 e intentar hacer un nand entre dos variables en altas impedancia, la salida es x.

Para hacer esto se uso un case para la alu, para asignarle los valores a una variable out1 de 5 bits y los 4 bits menos significativos, son la salida de la alu y se uso otro case para asignarle valor a la bandera c, para cualquier valor de sel que no es el de la suma o resta es 0 pero para estos dos casos es el 4 bit de out1.

Los buses son buffers triestado que se hicieron con un operador, si c esta en uno, la salida es la entrada, si no es alta impedancia.

Código

<https://github.com/mor19213/digital.git>