Daniela Morales
19213

Sección 10

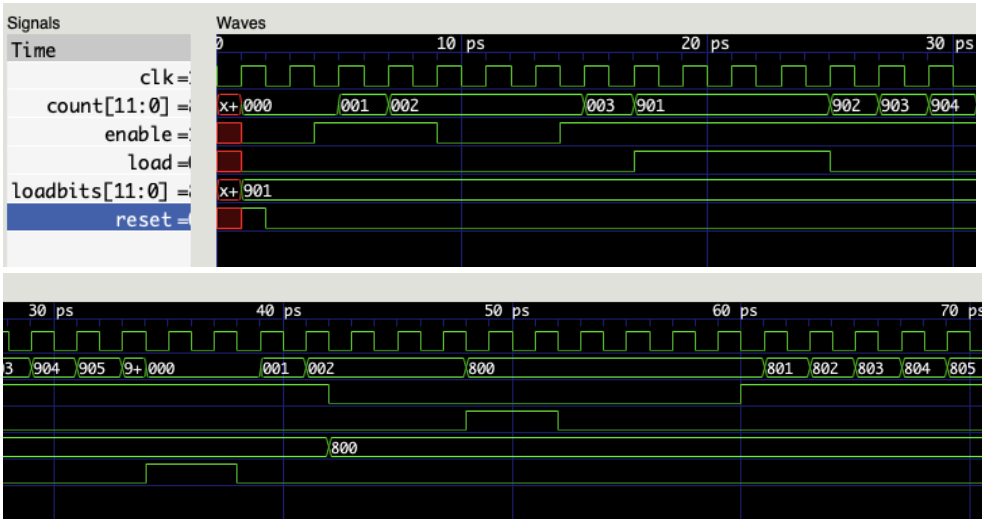
Laboratorio 8

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

Ejercicio 1: Contador de 12 bits

Cuando hay un flanco positivo en el Clock, reset o load se hacen 3 condicionales. La primera condicional es que si el Enable esta en 1, entonces Count incrementara un valor cada flanco de reloj. La segunda es que si el load esta en 1, se pasa el valor en la variable “loadbits” a la salida del contador. Y la tercera condicional es que si el reset esta en uno, se pone en 0 el valor del contador.

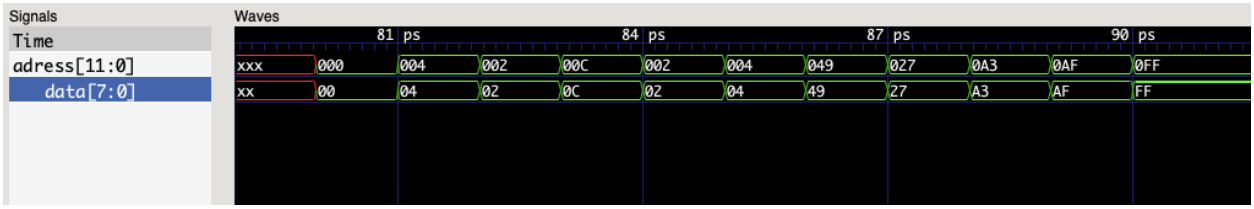
```
module contador(
    input clk, reset, enable, load, input[11:0]loadbits,
    output reg[11:0] count
);

always @ (posedge clk, posedge reset, posedge load) begin


if(enable) begin
        count <= count + 1'b1;
    end

if(load) begin
    count <= loadbits;
    end

if(reset) begin
    count <= 000000000000;
    end

end
endmodule
```



En el diagrama de timing anterior se puede observar que primero se tiene que resetear el counter, luego al estar en 1 el Enable se comienza a contar. Luego, al apagar el Enable deja de contar y se queda en el mismo número. Luego, se volvió a prender el Enable y se vuelve a contar y al prender el load, se pasa el valor de load bits hacia el Count y este vuelve a contar hasta apagar el bit load. De la misma manera, al estar en 0 el Enable y prender el bit load se pasa el valor de loadbits hacia el counter, es decir, la actualización del load es independiente del bit Enable.

Ejercicio 2: Memoria ROM

```
// ejercicio 2
module ROM(
    input logic [11:0] adress,
    output wire [7:0] data);
reg[0:11] memoria [0:4095];

initial begin
    $readmemh("memory_list", memoria);
    end
assign  data = memoria[adress];
endmodule
```



Se coloco cómo entrada solamente una variable de 12 bits de direccionamiento y de salida se tiene una variable de 8 bits que son los valores que se recuperan de la memoria en la base de datos. Luego se indica que se utilizará una matriz de 12 por 4096 bits llamada memoria.

Un array se utiliza para agrupar elementos en arreglos de varias dimensiones. Para establecer los arrays multidimensionales se tienen que establecer el valor mínimo y máximo de bits en cada dimensión usando arrays de una dimensión.

El reademb y readmemh funcionan y se implementan de maneras similares, pero el readmemb es para archivos con valores en binario y el readmemh es para archivos con valores en hexadecimal. Estos se usan para mover los valores en el archivo a la matriz planteada al inicio. Finalmente con un assign a la variable “data” se copia a esta variable, el valor en la matriz “memoria” en la linea que se establece con la variable Address.

En el archivo con valores hexadecimales se colocaron los números de 00 a ff, por lo tanto, se puede observar en el diagrama de timing que, como se esperaba, los dos bits menos significativos de Address son iguales a los dos bits de data.

Ejercicio 3: ALU

Para simular el funcionamiento de la ALU se usa case, por medio de la cual según el valor en la variable de input “sel” se determina cuál operación se realizara para asignarle un valor de salida. Existen 8 posibles funciones de la ALU debido a que se tienen 3 bits de selección, en la selección 011 se colocó que la salida fuera igual a 0 ya que no es usada esa función. En los demás posibles valores del “sel” se asignaron operaciones utilizando los inputs de 4 bits A y B, estos con forma behavioral.

Daniela Morales
19213

Sección 10

```
module alu(
          input wire [3:0] A,
          input wire [3:0] B,
          input logic [2:0] Sel,
          output logic [3:0] out
    );
    always @(*)
    begin
        case(Sel)
        0:
            out = A & B;
        1:
            out = A | B;
        2:
            out = A + B;
        3:
            out = 4'b0000;
        4:
            out = A & ~B;
        5:
            out = A | ~B;
        6:
            out = A - B;
        7:
            out = (A < B) ? 4'b1111 : 4'b0000;
        default:
            out = 4'b0000;
        endcase
    end

endmodule
```

ALU

A	B	sel	out
0101	0110	000	0100
1101	0010	000	0000
0101	0110	001	0111
1001	0110	001	1111
0101	0110	010	1011
0111	0100	010	1011
0101	0110	011	0000
0001	1110	011	0000
0101	0110	100	0001
0110	1000	100	0110
0101	0110	101	1101
0001	0100	101	1011
0101	0110	110	1111
1111	0001	110	1110
0101	0110	111	1111
1111	0100	111	0000