Pre-Informe 3

Patricio Calderón
201773056-0

Pregunta 1: Investigación de Conceptos

1. Un lenguaje o dialecto especializado para definir la operación, estructura y diseño de circuitos electrónicos como un reloj o cualquier antena satélite o de redes inalámbricas. Dan una descripción formal de un circuito electrónico para el análisis y simulación. A diferencia de un lenguaje de programación solo están diseñados para describir hardware y no pueden representar o simular software.

2. wire pertenece al tipo de datos net el cual representa las conecciones físicas entre módulos o compuertas por lo tanto es usada para recibir o enviar información, no almacena información solo la mantiene temporalmente. Por otra parte reg pertenece al tipo de datos variable el cual mantiene los datos que se la asignan hasta la próxima asignación (tal cual como los registers). Y finalmente logic es un tipo de variable (comodin) que sirve para representar ambos casos anteriores dado que se adapta segun la necesidad. Por ejemplo en un modulo reg seria la entrada CLK, reg seria el estado Q y logic podría ser ambos.

3. "<=" en SystemVerilog es llamado non-blocking assignment el cual es usado para las secciones de lógica secuencial (ej: always) y "=" en SystemVerilog es llamado blocking assignment el cual es usado para las secciones de lógica combinatoria (ej: assign). Se debe evitar usar estos operadores en la misma sección.

4. Es la declaración de un 16-bit ([15:0]) register (reg) llamado g (g) el cual esta siendo usado para guardar un valor hexadecimal (A6B2 = 1010011010110010).

5. Como vemos en primera instancia first que es un reg se le asigna un 0 y luego a second que es un wire se le asigna el negado de first. Por lo tanto cuando comienza el always a first se le asigna 1 de manera instantanea y luego se modifica arr usando el first actualizado con valor 1 pero como second esta establecido como un wire con assign fuera del always este no se alcanza a actualizar con el valor nuevo de first por lo tanto queda arr[1] = arr[1]. Vemos que lo anterior se debe al delay que tiene los assign o operaciones sincronicas respecto a los always o operaciones que deberian ser sincronicas, cabe mencionar que es mala practica usar el operador "=" dentro de un always.

Pregunta 2: Análisis de Código

```
module Modulo ( input logic [2:0] r,
                input logic clk ,
                output logic [7:0] n);
reg [7:0] x = 8'b0 ;
reg [7:0] y = 8'b01110111 ;
reg [2:0] z = 3'b0 ;
wire [2:0] m;
assign n = ( x + y )%256;
assign m = ( z + 1)%8;
reg s = 1'b0;
always_ff@(posedge clk) begin
    if ( ~( s )) begin
```

```
            z <= r;
            s <= 1;
        end
        y [z] <= ( ~( y [m] ));
        x [m] <= ( ~( y [ z ]));
        z <= (z + 1)%8;
    end
    endmodule
```

Comenzamos con un r = 3'b0;

nº clock	n	x	y	z	s
1	01110110	00000000	01110110	001	1
2	01110100	00000000	01110100	010	1
3	01110100	00000000	01110100	011	1
4	10000100	00010000	01110100	100	1
5	01110100	00010000	01100100	101	1
6	01010100	00010000	01000100	110	1
7	01010100	00010000	01000100	111	1
8	11010101	00010001	11000100	000	1
9	11011000	00010011	11000101	001	1
10	11011100	00010111	11000101	010	1

Pregunta 3: Diseño de modulo

Testbench en Modelsim:

```
` timescale 1ns/1ps
module Fibonacci_Counter_tb;
logic [4:0] counter;
logic fib, clk;

Fibonacci_Counter FC(.clk (clk), .counter (counter), .fib (fib));

initial begin
        clk = 0;
        #10;
end

always begin
      #1 clk = ~clk;
end

integer file_out;

initial begin
        file_out = $fopen("fb_out.txt", "wb");
        $fstrobe(file_out, "time\tclk\tcounter\tfib");
end
always @ (posedge clk)
begin
$fstrobe(file_out,"%3d\t%b\t%b\t%b",$time,clk,counter,fib);
end
endmodule
```

Modulo en Modelsim:

```
module Fibonacci_Counter (  input logic clk,
                            output logic [4:0] counter,
                            output logic fib);
    logic [4:0] counter_out = 5'b0;
    logic fib_out = 1;
    logic state = 0;
    logic [4:0] previous = 5'b00000;
    logic [4:0] current = 5'b00001;
    assign counter = counter_out;
    assign fib = fib_out;

    always @ (posedge clk) begin
        if (counter_out == 5'b0 | (counter_out + 1)%32  == 5'b0 ) begin
            previous <= 5'b00000;
            current <= 5'b00001;
            fib_out <= 1;
            end
        else if (counter_out + 1 == 2 * current + previous) begin
            fib_out <= 1;
            previous <= current;
            current <= current + previous;
            end
        else begin
            fib_out <= 0;
            end
        state <= 0;
        counter_out <= counter_out + 1;
    end
endmodule
```

Fuentes

- https://sudip.ece.ubc.ca/modelsim-systemverilog/

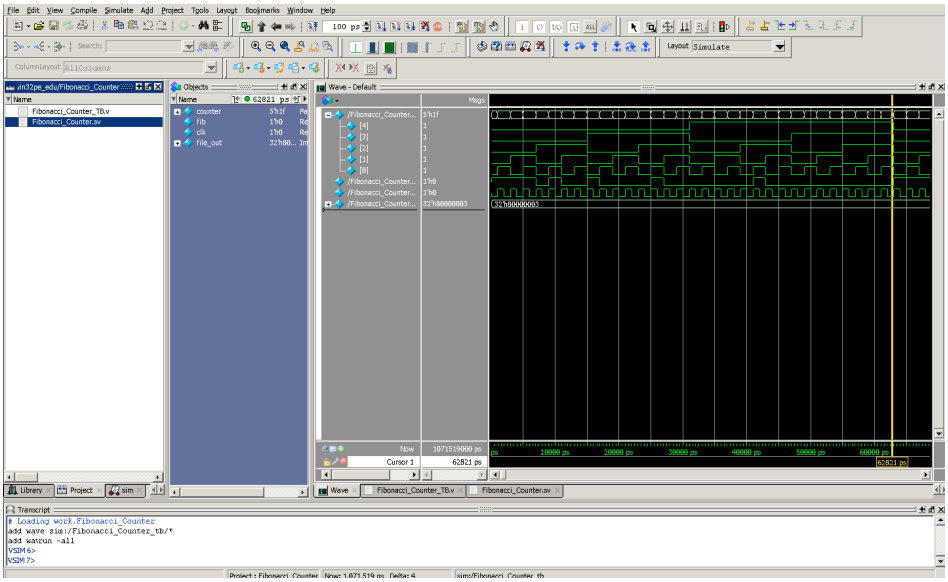
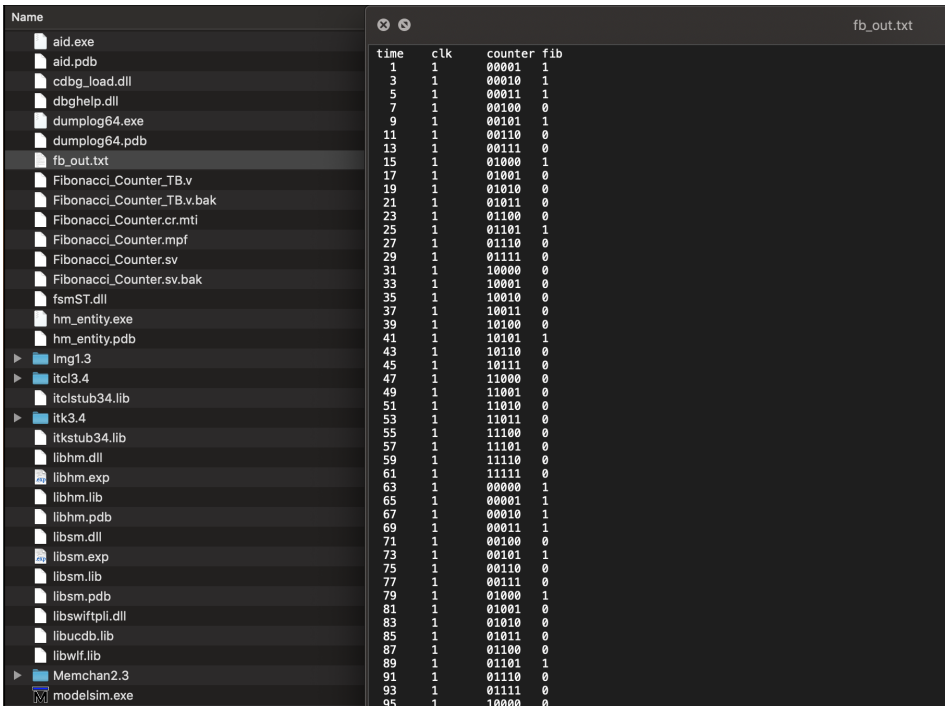


Figure 1: Simulacion en ModelSim usado Signals in Region



Name

aid.exe
aid.pdb
cdbg_load.dll
dbghelp.dll
dumplog64.exe
dumplog64.pdb
fb_out.txt
Fibonacci_Counter_TB.v
Fibonacci_Counter_TB.v.bak
Fibonacci_Counter.cr.mti
Fibonacci_Counter.mpf
Fibonacci_Counter.sv
Fibonacci_Counter.sv.bak
fsmST.dll
hm_entity.exe
hm_entity.pdb
Img1.3
itcl3.4
itclstub34.lib
itk3.4
itkstub34.lib
libhm.dll
libhm.exp
libhm.lib
libhm.pdb
libsm.dll
libsm.exp
libsm.lib
libsm.pdb
libswiftpli.dll
libucdb.lib
libwlf.lib
Memchan2.3
modelsim.exe

fb_out.txt

time	clk	counter	fib
1	1	00001	1
3	1	00010	1
5	1	00011	1
7	1	00100	0
9	1	00101	1
11	1	00110	0
13	1	00111	0
15	1	01000	1
17	1	01001	0
19	1	01010	0
21	1	01011	0
23	1	01100	0
25	1	01101	1
27	1	01110	0
29	1	01111	0
31	1	10000	0
33	1	10001	0
35	1	10010	0
37	1	10011	0
39	1	10100	0
41	1	10101	1
43	1	10110	0
45	1	10111	0
47	1	11000	0
49	1	11001	0
51	1	11010	0
53	1	11011	0
55	1	11100	0
57	1	11101	0
59	1	11110	0
61	1	11111	0
63	1	00000	1
65	1	00001	1
67	1	00010	1
69	1	00011	1
71	1	00100	0
73	1	00101	1
75	1	00110	0
77	1	00111	0
79	1	01000	1
81	1	01001	0
83	1	01010	0
85	1	01011	0
87	1	01100	0
89	1	01101	1
91	1	01110	0
93	1	01111	0
95	1	10000	0

Figure 2: Archivo de los resultados de la simulacion