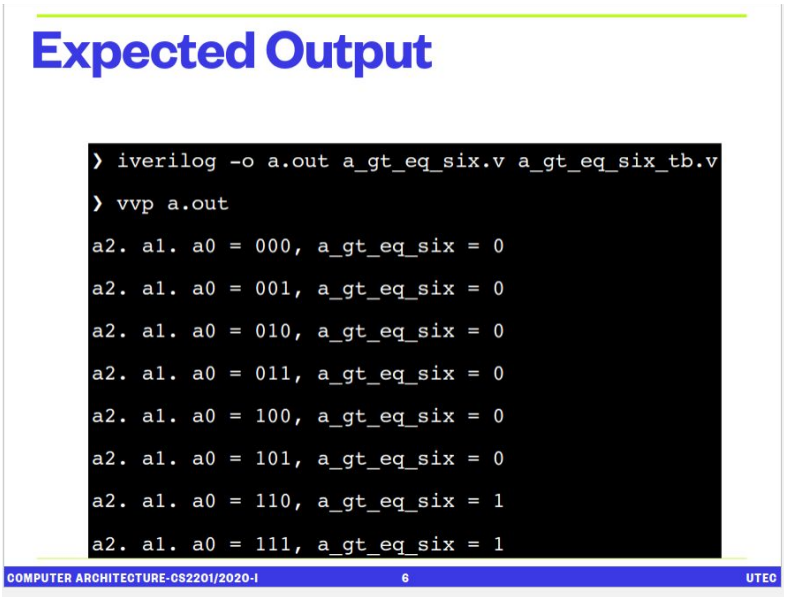
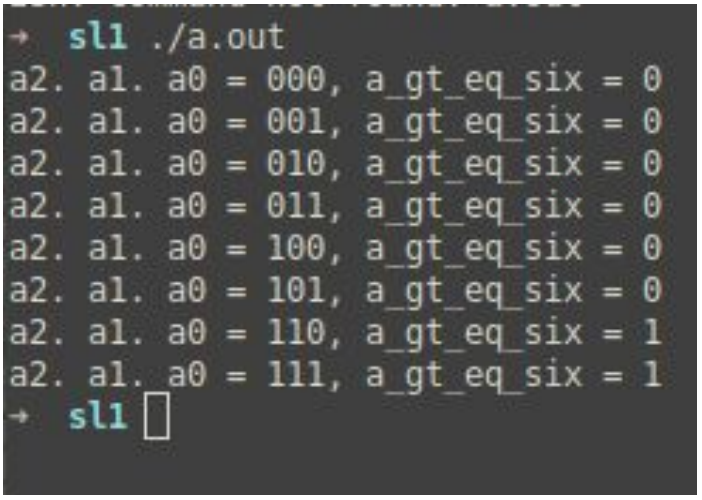
Laboratorio 04

Ejercicios slides

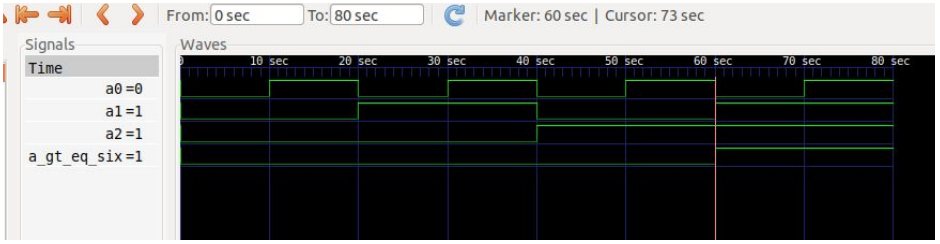
1. Obtener y analizar el waveform de la FSM simple [Ejercicio 1 FSM]



Output conseguido



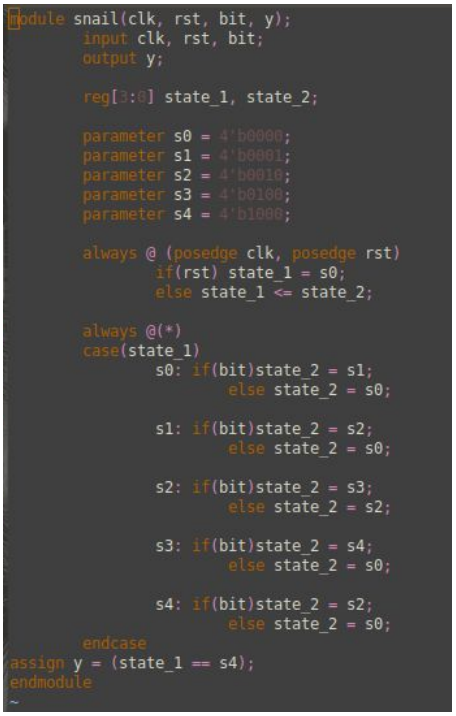
Entonces puedo confirmar que el código funciona, entonces es hora de verificar el gtkwave



Efectivamente, como se puede ver en el gtkwave el valor de a_gt_eq_six depende de a2 y a1, si estos están en 1 el valor de a_gt_eq_six es 1, de lo contrario es 0. (a1 y a2 son 1 solamente cuando el número es mayor a 6)

2. Realizar la implementación behavioral de la FSM con tres salidas Z1, Z2, Z3.

3. Realizar la implementación behavioral de la FSM del ejercicio del caracol.



```
module snail(clk, rst, bit, y);
        input clk, rst, bit;
        output y;

        reg[3:0] state_1, state_2;

        parameter s0 = 4'b0000;
        parameter s1 = 4'b0001;
        parameter s2 = 4'b0010;
        parameter s3 = 4'b0100;
        parameter s4 = 4'b1000;

        always @ (posedge clk, posedge rst)
                if(rst) state_1 = s0;
                else state_1 <= state_2;

        always @(*)
        case(state_1)
                s0: if(bit)state_2 = s1;
                        else state_2 = s0;

                s1: if(bit)state_2 = s2;
                        else state_2 = s0;

                s2: if(bit)state_2 = s3;
                        else state_2 = s2;

                s3: if(bit)state_2 = s4;
                        else state_2 = s0;

                s4: if(bit)state_2 = s2;
                        else state_2 = s0;
        endcase
assign y = (state_1 == s4);
endmodule
~
```

Ejercicio: FSM del Thunderbird

Diseño

Diagrama de estados

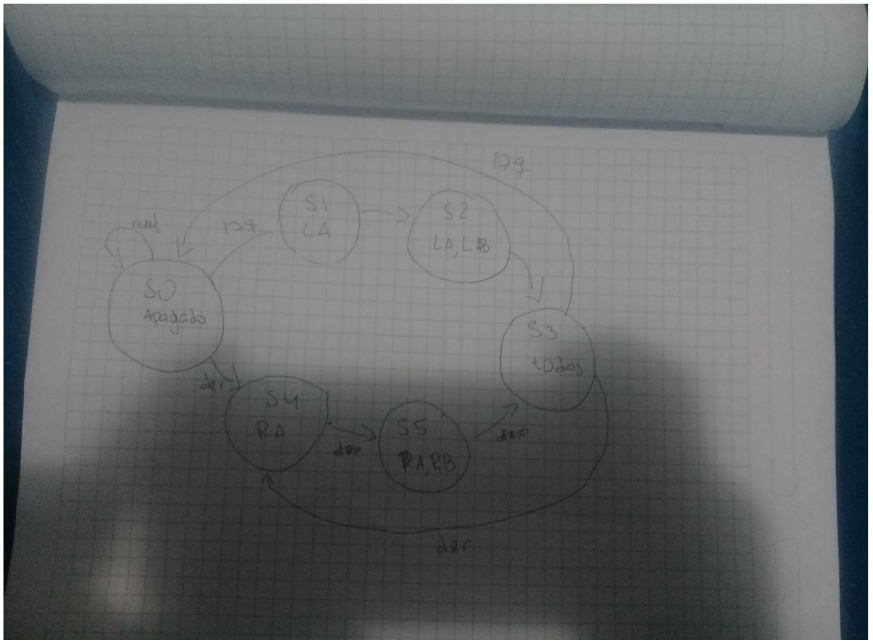


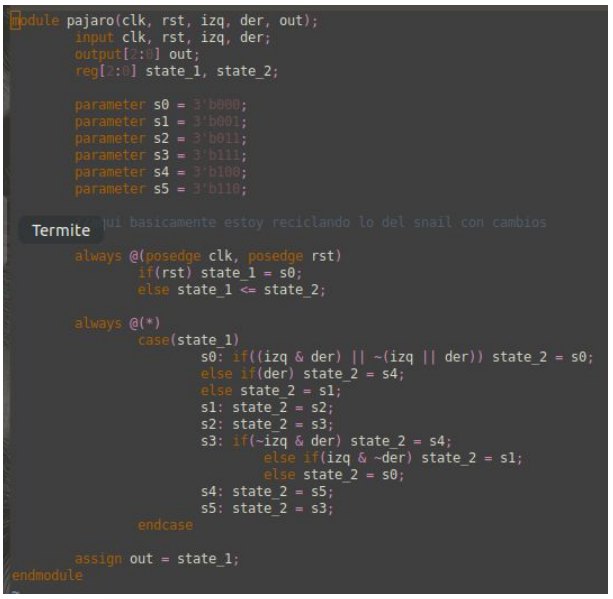
Tabla de transición de estados

s	reset	derecha	izquierda	s’
x	1	x	x	s0
s0	0	1	1	s0
s0	0	0	0	s0
s0	0	1	0	s4
s0	0	0	1	s1
s1	0	x	x	s2

s2	**0**	**x**	**x**	**s3**
s3	**0**	**1**	**0**	**s4**
s3	**0**	**0**	**1**	**s1**
s3	**0**	**0**	**0**	**s0**
s3	**0**	**1**	**1**	**s0**
s4	**0**	**x**	**x**	**s5**
s5	**0**	**x**	**x**	**s3**

Básicamente lo que hice en el diseño fue analizar el video que se nos adjuntó en el documento y en base a eso empecé a formar una idea que como funcionaria esta simulación. Como se aprecia en el video, una vez prendida una de las luces, sea de izquierda o de derecha, las otras luces se prenden después de un tiempo x sin tener que recibir ningún otro input.

Implementación




```
module pajaro(clk, rst, izq, der, out);
	input clk, rst, izq, der;
	output[2:0] out;
	reg[2:0] state_1, state_2;

	parameter s0 = 3'b000;
	parameter s1 = 3'b001;
	parameter s2 = 3'b011;
	parameter s3 = 3'b111;
	parameter s4 = 3'b100;
	parameter s5 = 3'b110;

	// Aqui basicamente estoy reciclando lo del snail con cambios

	always @(posedge clk, posedge rst)
		if(rst) state_1 = s0;
		else state_1 <= state_2;

	always @(*)
		case(state_1)
			s0: if((izq & der) || ~(izq || der)) state_2 = s0;
			else if(der) state_2 = s4;
			else state_2 = s1;
			s1: state_2 = s2;
			s2: state_2 = s3;
			s3: if(~izq & der) state_2 = s4;
				else if(izq & ~der) state_2 = s1;
				else state_2 = s0;
			s4: state_2 = s5;
			s5: state_2 = s3;
		endcase

	assign out = state_1;
endmodule
```

Como dice el comentario en la implementación, se está usando una estructura parecida al del caracol del ejercicio anterior, debido a que ambos son máquinas de estados. Pero muy aparte de eso podemos ver que en los cases se está aplicando lo mostrado en la tabla superior. Partimos del estado 0 y dependiendo de que reciba este nos moveremos al siguiente estado.

```
`timescale 1ns/1ns
module pajaro_tb;
	reg clk, rst, izq, der;
	output wire [2:0] out;
	always #1 clk = ~clk;
	always #1 rst = 0;
	pajaro dodo(clk, rst, izq, der, out);
	initial begin
		clk <= 1;
		rst <= 1;
		izq <= 1;
		der <= 0;
		#3 izq = 1;
		#3 der = 0;
		#5 der = 1;
		#10 izq = 1;
		#4 der = 0;
		#4 izq = 0;
		#1 der = 1;
		#7 $finish;
	end
	initial begin
		$dumpfile("pajaro.vcd");
		$dumpvars;
	end
	endmodule
```

En el testbench especificamos con que valores comienza la simulación, clock 1, reset 1, izquierda 1 y derecha 0, de ahí partimos con la simulación.

Aquí podemos apreciar los waves de la simulación, los cuales son los previstos con lo que habíamos visto anteriormente.