INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

DISEÑO DE SISTEMAS DIGITALES

Flores Escobar José Antonio

FSM SEMÁFORO

Integrantes:

Álvarez Hernández Gabriel Alexander

García Quiroz Gustavo Iván

Huesca Laureano José Alejandro

Muñoz Valdivia Irving Omar

Pedroza Villagomez Emir

Introducción.

La implementación de sistemas digitales con lógica secuencial es un pilar fundamental en el diseño de circuitos integrados y sistemas embebidos. Una de las herramientas más potentes para gestionar estos sistemas es la Máquina de Estados Finitos (FSM, por sus siglas en inglés). Las FSM permiten describir comportamientos secuenciales complejos mediante un conjunto de estados y transiciones, simplificando así el diseño y la verificación del sistema.

En este proyecto, se implementa un semáforo utilizando una FSM para gestionar sus estados. Un semáforo es un dispositivo esencial en la gestión del tráfico vehicular, garantizando el flujo ordenado y seguro de vehículos mediante la alternancia de luces verde, amarilla y roja.

Máquina de Estados Finitos (FSM) en el Contexto del Semáforo.

Una FSM es un modelo matemático utilizado para diseñar lógica secuencial. Consiste en un número finito de estados, transiciones entre esos estados, y acciones que se ejecutan en respuesta a las transiciones. En el caso del semáforo, los estados corresponden a las luces (verde, amarillo y rojo), y las transiciones ocurren en función del tiempo que cada luz permanece encendida.

La implementación del semáforo incluye la gestión de estados y transiciones entre ellos, basándose en un reloj de baja frecuencia generado por un divisor de frecuencia. El propósito de este divisor es reducir la frecuencia de un reloj de alta frecuencia a una frecuencia adecuada para controlar las transiciones del semáforo

Desarrollo.

El semáforo se implementa en dos módulos principales: semaforo y DivFreq. El módulo semaforo utiliza una FSM para manejar los estados del semáforo y definir las transiciones entre ellos, mientras que el módulo DivFreq actúa como un divisor de frecuencia para reducir la frecuencia del reloj de entrada, permitiendo que las transiciones del semáforo ocurran a intervalos de tiempo adecuados.

```
1   /*
2      Proyecto: SEMAFORO
3      Archivos: semaforo.v
4      Asignatura: DSD
5      Prof: Flores Escobar Jose Antonio
6      Equipo:  Álvarez Hernández Gabriel Alexander
7               García Quiroz Gustavo Iván
8               Huesca Laureano José Alejandro
9               Muñoz Valdivia Irving Omar
10              Pedroza Villagomez Emir
11  */
12  module semaforo #(
13      parameter E0 = 3'b000, //VERDE
14      parameter E1 = 3'b010, //AMARILLO
15      parameter E2 = 3'b100  // ROJO
16  )(
17      input                           clk_i,
18      input                           rst_ni,
19      input                           x_i,
20      output reg              [2:0] y_o
21  );
22      reg             [2:0] actual_state_w;
23      reg                   transicion_verde;
24      reg             [2:0] cuenta_verde;
25      reg                   transicion_amarilla;
26      reg             [2:0] cuenta_amarilla;
27      reg                   transicion_roja;
28      reg             [2:0] cuenta_roja;
29      // Instancia del divisor de frecuencia
30      wire slow_clk;
31      DivFreq #(
32          .freqdev(10),     // Frecuencia de desarrollo de 10MHz
33          .freqfinal(1000000)     // Frecuencia final dividida en 4
34      ) div_freq_instance (
35          .clk_i(clk_i),
36          .rst_ni(rst_ni),
37          .clk_o(slow_clk)        // Salida de reloj dividido
38      );
39      //Proceso secuencial para calcular estados
40      always @(posedge slow_clk or negedge rst_ni)
```

```
40      always @(posedge slow_clk or negedge rst_ni)
41      begin
42          if (!rst_ni)
43              actual_state_w <= E0;
44          else
45              case(actual_state_w)
46                  E0: if (!transicion_verde) actual_state_w <= E0; else actual_state_w <= E1;
47                  E1: if (!transicion_amarilla) actual_state_w <= E1; else actual_state_w <= E2;
48                  E2: if (!transicion_roja) actual_state_w <= E2; else actual_state_w <= E0;
49              endcase
50      end
51
52      //Proceso para definir las salidas
53      always @(*)
54      begin
55          case(actual_state_w)
56              E0: y_o = 3'b100;
57              E1: y_o = 3'b010;
58              E2: y_o = 3'b001;
59          endcase
60      end
61
62      //Proceso para calcular cambios de estados
63      always @(posedge clk_i or negedge rst_ni)
64      begin
65          if (!rst_ni) //Si estoy en estado reset
66          begin
67              transicion_verde        =   1'b0;
68              cuenta_verde            =   3'b0;
69              transicion_amarilla     =   1'b0;
70              cuenta_amarilla         =   3'b0;
71              transicion_roja         =   1'b0;
72              cuenta_roja             =   3'b0;
73          end
74          else
75          begin
76              case(actual_state_w)
77                  E0: begin
78                          if (cuenta_verde == 4'h4)
```

```
78                          if (cuenta_verde == 4'h4)
79                          begin
80                              transicion_verde    = 1'b1;
81                              cuenta_verde        = 3'b0;
82                          end
83                          else
84                          begin
85                              transicion_verde    = 1'b0;
86                              cuenta_verde        = cuenta_verde + 1'b1;
87                          end
88                      end
89                  E1: begin
90                          if (cuenta_amarilla == 4'h4)
91                          begin
92                              transicion_amarilla = 1'b1;
93                              cuenta_amarilla     = 3'b0;
94                          end
95                          else
96                          begin
97                              transicion_amarilla = 1'b0;
98                              cuenta_amarilla     = cuenta_amarilla + 1'b1;
99                          end
100                     end
101                 E2: begin
102                         if (cuenta_roja == 4'h4)
103                         begin
104                             transicion_roja     = 1'b1;
105                             cuenta_roja         = 3'b0;
106                         end
107                         else
108                         begin
109                             transicion_roja     = 1'b0;
110                             cuenta_roja         = cuenta_roja + 1'b1;
111                         end
112                     end
113             endcase
114         end
115     end
116 endmodule
117 //Reloj pin 12
```

Semáforo.

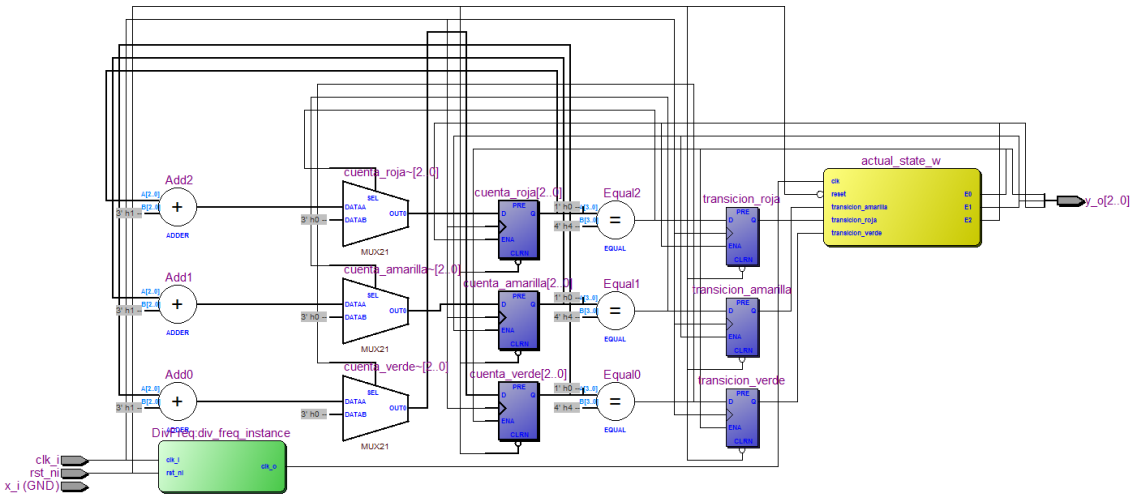
Gestiona los estados de un semáforo (verde, amarillo y rojo) y sus transiciones. Define tres estados (E0 para verde, E1 para amarillo y E2 para rojo) y emplea varios registros para controlar las transiciones entre ellos. Las transiciones son gestionadas por un contador para cada estado, que incrementa en cada ciclo del reloj principal (clk_i). Cuando un contador alcanza el valor predefinido, se activa una señal de transición que cambia el estado del semáforo en el siguiente ciclo del reloj.

```
/*
   Proyecto: SEMAFORO
   Archivos: DivFreq.v
   Asignatura: DSD
   Prof: Flores Escobar Jose Antonio
   Equipo:  Álvarez Hernández Gabriel Alexander
            García Quiroz Gustavo Iván
            Huesca Laureano José Alejandro
            Muñoz Valdivia Irving Omar
            Pedroza Villagomez Emir
*/
module DivFreq #(
    parameter freqdev = 1000, // 10MHz
    parameter freqfinal = 10 // freqdev Dividido en 4
)(
    input clk_i,
    input rst_ni,
    output reg clk_o
);
    reg [31:0] counter_r;

    always @(posedge clk_i or negedge rst_ni) begin
        if (!rst_ni) begin
            counter_r <= 32'b0;
            clk_o <= 1'b1; // Comenzamos con el reloj en bajo
        end else begin
//      if (counter_r >= (freqfinal - 1)) begin
        if (counter_r == (32'd4000)) begin

            counter_r <= 32'b0;
            clk_o <= ~clk_o;
        end else begin
            counter_r <= counter_r + 32'b1;
        end
        end
    end
endmodule
```

DivFreq.

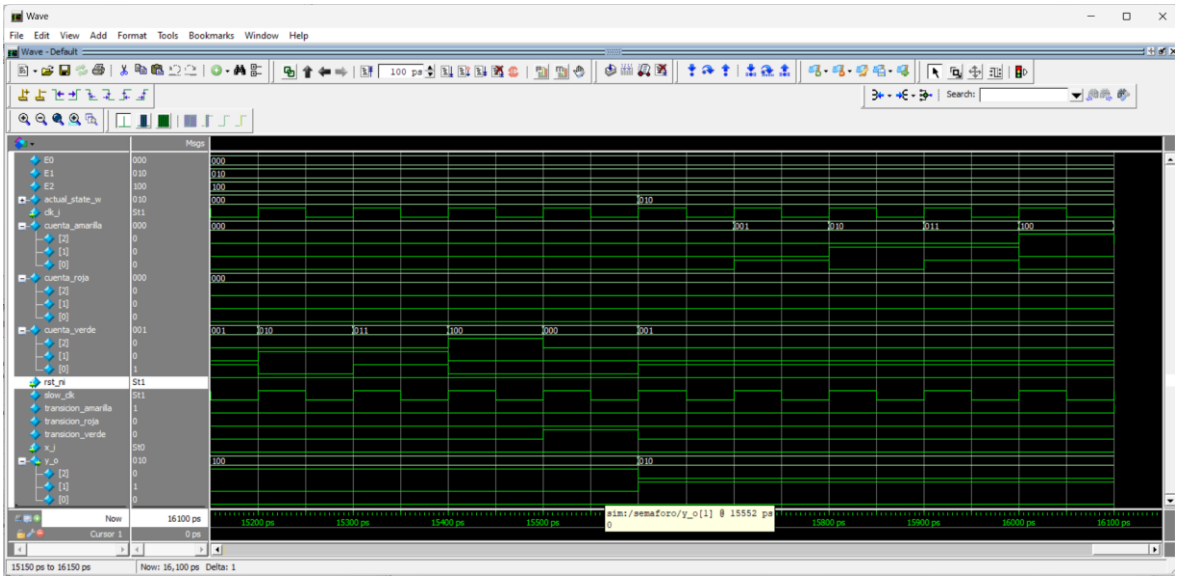
El módulo DivFreq reduce la frecuencia del reloj de entrada (clk_i) a una frecuencia más baja adecuada para controlar el semáforo. Define dos parámetros (freqdev y freqfinal) para establecer la frecuencia de desarrollo y la frecuencia final dividida, respectivamente. El módulo utiliza un registro contador (counter_r) que se incrementa en cada ciclo del reloj de entrada.

.



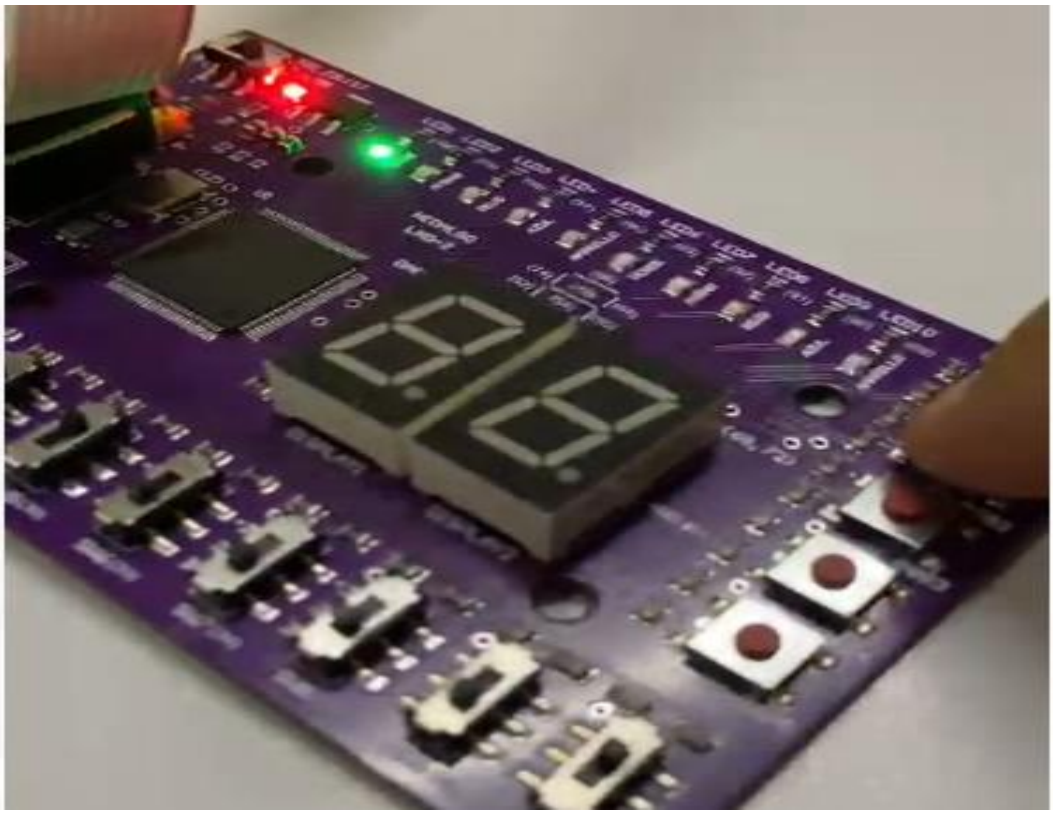
RTLViewer.

Podemos observar como es que se representa con compuertas la maquina de estado que se diseñó en código, así como los componentes necesarios para que funcione nuestro semáforo. Teniendo en cuenta que se utiliza el reloj, reset y el habilitador para su correcto funcionamiento.



Simulación.

En la simulación podemos observar que para cada estado se le asigna un contador, para que después de nuestra cuenta, se cambie de estado, al estado siguiente. Esto aplicándolo a todos los estados, logrando observar como una salida, el semáforo.



Conclusión

La implementación de un semáforo implica la gestión de estados y la división de frecuencia para controlar las transiciones entre los estados de manera efectiva. Este ejemplo muestra cómo utilizar registros, contadores y parámetros para diseñar un sistema de semáforo funcional. La correcta división de la frecuencia del reloj asegura que las luces del semáforo cambien a intervalos adecuados, mejorando la eficiencia en la gestión del tráfico. Este proyecto proporciona una base sólida para aplicaciones más complejas en sistemas y controladores de tráfico.

Bibliografía.

1. Massachusetts Institute of Technology. (2017). *Finite State Machines*. Recuperado de http://web.mit.edu/6.111/www/f2017/handouts/L06.pdf
2. Brown University. (s.f.). *Finite-State Machines and Pushdown Automata*. Recuperado de:dhttps://cs.brown.edu/people/jsavage/book/pdfs/ModelsOfComputation_Chapter4.pdf