Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires

Sistemas Digitales (86.41)

Tp1

Implementación de un sistema secuencial en VHDL

Integrantes:

Niz Silke Jonathan 98722 jniz@fi.uba.ar

1 er cuatrimestre 2022

Enunciado

Implementar un circuito para controlar dos semáforos en un cruce de calles. Dicho circuito tendrá 6 salidas: rojo1, amarillo1, verde1, rojo2, amarillo2,verde2. El tiempo en amarillo será de 3 seg. mientras que en rojo y verde será de 30 segundos. El reloj del sistema tendrá una frecuencia de operación de 50MHz.

1. Introducción

En el siguiente trabajo práctico se implementará un circuito para controlar dos semáforo, tal como lo indica el enunciado. El mismo estará descripto en **VDHL**, formado por 4 módulos. Dichos módulos son : un contador, una maquina de estados, un modulo que integra estos dos anteriores y un **testbench** para poder simularlo. Una vez que este circuito este descripto se procede a su simulación a través de **GTKWAVE** para observar si el comportamiento es correcto. Finalmente una vez comprobado su correcto funcionamiento se procede a sintetizarlo en **Vivado** y la **FPGA** elegida es la xc7a15tftg256-1. A continuación podremos observar el circuito que se desea implementar.

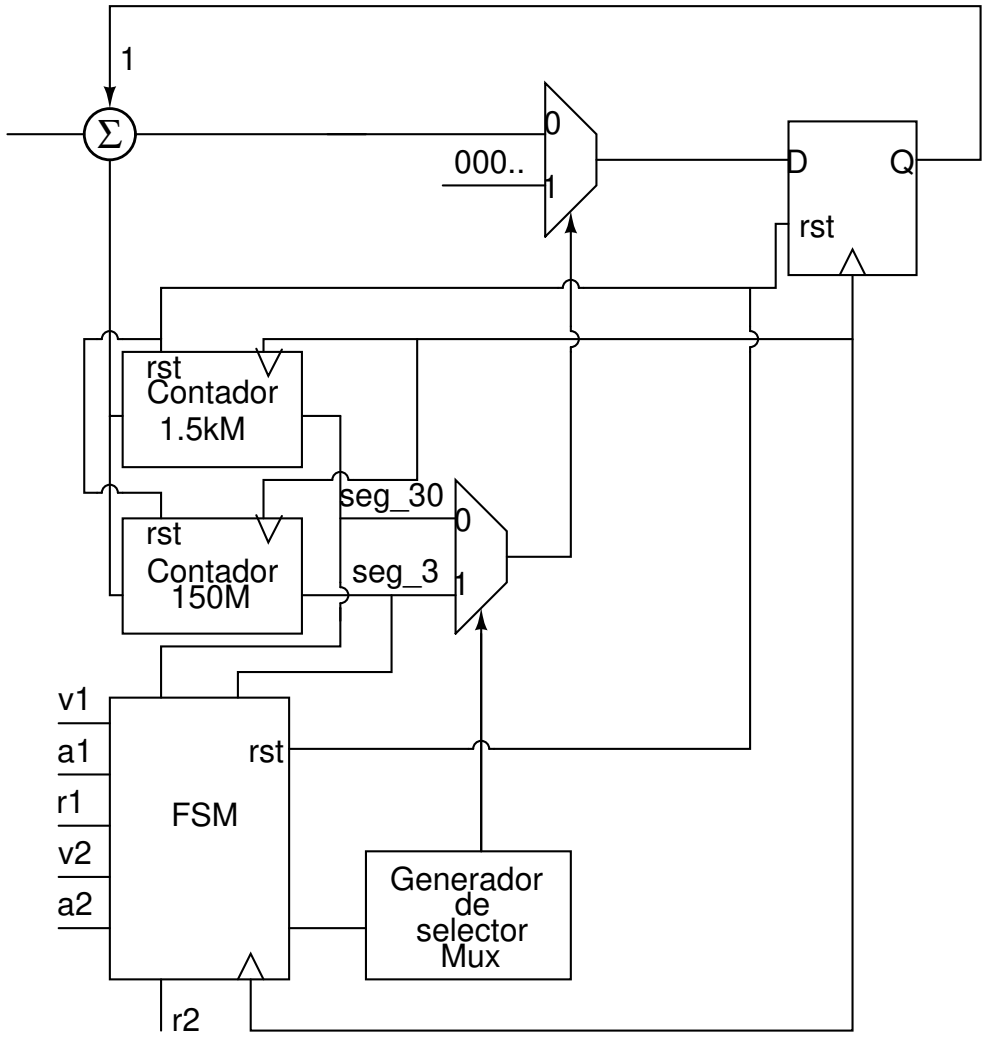

Figura 1: Circuito a implementar

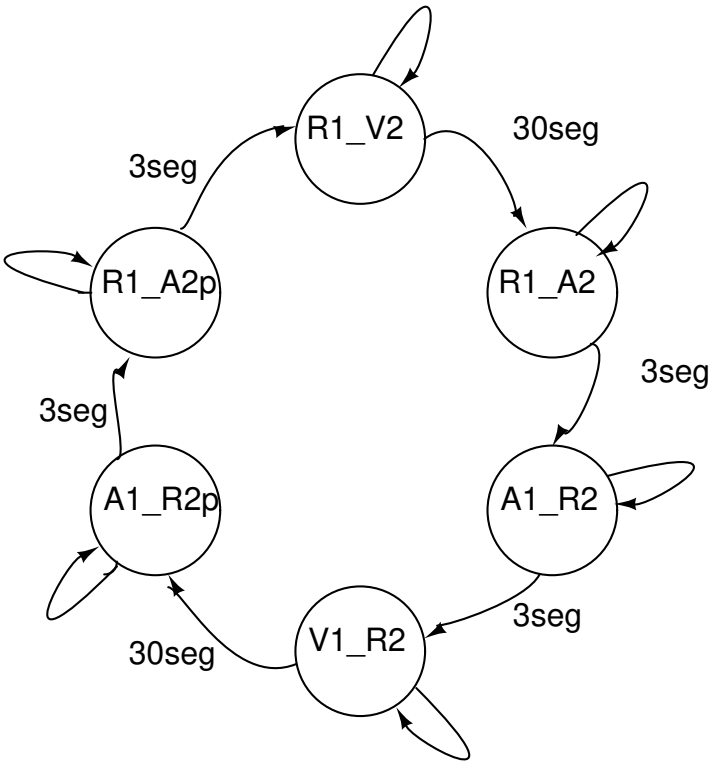

Figura 2: FSM

2. Descripción VHDL

A continuación se explicara el diseño de cada modulo con su respectivo código

2.1. Máquina de estados (fsm_sem)

El módulo de la máquina de estados consiste 4 entradas siendo estas el clock, reset, un indicador de 30 segundos y otro indicador de 3 segundos. Estos indicadores se explicaran luego en el módulo de contador.A su vez posee 7 salidas, 6 de las cuales corresponde a los colores de cada semáforo y la restante representa un selector de un mux.

Al ingresar a esta máquina de estados se verifica antes que nada si esta activo el reset global o no. En caso de que dicho reset este activo y que haya un flanco ascendente se trancisiona al primer estado que en este caso fue elegido rojo 1 - verde 2. Para todos los demás estados se verifica el estado actual y además si esta activo o no el indicador de 3 segundos para los estados que contengan amarillo o 30 segundos para aquellos que tengan verde.

Una vez verificado el estado actual y analizado si corresponde o no la transición, se realiza la asignación al selector mux. Este selector valdrá '1' si el estado actual posee alguno de los dos semáforos en amarillo, y '0' en caso contrario.

```
library IEEE;
use ieee.std_logic_1164.all;


entity fsm_semaforo is
    port(
    clk:in      std_logic;
```

```
    rst:in      std_logic;
    seg_30:in   std_logic;
    seg_3:in    std_logic;
    sem1_v:out  std_logic;
    sem1_a:out  std_logic;
    sem1_r:out  std_logic;
    sem2_v:out  std_logic;
    sem2_a:out  std_logic;
    sem2_r:out  std_logic;
    sel_mux:out std_logic  ---carga para el mux que luego resetea el contador
    );

end fsm_semaforo;

architecture behavioral of fsm_semaforo is

type t_estado is (R1_V2,R1_A2,A1_R2,V1_R2,A1_R2_p,R1_A2_p);
signal estado   :   t_estado;

begin

    process(clk,rst)
    begin
        if rst='1'then
            estado<=R1_V2;
        elsif clk='1'and clk'event then
            case estado is
                when R1_V2 =>
                    if seg_30='1'then
                        estado<=R1_A2;
                    end if;
                when R1_A2 =>
                    if seg_3='1'then
                        estado<=A1_R2;
                    end if;
                when A1_R2 =>
                    if seg_3='1'then
                        estado<=V1_R2;
                    end if;
                when V1_R2 =>
                    if seg_30='1'then
                        estado <= A1_R2_p;
                    end if;
                when A1_R2_p =>
                    if seg_3='1'then
                        estado<=R1_A2_p;
                    end if;
                when R1_A2_p =>
```

```
                    if seg_3='1'then
                        estado<=R1_V2;
                    end if;
                end case;
            end if;
        end process;
sel_mux <=  '1' when ((estado = R1_A2) OR
(estado = A1_R2) OR (estado = A1_R2_p) OR
(estado = R1_A2_p)) else
            '0';


sem1_v <=   '1' when estado = V1_R2 else
            '0';
sem1_a <=   '1' when ((estado = A1_R2) OR (estado = A1_R2_p)) else
            '0';
sem1_r <=   '1' when ((estado = R1_V2) OR
(estado = R1_A2)  OR
(estado = R1_A2_p)) else
            '0';

sem2_v <=   '1' when estado = R1_V2 else
            '0';
sem2_a <=   '1' when ((estado = R1_A2) OR (estado =  R1_A2_p)) else
            '0';
sem2_r <=   '1' when ((estado = V1_R2) OR
(estado = A1_R2) OR
(estado = A1_R2_p)) else
            '0';
end behavioral;
```

2.2. Contador (contadorN32)

Este modulo posee 4 entradas, las cuales son reset, clock, value y load, y dos salidas : count y seg_stop.La salida seg_stop es un vector de dos posiciones, donde la primer posición corresponde al indicador de 3 segundos y la segunda posición al indicador de 30 segundos.Cuando el contador llega a los 3 segundos se carga el valor 01 a seg_stop y cuando llega a los 30 segundos se carga el valor ”10”. En caso de que no este en ninguna de las dos situaciones su valor es ”00”. Esta salida es la que luego alimenta a la maquina de estados para indicarle si paso por ejemplo 30 segundos desde que algún estado que contenga verde fue seleccionado para así transicionar al siguiente. La entrada load indica si se debe reiniciar o no la cuenta con el valor cargado en value.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity contadorN_32 is
    generic(N:natural:=31);
```

```
    port    (
        rst :       in  std_logic;
        clk :       in  std_logic;
        -- la pos 0 es para la cuenta 3 seg pos 1 para 30 seg
        seg_stop:   out std_logic_vector(1 downto 0);
        value:      in  std_logic_vector(N-1 downto 0);
        load:       in  std_logic;
        count:      out std_logic_vector(N-1 downto 0)
    );
end contadorN_32;

architecture    behavioral  of  contadorN_32    is
    constant seg_30:natural:=149; --1.5kM
    constant seg_3:natural:=14; --150M
    signal  aux_count   :   unsigned(N-1 downto 0);
begin
    process(clk,rst)
    begin
        if  rst='1' then
            aux_count <= (others => '0');
        elsif   clk='1' and clk'event then
            if load = '1' then
                aux_count <= unsigned(value);
            else
                aux_count <= aux_count + 1;
            end if;

        end if;
    end process;
    --matcheo contra 30 seg o 3 seg.
    --Esto entra a la fsm para verificar si debe pasar o no de estado
    seg_stop <= "10"    when seg_30 = aux_count else
                "01"    when seg_3 = aux_count  else
                "00";

    count <= std_logic_vector(aux_count);

end behavioral;
```

2.3. Semáforo

Este módulo esta compuesto por los dos módulos ya explicados previamente.Posee dos entradas (clock y reset), y las seis salidas de los semáforos. En este modulo se carga el valor de load con los valores 1 para reinicio del contador o 0 para que siga contando. La condición de reinicio se da en los escenarios que el contador llego a los 30 segundos, lo cual se puede verificar por la salida seg_stop del modulo fsm_sem o cuando se llega a los 3 segundos y el valor de selector del mux se encuentra en '1', es decir al menos 1 semaforo esta en amarillo.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity semaforo is
    port(
        clk:in          std_logic;
        rst:in          std_logic;
        osem1_v: out     std_logic;
        osem1_a: out     std_logic;
        osem1_r: out     std_logic;
        osem2_v: out     std_logic;
        osem2_a: out     std_logic;
        osem2_r: out     std_logic
    );
end semaforo;

architecture behavioral of semaforo is
    constant    Ncount:integer:=31;
    signal      count:      std_logic_vector(Ncount-1 downto 0);
    signal      init_value: std_logic_vector(Ncount-1 downto 0);
    signal      load:       std_logic;
    signal      seg_stop:   std_logic_vector(1 downto 0);
    signal      sem1_v:     std_logic;
    signal      sem1_a:     std_logic;
    signal      sem1_r:     std_logic;
    signal      sem2_v:     std_logic;
    signal      sem2_a:     std_logic;
    signal      sem2_r:     std_logic;
    signal      sel_mux:    std_logic;
begin

fsm_sem:        entity work.fsm_semaforo
                port map(
                clk => clk,rst => rst,
                seg_30 => seg_stop(1),
                seg_3 => seg_stop(0),
                sem1_v => sem1_v,sem1_a => sem1_a,
                sem1_r => sem1_r,
                sem2_v => sem2_v,sem2_a => sem2_a,
                sem2_r => sem2_r,sel_mux => sel_mux);

contadorN_32:   entity work.contadorN_32
                port map(clk => clk,rst => rst,
                seg_stop => seg_stop,
                value => init_value,
                load=>load,count => count);
```

```
init_value <= (others=>'0');

load <= '1' when  ((seg_stop(1) = '1') OR
((seg_stop(0) = '1') AND (sel_mux ='1'))) else
       '0';
osem1_v <= sem1_v;
osem1_a <= sem1_a;
osem1_r <= sem1_r;
osem2_v <= sem2_v;
osem2_a <= sem2_a;
osem2_r <= sem2_r;

end;
```

2.4. Testbench

Este módulo es creado para poder realizar la simulación del circuito descripto. Las entradas son clock/reset y las salidas los valores de los semáforos.

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_semaforo is
end tb_semaforo;

architecture behavioral of tb_semaforo is
    signal tb_rst: std_logic;
    signal tb_clk: std_logic:='0';
    signal      sem1_v:     std_logic;
    signal      sem1_a:     std_logic;
    signal      sem1_r:     std_logic;
    signal      sem2_v:     std_logic;
    signal      sem2_a:     std_logic;
    signal      sem2_r:     std_logic;



begin
    tb_rst <= '1','0' after 10 ns;
    tb_clk <= not tb_clk after 10 ns;

    semaforo_inst: entity work.semaforo
      port map (
        clk     => tb_clk,
        rst     => tb_rst,
        osem1_v => sem1_v,
        osem1_a => sem1_a,
        osem1_r => sem1_r,
        osem2_v => sem2_v,
```

```
        osem2_a => sem2_a,
        osem2_r => sem2_r
      );
end behavioral;
```

3. Simulación

En esta sección observaremos los resultados de la simulación del Testbench. Hay que tener en cuenta que para poder simular todas las transiciones el contador tiene que llegar a un valor de 1500.000.000 ya que la frecuencia del clock pedida es de **50 Mhz** y la máxima espera a realizar es de 30 segundos. Debido al tamaño de este numero simularlo es muy difícil por lo que se toma para los 30 segundos 149 flancos ascendentes de clock y 14 flancos ascendentes de clock para los 3 segundos.

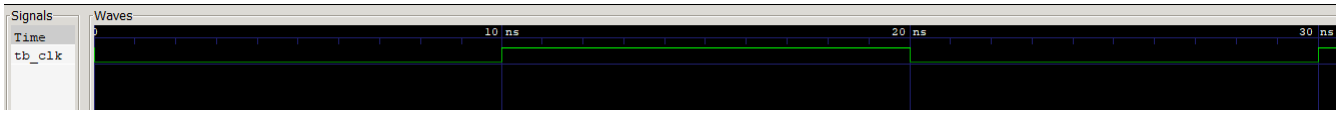

Figura 3: Simulación de clock de testbench. Periodo 20ns

A continuación podemos observar que al estar en el valor 148 estamos en el estado inicial con el semaforo 1 en rojo y el semaforo 2 en verde. Tanto el vector seg_stop (explicado previamente) como el valor de load están en 0.

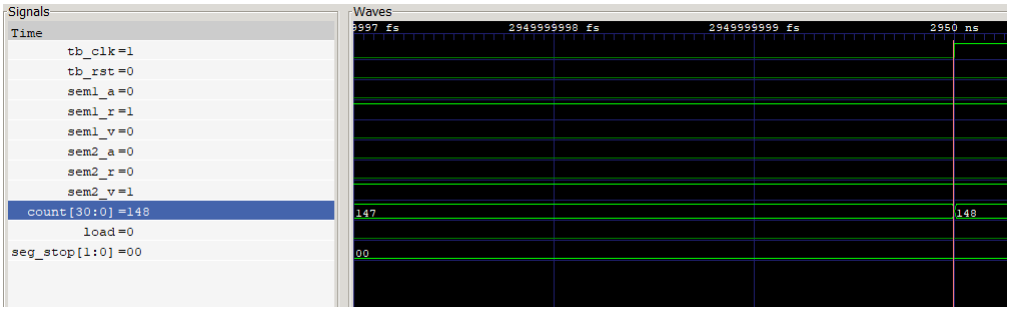

Figura 4: Contador previo a transición

Luego al llegar al valor 149 se modifica el valor de load y el valor de seg_stop indicando que llego a los 30 segundos, o lo equivalente en este caso.

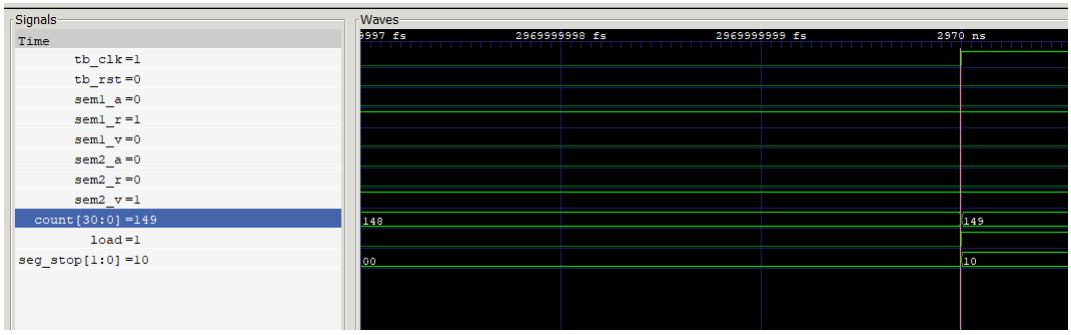

Figura 5: Contador en transición

Pasado los 30 segundos el contador se debe *resetear* y tiene que realizar la transición al siguiente estado, que en este caso es el semaforo 1 en rojo y el semaforo 2 en amarillo, tal como se puede observar en la imagen a continuación.

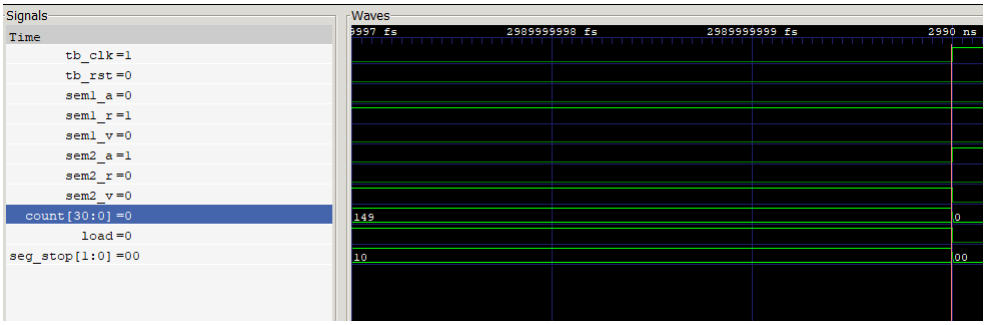

Figura 6: Contador reseteado y en nuevo estado

Al llegar a los 3 segundos en un estado que contiene amarillo en alguno del los semáforos se debe reiniciar , esto se hace con la carga en 1 y el indicador de 3 segundos.

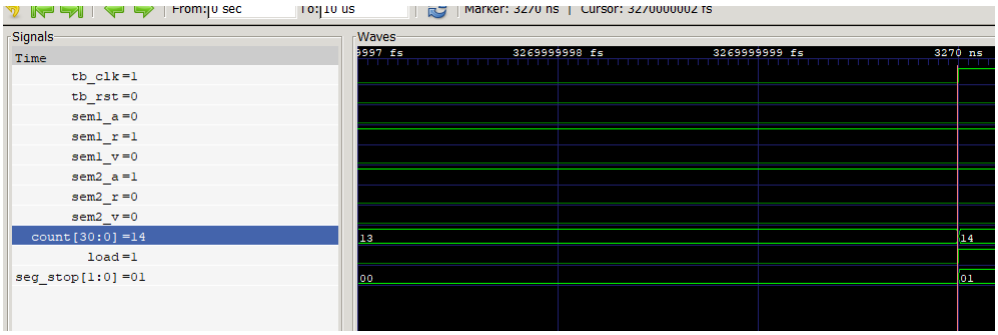

Figura 7: Contador en transición 3 segundos

Luego de el estado anterior que se puede apreciar en la imagen previa se llega al siguiente estado de amarillo que es semaforo 1 en amarillo y semaforo 2 en rojo.

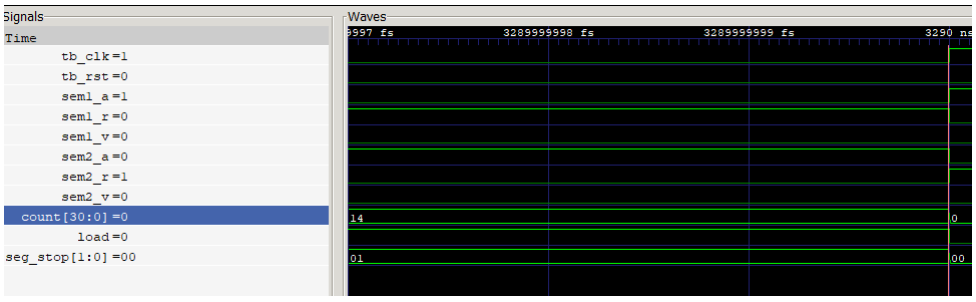

Figura 8: Contador reseteado y en nuevo estado 3 segundos

4. Síntesis

En este apartado veremos el resultado de la síntesis. Esta síntesis se realiza sobre una FPGA xc7a15tftg256-1.

En primera instancia podemos observar en la siguiente imagen el RTL del circuito el cual coincide en gran escala con lo esperado.

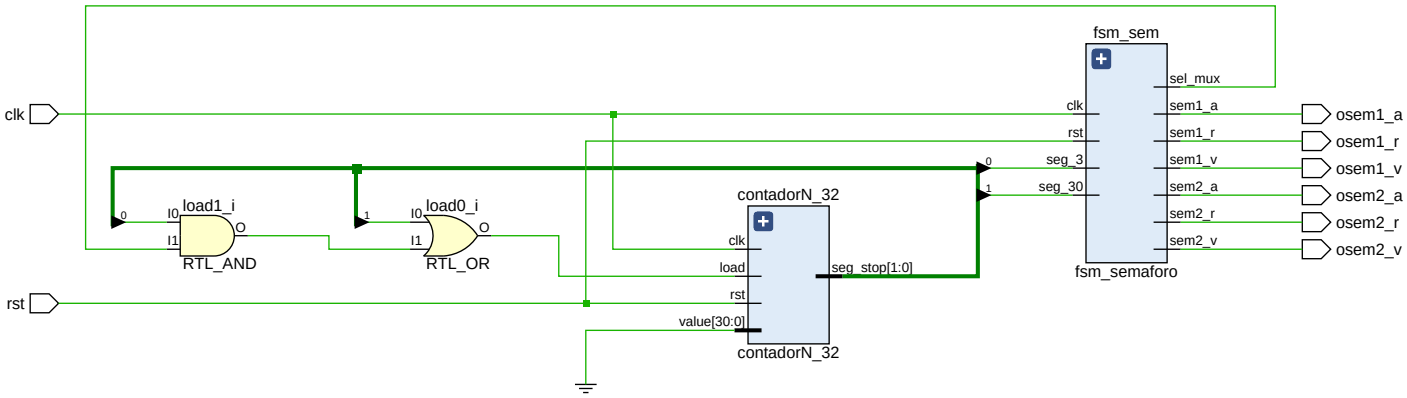

Figura 9: RTL del circuito completo

Para el contador a su vez se puede apreciar que su RTL es acorde a lo previamente diseñado

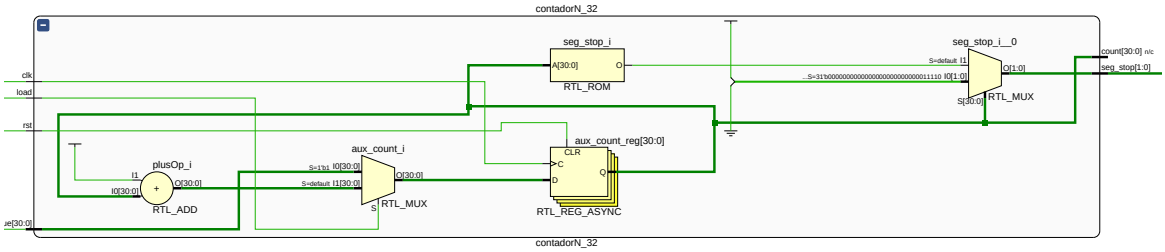

Figura 10: RTL Contador

Luego en la siguiente imagen observamos el RTL de la maquina de estados la cual por cuestiones lógicas no se asemeja a su diseño inicial, ya que el diseño previamente mostrado es una abstracción.

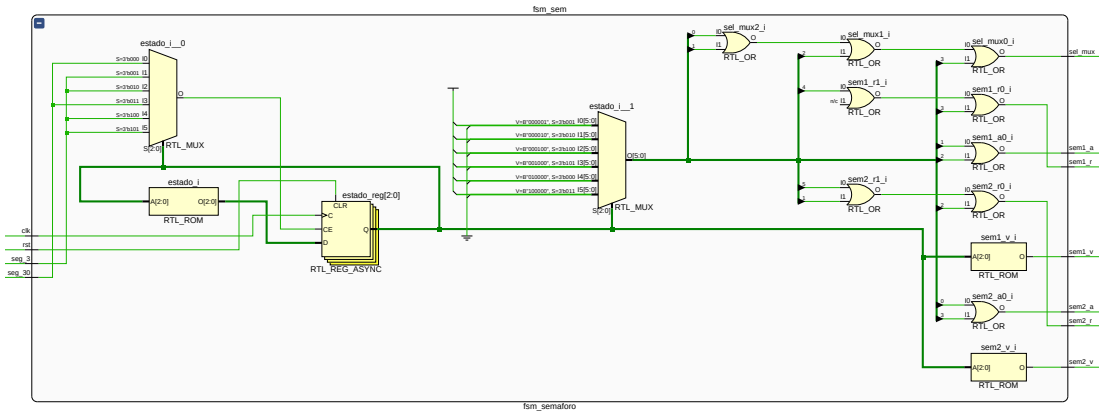

Figura 11: RTL FSM

Ahora pasaremos a observar la implementación del circuito en la FPGA.

Siguiendo la misma estructura que antes , ahora observamos la implementación de forma completa. Aquí ademas de observar las entradas y salidas podemos observar los *buffers* de entrada y salida que tiene la FPGA.

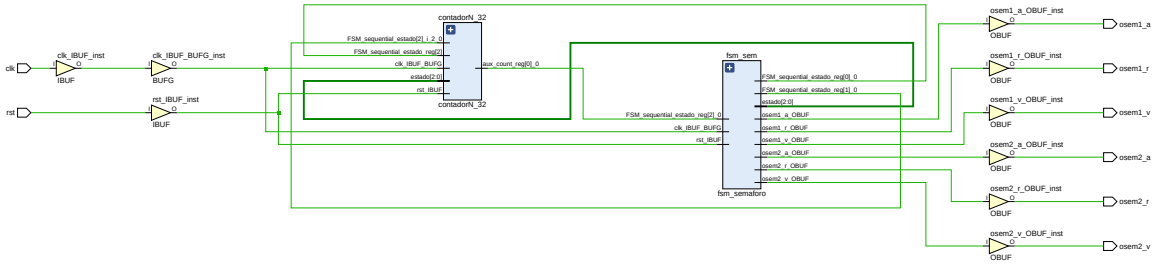

Figura 12: Implementación del Circuito completo

.

Lo siguiente que podemos observar es una parte de la implementación del contador, donde por ejemplo vemos los FF con los que se trabaja y las LUTS. Por cuestiones del tamaño de la imagen no se incluye el contador completo.

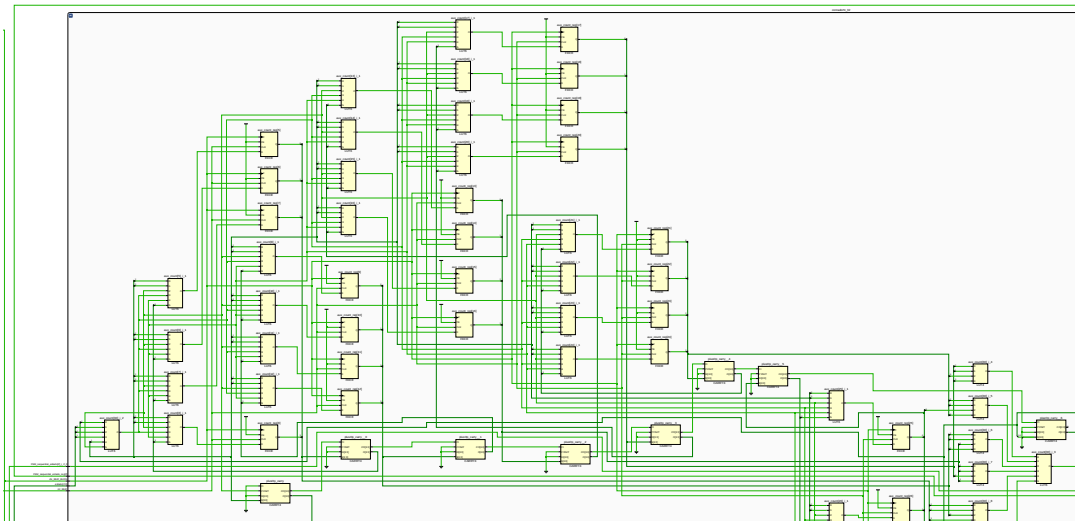

Figura 13: Parte de la implementación del Contador

.

Finalmente al revisar el análisis que nos brinda **Vivado** se observa que los FF utilizados son 34 y las LUTS 48.

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF
✓ synth_1	constrs_1	synth_design Complete!									48	34
✓ impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA		NA	0.844	0	48	34

Figura 14: Tabla

.

Conclusión

A lo largo de este primer trabajo se pudo afianzar los conocimientos teóricos de VHDL plasmándolo en un pequeño proyecto, siendo este sintetizado en Vivado. Además se puede apreciar la importancia de la modularización de componentes ya que por ejemplo el contador puede ser utilizado para cualquier otro proyecto con pequeñas modificaciones.