**Universidad Autónoma de Baja California**
**Facultad de Ciencias Químicas e Ingeniería**
**Ingeniería en computación**


**DISEÑO DIGITAL**

Proyecto final

# Crucero vial

**GRUPO:** 551

**MAESTRA:** Carelia Guadalupe Gaxiola Pacheco

**AUTOR:** Carrasco Vega Alondra (1270851)

Rico López Tamara Illian (1270673)

**7-diciembre-2021**

# Crucero vial

## Resumen

El propósito de este proyecto es el diseño de un crucero vial a partir del uso de los conceptos de diseño digital y el uso del lenguaje de descripción de hardware. El diseño final involucra todas las estructuras básicas que el lenguaje proporciona y cumple con éxito las funciones estipuladas por el problema, logrando reducir el uso de componentes a dos dispositivos programables GAL. El enfoque del diseño está basado en los modelos de máquinas de estados con la diferencia de que se ajusta al uso de sensores como señales de entrada. A partir de nuestro diseño final, resolvemos el problema de la sincronización entre los dos semáforos que se desea controlar utilizando la reducción de los estados hasta llegar a un sistema sencillo con la complejidad suficiente para resolver el problema. Finalmente, resolvemos el problema de una forma compacta y elegante que permite ver los resultados de forma clara y concisa.

# Crucero vial

## Introducción

En base a distintos temas vistos a lo largo del curso de *Circuitos Digitales,* como circuitos secuenciales, maquinas de estado, lenguaje de descripción de hardware, se solicitó realizar un proyecto en el cual se pudieran aplicar estos conocimientos adquiridos durante el curso, por lo cual se pide diseñar un crucero vial que cumpliera con lo que se describe a continuación.

Se tiene el caso de una autopista principal con un camino de acceso secundario, donde el camino principal es la carretera E-O y el camino de acceso es la carretera N-S. En este mismo crucero vial se colocaron sensores de detección de vehículos a lo largo de los carriles C y D (camino principal) y de los carriles A y B (camino de acceso). Las salidas de estos sensores están en BAJO (0) cuando no hay vehículos presentes, y en ALTO (1) cuando hay vehículos presentes.

El semáforo de la intersección debe controlarse de acuerdo con la siguiente lógica:

1. El semáforo E-O se mantendrá en verde cuando no haya vehículos presentes.
2. El semáforo E-O estará en verde cada vez que C o D estén ocupados, mientras no haya vehículos presentes en A y B.
3. El semáforo E-O se mantendrá en verde por 20 segundos cada vez que C y D estén ambos ocupados, mientras ambos o alguno de los carriles A y B estén ocupados. Cambiará a rojo por 5 segundos y después regresará a verde.
4. El semáforo norte-sur (N-S) se pondrá en verde por 10 segundos cuando ambos carriles A y B estén ocupados, pero cuando C y D no estén ambos ocupados.
5. El semáforo E-O estará en verde por 20 segundos cada vez que C o D estén ocupados, mientras A y B no estén ambos ocupados. Cambiará a rojo por 5 segundos a rojo y después regresará a verde.
6. El semáforo N-S también se pondrá en verde cuando A o B estén ocupados, mientras que C y D estén ambos vacantes.

Considerando las salidas de los sensores A, B, C y D como entradas, se diseñó un circuito lógico secuencial utilizando dispositivos lógicos programables y lenguajes de descripción de hardware para controlar los dos semáforos, N-S y E-O, los cuales deben contar con luz verde, amarilla y roja cada uno.

## Objetivos

El objetivo del desarrollo del proyecto es aplicar los conocimientos adquiridos a lo largo de la clase acerca del diseño de sistemas digitales usando los recursos electrónicos a nuestra disposición. El proyecto consiste en el diseño de un crucero vial usando el lenguaje de descripción de hardware (VHDL) así como simuladores de circuitos para poner a prueba el resultado final.

# Teoría

## Circuitos secuenciales

Un circuito secuencial es un tipo de circuito lógico. Envía una salida que depende de la entrada actual, además de la historia de la entrada. Eso lo hace diferente de un circuito combinacional, cuya salida depende únicamente de la entrada actual. Dada su dependencia de la historia de entrada (o entrada almacenada), los circuitos secuenciales son particularmente útiles en la memoria de acceso aleatorio (RAM, por sus siglas en inglés) de la computadora. Los dos tipos de circuitos secuenciales son circuitos síncronos y circuitos asíncronos.

El estado de un sistema secuencial viene dado por el contenido de sus elementos de memoria. Es frecuente que en los sistemas secuenciales exista una señal que inicia los elementos de memoria con un valor determinado: señal de inicio (reset). La señal de inicio determina el estado del sistema en el momento del arranque (normalmente pone toda la memoria a cero).

### Circuitos síncronos

Los circuitos síncronos hacen uso de flip-flops y compuertas lógicas. Similar a los flip-flops, las compuertas lógicas permiten o restringen el flujo de información en función de determinadas situaciones, tales como si dos variables son iguales una a otra. Un tipo específico de circuito síncrono, llamado circuito cronometrado secuencial, utiliza pulsos de reloj. Los pulsos de reloj se utilizan para sincronizar los diferentes elementos del circuito, tales como los flip-flops. Los flip-flops en un circuito cronometrado secuencial sólo se ven afectados cuando se les indica mediante un pulso de reloj.

### Circuitos asíncronos

Un circuito asíncrono es el que registra el orden en el que cambian sus variables de entrada, y envía una salida que depende del resultado. Este tipo de circuito también debe ser capaz de cambiar sus variables de entrada en cualquier momento. Hay también un tipo específico de circuito asíncrono, denominado circuito asíncrono tipo compuerta. Los circuitos tipo compuerta son circuitos combinacionales esencialmente (es decir, que se basan únicamente en la entrada actual) con un camino de realimentación. El camino de realimentación significa que la información de la salida puede ser realimentada en la entrada. Debido a la retroalimentación, estos tipos de circuitos pueden ser inestables, por lo que no se usan comúnmente.

## Lenguaje de descripción de hardware

Los lenguajes de descripción de hardware (HDLs) son utilizados para describir la arquitectura y comportamiento de un sistema electrónico los cuales fueron desarrollados para trabajar con diseños complejos.

Comparando un HDL con los lenguajes para el desarrollo de software vemos que en un lenguaje de este tipo un programa que se encuentra en un lenguaje de alto nivel (VHDL) necesita ser ensamblado a código máquina (compuertas y conexiones) para poder ser

interpretado por el procesador. De igual manera, el objetivo de un HDL es describir un circuito mediante un conjunto de instrucciones de alto nivel de abstracción para que el programa de síntesis genere (ensamble) un circuito que pueda ser implementado físicamente.

Un circuito hecho mediante una descripción en un HDL puede ser utilizado en cualquier tipo de dispositivo programable capaz de soportar la densidad del diseño. Es decir, no es necesario adecuar el circuito a cada dispositivo porque las herramientas de síntesis se encargan de ello.

### VHDL

VHDL es un lenguaje de descripción de hardware utilizado para describir circuitos en un alto nivel de abstracción el cual está siendo rápidamente aceptado como un medio estándar de diseño. VHDL es producto del programa Very High Speed Integrated Circuit (VHSIC) desarrollado por el Departamento de Defensa de los Estados Unidos a finales de la década de los 70 's. El propósito era hacer un estándar para diseñar, modelar, y documentar circuitos complejos de tal manera que un diseño desarrollado por una empresa pudiera ser entendido por otra y, además, pudiera ser procesado por software con propósitos de simulación.

VHDL divide los circuitos en dos “vistas” entidades y arquitecturas. La entidad modela al circuito, componente o sistema externamente definiendo a este mediante un nombre y sus conexiones que vienen siendo las entradas y salidas del circuito. En tanto que la arquitectura, que es la vista interna, define el funcionamiento del circuito.

### PLDs

Son circuitos integrados que ofrecen a los diseñadores en un solo chip, un arreglo de compuertas lógicas y flip-flops, que pueden ser programados por el usuario para implementar funciones lógicas y así, una manera más sencilla de reemplazar varios circuitos integrados estándares o de funciones fijas. Lo cual hace más práctico y más funcional su diseño ya que la cantidad de errores es menor por la cantidad de piezas que se ahorran al instalarlo.

Existen cuatro tipos de dispositivos que se clasifican como PLD.

- PROM (Programmable Read Only Memory). Memoria programable de sólo lectura.
- PLA (Programmable Logic Array). Matriz lógica programable.
- PAL (Programmable Array Logic). Matriz lógica programable.
- GAL (Generic Array Logic). Matriz lógica genérica.

## Metodología

El proyecto consiste en el desarrollo de un crucero vial, con dos semáforos que funcionan a través de cuatro sensores para identificar si existen autos esperando o si, de lo contrario, los carriles se encuentran vacíos. El diseño realizado para resolver el problema planteado fue desarrollado utilizando la herramienta de software de ispLever y, como se había mencionado anteriormente, se hizo en VHDL.

Para comenzar con la solución del problema, se tradujeron todos los enunciados referentes al problema a la siguiente tabla de estados, donde cada número representa uno de los estados. Principalmente, se consideran los sensores como las entradas donde alto representa un automóvil presente y baja significa que no hay autos; asimismo, se cuenta con dos salidas, que son el semáforo de este a oeste (EO) y el semáforo norte a sur (NS).

| **ESTADO** **(estado equivalente)** | **Sensores** | | | | **Semáforos** | |
|---|---|---|---|---|---|---|
| | **C** | **D** | **A** | **B** | **EO** | **NS** |
| 1 | 0 | 0 | 0 | 0 | verde | rojo |
| 6 | 0 | 0 | 0 | 1 | rojo | verde |
| 6 | 0 | 0 | 1 | 0 | rojo | verde |
| 6 | 0 | 0 | 1 | 1 | rojo | verde |
| 2 (1) | 0 | 1 | 0 | 0 | verde | rojo |
| 5 (3) | 0 | 1 | 0 | 1 | verde 20 | verde 5 |
| 5 (3) | 0 | 1 | 1 | 0 | verde 20 | verde 5 |
| 4 | 0 | 1 | 1 | 1 | verde 20 | verde 10 |
| 2 (1) | 1 | 0 | 0 | 0 | verde | rojo |
| 5 (3) | 1 | 0 | 0 | 1 | verde 20 | verde 5 |
| 5 (3) | 1 | 0 | 1 | 0 | verde 20 | verde 5 |
| 4 | 1 | 0 | 1 | 1 | verde 20 | verde 10 |
| 2 (1) | 1 | 1 | 0 | 0 | verde | rojo |
| 3 | 1 | 1 | 0 | 1 | verde 20 | verde 5 |
| 3 | 1 | 1 | 1 | 0 | verde 20 | verde 5 |
| 3 | 1 | 1 | 1 | 1 | verde 20 | verde 5 |

Tabla 1. Tabla de estados.

Como se puede observar en la tabla, hay ciertos tiempos a considerar y ciertos estados similares, por lo que se decidió trabajar únicamente con cuatro estados. De la misma manera, en la tabla 1 sólo se muestran los colores verde y rojo pero es esencial para el funcionamiento correcto del semáforo incluir el color amarillo, cuando la situación lo requiera.

## Metodología

Para comenzar, el proyecto se desarrolló en dispositivos GAL22V10 y, al inicio, se intentó utilizar un sólo dispositivo; no obstante, esto no fue posible debido al número de puertos y se decidió hacer el proyecto en dos dispositivos diferentes, uno para los semáforos y otro para el contador a cargo de manejar los tiempos en cada estado.

Al tener la oportunidad de desarrollar el proyecto utilizando un lenguaje de programación, el análisis y la ejecución se tradujeron en estructuras de control. Para comenzar, el manejo de los cambios en los semáforos se ve afectado por dos factores, los cuales son las entradas de los sensores y el estado anterior del semáforo, ya que se debe incluir el color amarillo.

En el primer diseño se comenzó planteando la posibilidad de realizar una máquina de estados cíclica donde los cambios serían afectados por las transiciones de estado a estado siguiendo el orden establecido por la tabla 1; sin embargo, este enfoque fue descartado debido a las implicaciones técnicas que ocasionaba al tener más de dos controles manejando las transiciones del semáforo.

Después se decidió utilizar las entradas de los sensores como la forma principal de realizar las transiciones, donde se utilizaba una estructura *case* para manejar los distintos casos que podrían ocurrir en el semáforo principal y éste, determinaría el estado del semáforo NS a partir de variables de control. En este caso, ambos semáforos se encontraban en dispositivos diferentes y, se necesitaba de un dispositivo adicional para el contador, lo cual requería de tres dispositivos GAL ocasionando ciertos retrasos en los cambios de estado de los semáforos así como ciertas discrepancias entre el resultado esperado y el resultado obtenido. Esta propuesta fue perfeccionada en diseños posteriores manteniendo la misma estructura y solucionando los detalles en este modelo.

En otro enfoque, se decidió cambiar todas las señales que el código había estado manejando en un intento por reducir el uso de los *pines* del dispositivo por parte de estas señales que no eran requeridas como entradas o salidas del componente hacia el exterior. El resultado fue similar al anterior, con la diferencia de que ambos semáforos contaban con su respectivo dispositivo que recibía las entradas de los sensores y realizaban los cambios de acuerdo a los establecido en la tabla de estados; sin embargo, este diseño continuo con el mismo problema de sincronización a pesar de contar con la misma señal de reloj y contador.

Otra de los acercamientos al problema fue el retroalimentar las salidas del semáforo NS al dispositivo con el semáforo EO, donde éste último realizaba sus respectivas transiciones de acuerdo a los sensores y al último estado del semáforo NS. De la misma manera, el semáforo principal se encargaba de manejar los cambios en el semáforo NS a partir de variables de control. Este diseño no presentó problemas mayores y fue capaz de resolver el problema de la sincronización entre ambos semáforos, así como el ajuste en los tiempos en que ambos semáforos trabajaban en conjunto.

Por último, el diseño final es una combinación de cada uno de los diseños anteriores, tomando en cuenta lo aprendido de cada uno de ellos mientras se solucionaban los problemas menores que dichos diseños provocaban. El proyecto final es el último diseño realizado, con la diferencia de que ambos semáforos pudieron incluirse en un sólo dispositivo GAL, permitiendo un mejor funcionamiento y comunicación entre las salidas así como el diseño más sencillo y económico que representa.

## Funcionamiento

Para el diseño final de los semáforos se necesitó de las entradas de los sensores a formar de un vector de cuatro elementos, una señal de reloj y las entradas correspondientes al contador de tres bits. De la misma manera, las salidas que resultaron fueron los dos semáforos conformados por dos vectores de tres bits y una salida lógica *reset* para reiniciar el contador. En la figura 1 se muestra como se declaran estas características, al establecer que se les asignará uno de los puertos del dispositivo PLD.

```
entity semaforo is
port(
  sensores:in std_logic_vector(3 downto 0);--sensores
  clk: in std_logic;
  count: in std_logic_vector(0 to 2);
  reset: out std_logic;
  NS: out std_logic_vector(0 to 2);--salida semaforo NS
  EO: out std_logic_vector(0 to 2));--salida semaforo EO
end semaforo;
```

Figura 1. Declaración de los puertos de la entidad “semáforo”.

El primer enfoque para la resolución del problema respecto a los semáforos, fue el uso de una máquina de estados, que comenzará en el estado uno y el siguiente fuera dado por la tabla de estado. Sin embargo, debido a esta observación de que hay dos factores diferentes, la propuesta cambió. Se utilizó una estructura *case*, para manejar los cambios de estados generados por la señales de los sensores así como otra estructura *case* generada dentro de cada caso anterior para controlar los cambios entre los colores del semáforo.

```
when "1111"|"1101"|"1110"|"0101"|"0110"|"1001"|"1010" =>
      case EO is
              when "001" =>-- si EO es verde
                      EO <= "001"; NS <= "100";
                      reset <= '0';
                      if(count = "100") then --si pasaron 20 segundos cambia a amarillo
                              EO <= "010"; NS <= "100";
                              reset <= '1';
                      end if;
              when "010" => --si EO es amarillo
                      EO <= "100"; NS <= "001";
                      reset <= '0';
              when "100"=>--si  EO es rojo
                      if(count < "001") then
                              EO <= "100"; NS <= "001";
                              reset <= '0';
                      elsif(count = "001")then --si pasaron 5 segundos cambia a amarillo
                              EO <= "100"; NS <= "010";
                              reset <= '0';
                      else
                              EO <= "001"; NS <= "100";
                              reset <= '1';
                      end if;
      end case;
```

Figura 2. Representación de la estructura utilizada en el código.

En la figura 2 se muestra un fragmento de código en el que se emplea esta estructura donde el control principal es manejado por la entrada de los sensores y, por lo tanto, provoca el cambio principal en el comportamiento del semáforo. Sin embargo, la segunda estructura *case* se encuentra controlada por el estado anterior del semáforo principal, denominado EO (este-oeste); de la misma forma, esta estructura controla la salida del semáforo EO así como la del semáforo NS (norte-sur), ya que este segundo semáforo es dependiente del principal.

Por otro lado, en la figura 2 también se muestra como se establecen los tiempos respectivos para cada color del semáforo, a través de estructuras condicionales *if/else* donde se establecen

los segundos que debe durar en cada color y, después se manda la señal de *reset* al contador, indicando que inicie en cero y vuelva a contar hasta llegar al tiempo nuevamente establecido. Para el caso del semáforo EO, cuando está en verde y pasaron 4 ciclos de reloj, cambia a amarillo por un ciclo de reloj y despues a rojo hasta que se cumplan los ciclos de reloj en verde establecidos por el semáforo NS dependiendo del caso (ya sea 1 o 2 ciclos de reloj), despues este último pasa a amarillo durante otro ciclo de reloj y luego el semáforo EO vuelve a pasar a verde y el semáforo NS pasa a rojo.

```
case sensores is

when "0000"|"0100"|"1000"|"1100" =>
      if(NS = "001") then -- si el semaforo NS no esta en rojo
              EO <= "100"; NS <= "010";
              reset <= '1';
      else
              EO <= "001"; NS <= "100";
              reset <= '1';
      end if;
```

Figura 3. Representación de una estructura diferente dentro del código.

No obstante, en la figura 3 se muestra una situación diferente a la anterior donde, a partir del enunciado 1 y 2 del problema, se infiere que el comportamiento del semáforo en dicho caso es similar a estático en lugar de dinámico; por lo que una segunda estructura *case* no es necesaria. Por ello, se realiza el control del semáforo a partir de una estructura condicional *if/else* donde, si el semáforo NS se encuentra inicialmente en verde, su transición a rojo incluye el color amarillo, indicando que en el primer ciclo de reloj, el semáforo entra en la primera condición del if y, en el segundo ciclo, le corresponde la segunda condicion, es decir, *else*; de esta forma, ambos semáforos se manteriene en su estado correcto el cual es: EO en verde y NS en rojo.

```
when "0001"|"0011"|"0010" =>
      case EO is --si el semaforo EO no esta en rojo
              when "001" => EO <= "010"; NS <= "100";
              when "010" => EO <= "100"; NS <= "001";
              when "100" => EO <= "100"; NS <= "001";
      end case;
      reset <= '1';
```

Figura 4. Representación de un caso especial dentro del código.

En la figura 4 se muestra una situación similar a la anterior donde el control del semáforo es distinto debido a las condiciones especiales definidas por el planteamiento del problema. En el caso en el que se presentan las entradas especificadas en los sensores, el estado final del semáforo debe ser EO en rojo y NS en verde y, dado que el semáforo principal es EO, el segundo *case* se hace en base a él. Al igual que en el caso anterior, el comportamiento es similar a un caso estático, indicando que la transición se hace al inicio y, a lo largo de la simulación, las únicas transiciones posibles serán controladas por los sensores. Dado que el semáforo EO debe hacer su transición hacia rojo, es necesario pasar por el color amarillo; por ello, si el estado del semáforo es verde al ingresar al *case*, se realizara la transición a amarillo en el siguiente ciclo de reloj y, al ocurrir otro ciclo ascendente por parte de la señal de reloj, se llegara al estado rojo donde permacerá hasta que exista un cambio en los sensores.

Por otro lado, el funcionamiento de los semáforos requiere de un contador para establecer las unidades de tiempo en los casos que lo requieran y, como se había mencionado anteriormente, este contador se encuentra en una dispositivo individual permitiendo un diseño

especial. El diseño realizado para el contador, cuenta con dos entradas, las cuales son la entrada *reset*, que es una salida del dispositivo con el código del semáforo; así como la entrada reservada para la señal de reloj. Por otro lado, las salidas de este dispositivo son un vector de tres bits, utilizados como entrada para el semáforo. En la figura 5 se pueden observar las entradas y salidas mencionadas declaradas en la entidad.

```
entity contador is
port(
        reset: in std_logic ;
        cout: out std_logic_vector (0 to 2);
        clk: in std_logic);
end;
```

Figura 5. Declaración de los puertos de la entidad “contador”.

El funcionamiento del contador es sencillo, comienza en cero y se incrementa en una unidad por cada ciclo de reloj ascendente y, al llegar al límite establecido por los tres bits, el contador se reinicia; no obstante, el contador también se reinicia al recibir una señal alta en la variable *reset*, la cual es una salida del semáforo. En la figura 6 se muestra claramente el funcionamiento del contador a través de la arquitectura declarada en el código, donde se observa como una estructura *if* verifica si se ha recibido una señal de *reset* o si el contador puede seguir incrementando su cuenta.

```
architecture arch_contador of contador is
signal count :std_logic_vector (0 to 2);
begin
  process (clk, reset)
        begin
         if (reset = '1') then
                count <= (others=>'0');
         elsif (rising_edge(clk)) then
                count <= count + 1;
         end if;
        cout <= count;
 end process;
end architecture arch_contador;
```

Figura 6. Comportamiento del contador.

## Código

### semaforo.vhd

```
--SEMÁFORO

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity semaforo is
port(
  sensores:in std_logic_vector(3 downto 0);--sensores
  clk: in std_logic;
  count: in std_logic_vector(0 to 2);
  reset: out std_logic;
  NS: out std_logic_vector(0 to 2);--salida semaforo NS
  EO: out std_logic_vector(0 to 2));--salida semaforo EO
```

```
end semaforo;

architecture arch_semaforo of semaforo is
begin

state_trans:
      process(clk)
      begin
        if(clk'event and clk='1') then

            --definir la maquina de estados
                case sensores is
            --este orden esta determinado por los números a la
derecha de la tabla
                when "0000"|"0100"|"1000"|"1100" =>
                 if(NS = "001") then -- si el semaforo NS no esta
en rojo
                       EO <= "100"; NS <= "010";
                       reset <= '1';
                 else
                       EO <= "001"; NS <= "100";
                       reset <= '1';
                 end if;

             when "1111"|"1101"|"1110"|"0101"|"0110"|"1001"|"1010"
=>
                 case EO is
                       when "001" =>-- si EO es verde
                            EO <= "001"; NS <= "100";
                            reset <= '0';
                            if(count = "100") then
                           --si pasaron 20 segundos cambia a
amarillo
                                 EO <= "010"; NS <= "100";
                                 reset <= '1';
                            end if;
                       when "010" => --si EO es amarillo
                            EO <= "100"; NS <= "001";
                            reset <= '0';
                       when "100"=>--si  EO es rojo
                            if(count < "001") then
                                 EO <= "100"; NS <= "001";
                                 reset <= '0';
                            elsif(count = "001")then
                             --si pasaron 5 segundos cambia a
amarillo
                                 EO <= "100"; NS <= "010";
                                 reset <= '0';
                            else
                                 EO <= "001"; NS <= "100";
                                 reset <= '1';
                            end if;
                 end case;

                when "1011"|"0111" =>
                 case EO is
```

```
                    when "001" =>-- si es verde
                         EO <= "001"; NS <= "100";
                         reset <= '0';
                         if(count = "100") then
                        --si pasaron 20 segundos cambia a
amarillo
                              EO <= "010"; NS <= "100";
                              reset <= '1';
                         end if;
                    when "010" => --si es amarillo
                         EO <= "100"; NS <= "001";
                         reset <= '0';
                    when "100"=>--si es rojo
                         if(count < "010") then
                              EO <= "100"; NS <= "001";
                              reset <= '0';
                         elsif(count = "010")then
                        --si pasaron 10 segundos cambia a
amarillo
                              EO <= "100"; NS <= "010";
                              reset <= '0';
                         else
                              EO <= "001"; NS <= "100";
                              reset <= '1';
                         end if;
                end case;

            when "0001"|"0011"|"0010" =>
                case EO is --si el semaforo EO no está en rojo
                         when "001" => EO <= "010"; NS <= "100";
                         when "010" => EO <= "100"; NS <= "001";
                         when "100" => EO <= "100"; NS <= "001";
                    end case;
                reset <= '1';
               end case;

  end if;

end process state_trans;
end arch_semaforo;
```

contador.vhd

```
--Contador
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity contador is
port(
     reset: in std_logic ;
     cout: out std_logic_vector (0 to 2);
```

```
      clk: in std_logic  );

end;

architecture arch_contador of contador is
signal count :std_logic_vector (0 to 2);
begin

  process (clk, reset)
     begin
      if (reset = '1') then
          count <= (others=>'0');
      elsif (rising_edge(clk)) then
          count <= count + 1;
      end if;
     cout <= count;

 end process;
end architecture arch_contador;
```

# Resultados

Después de concluir con el diseño del crucero vial, se realizaron pruebas para comprobar su funcionamiento utilizando la herramienta de software para la simulación de circuitos Proteus y el software de ispLever que permite generar los archivos .jed necesarios para el funcionamiento de los dispositivos programables.

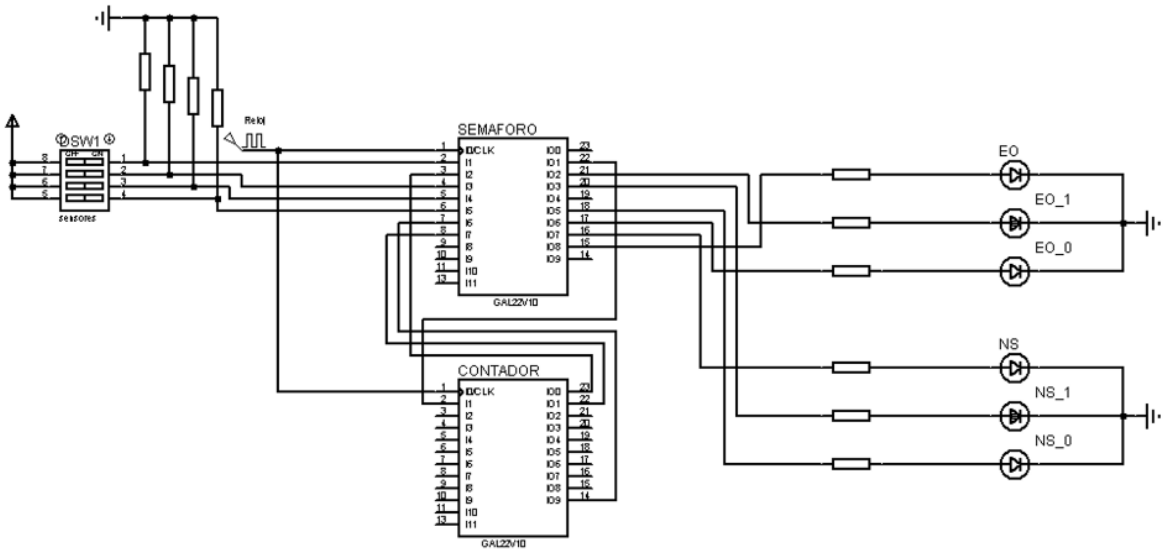


Figura 7. Diagrama de conexiones del circuito.

En la figura 7 se muestra el diagrama esquemático del circuito final, el cual requirió de distintos componentes para crearlo, tales como: dos dispositivos programables GAL22V10, un dipswitch de cuatro interruptores, seis diodos leds donde dos son rojos, dos son amarillos y los dos restantes son verdes, así como seis resistencias de 100 ohms, cuatros resistencias de un kilo ohm, una señal de reloj, una fuente de voltaje y tres señales a tierra.

Dado que el contador tiene como límite 7 ciclos de reloj, se cambió el periodo de la señal de reloj por 5 segundos para que se lograra tener los 20, 10 y 5 segundos para que así el

semáforo detectara cuando pasan 4, 2 y 1 ciclo de reloj respectivamente y cambie de estado. Otra cosa que se tiene que tomar en cuenta es que debido a la configuración del dipswitch, si se quiere tener una entrada baja se tiene que poner el interruptor del lado *on* y por el contrario en el lado de *off* se tiene una entrada alta.

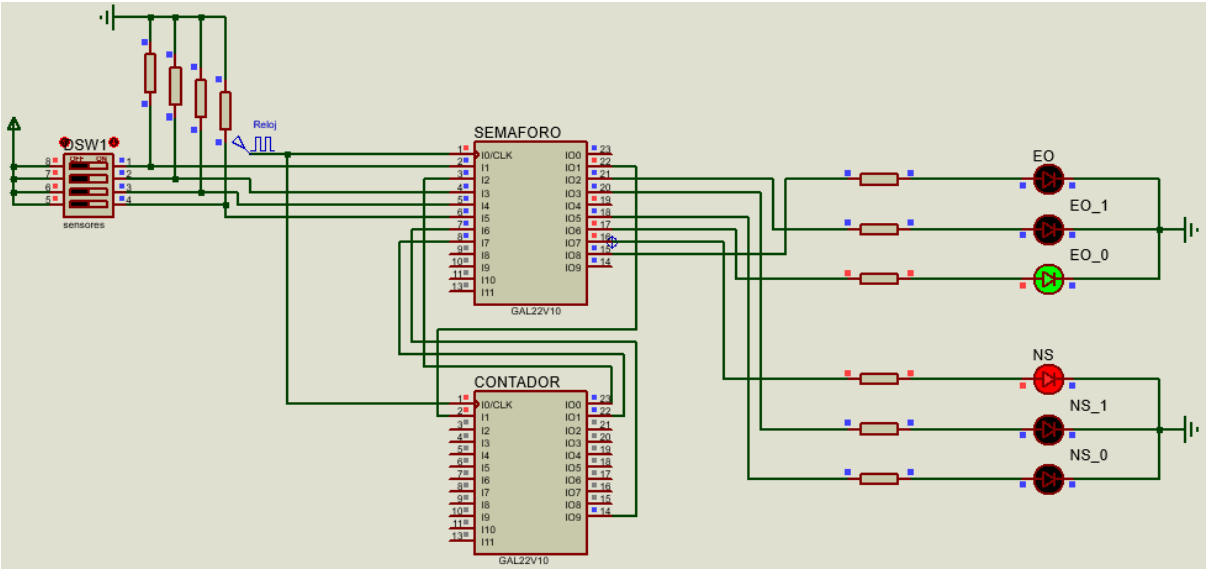


Figura 8. Crucero vial con semáforo EO verde y NS rojo con sensores en caso 1 (2).

En la figura 8 se puede observar que se ingresó una entrada baja en cada uno de los sensores por lo tanto se tiene el semáforo EO en verde y el semáforo NS en rojo por tiempo indefinido o hasta que se cambie el estado de los sensores, lo cual coincide con lo que indica la tabla 1, ya sea en el estado 1 o 2 ya que son equivalentes.

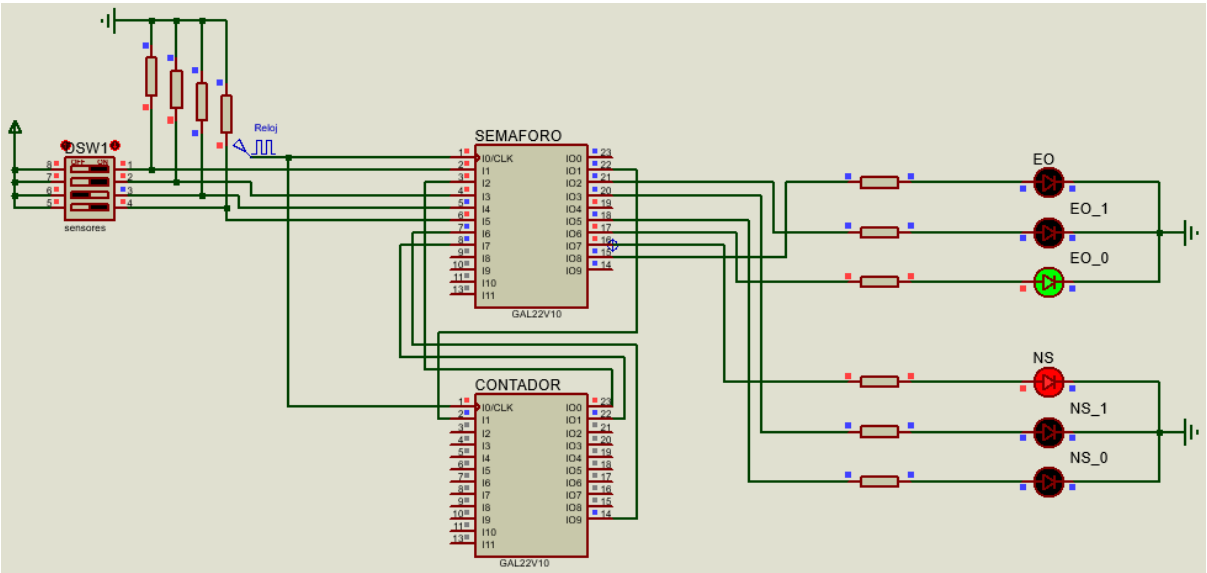


Figura 9. Crucero vial con semáforo EO verde y NS rojo con sensores en caso 3 (5).

Por otro lado en la figura 9 en la entrada en los sensores B, C y D se tiene una entrada alta y en el sensor A una entrada baja, lo cual aplica para el caso 3 o 5 ya que son equivalentes, por

lo tanto al venir del estado anterior donde el semáforo EO está en verde y el semáforo NS en rojo, estos permanecen así por 4 ciclos de reloj.

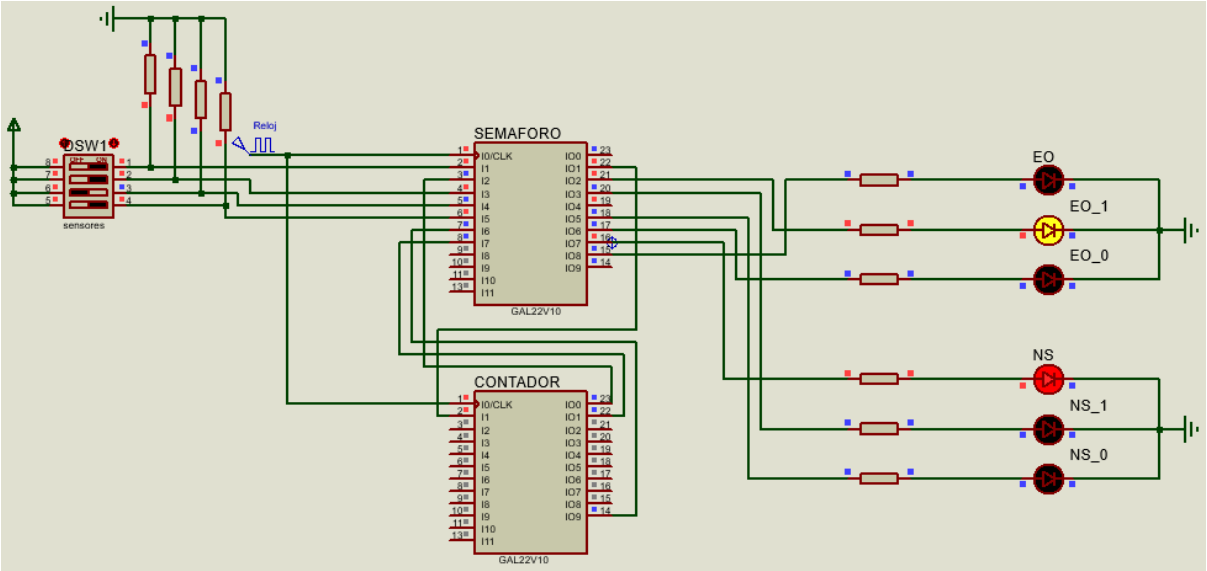


Figura 10. Crucero vial con semáforo EO amarillo y NS rojo con sensores en caso 3 (5).

Despues de cuatro ciclos de reloj el semaforo EO cambia a amarillo como se puede ver en la figura 10, pero el semaforo NS permanece en rojo ya que aun pueden seguir pasando carros por la carretera EO lo cual evita algun posible accidente en caso de implementarlo en la vida real.

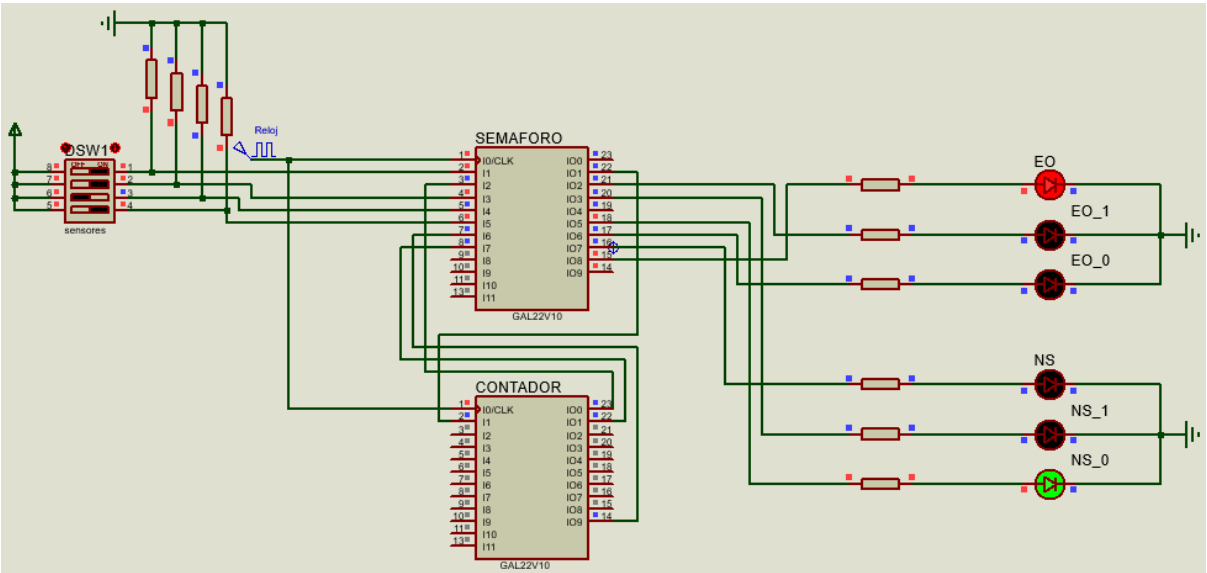


Figura 11. Crucero vial con semáforo EO rojo y NS verde con sensores en caso 3 (5).

Viniendo del caso anterior donde el semáforo EO estaba en amarillo, ahora cambio a rojo y esta vez el semaforo NS si cambio a verde como se observa en la figura 11 para permanecer asi por un ciclo mas de reloj.

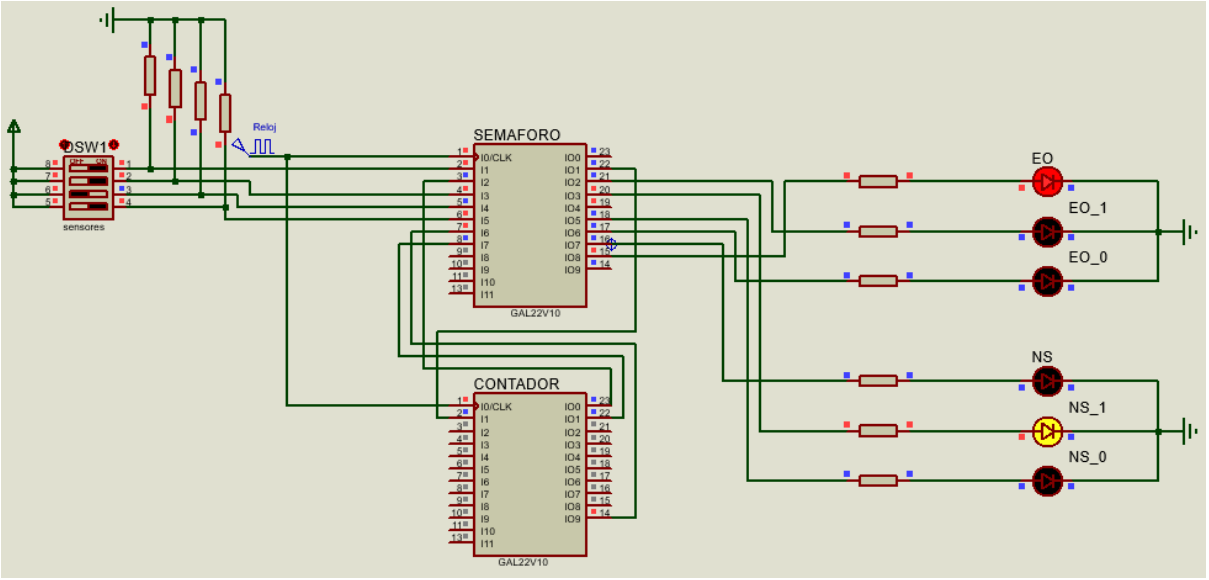


Figura 12. Crucero vial con semáforo EO rojo y NS amarillo con sensores en caso 3 (5).

Por último como en el caso 3 el semáforo NS solo necesita permanecer en verde por un ciclo de reloj, al pasar este el semáforo NS cambiara a amarillo y el semaforo EO permanecera en rojo ya que aun pueden seguir pasando carros como se observa en la figura 12. Después de esto el estado de los semáforos cambiará a como se tiene en la figura 9.

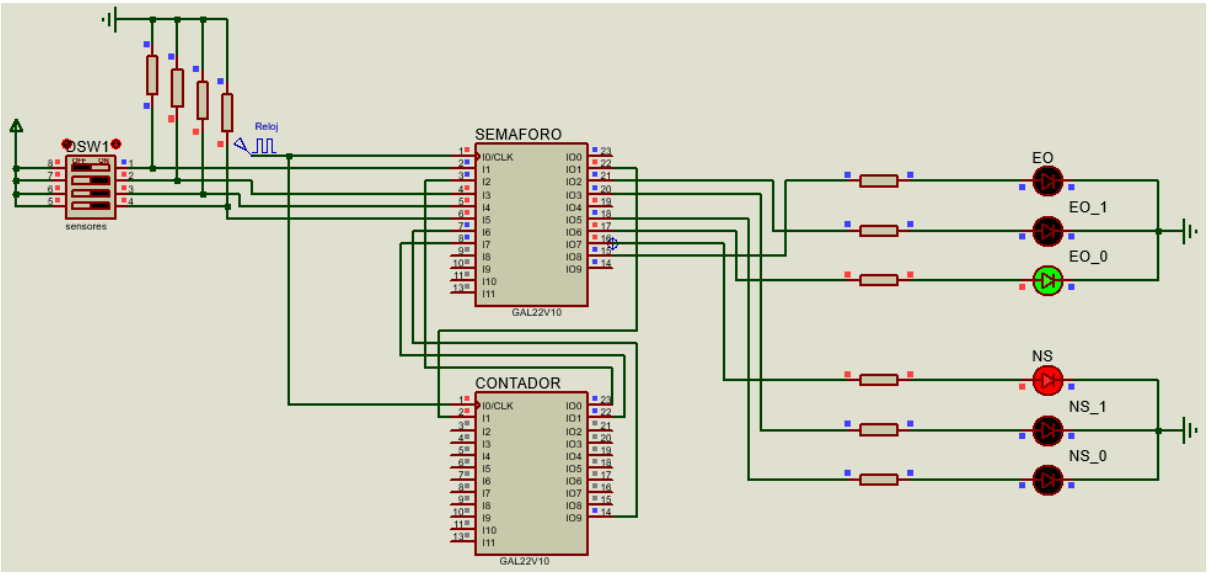


Figura 13. Crucero vial con semáforo EO verde y NS rojo con sensores en caso 4.

Por otro lado, en la figura 13 se muestra un cuarto caso, donde los sensores muestran que hay más autos esperando el semáforo NS; por lo que el semáforo EO permanecerá 20 segundos en verde y después el semáforo NS estará 10 segundos en verde. En la figura 13 se puede observar como el semáforo EO se encuentra en verde mientras que el otro semáforo permanece en rojo.

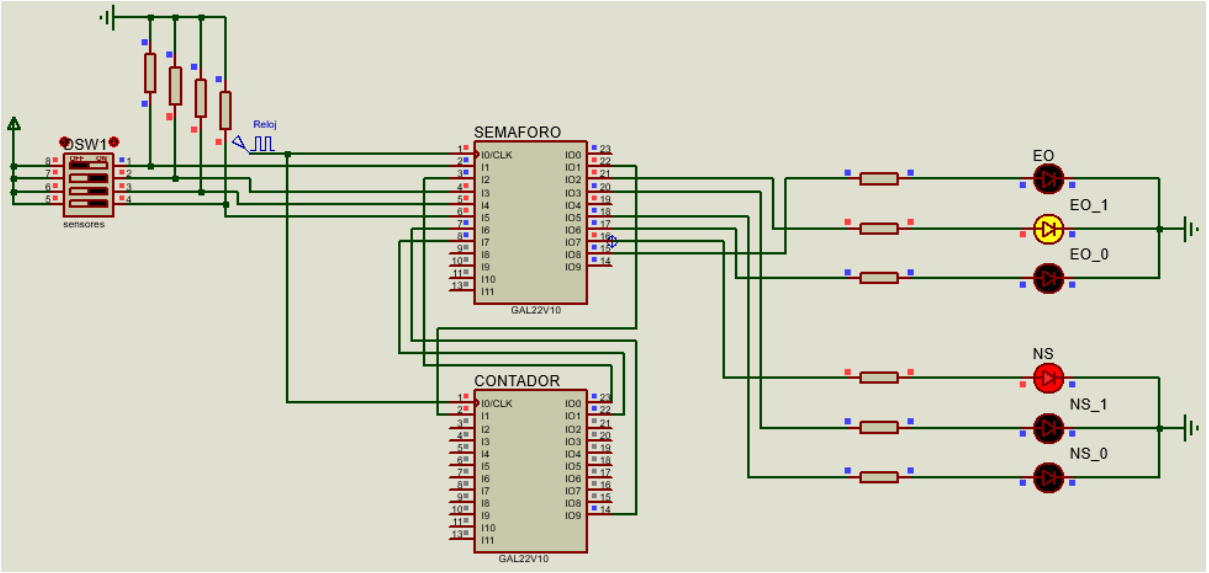


Figura 14. Crucero vial con semáforo EO amarillo y NS rojo con sensores en caso 4.

De la misma manera, en la figura 14 se puede ver como el semáforo EO comienza su transición hacia el estado rojo, lo que significa que requiere pasar por el amarillo. Mientras tanto, el semáforo NS permanece en rojo hasta el siguiente ciclo de reloj.

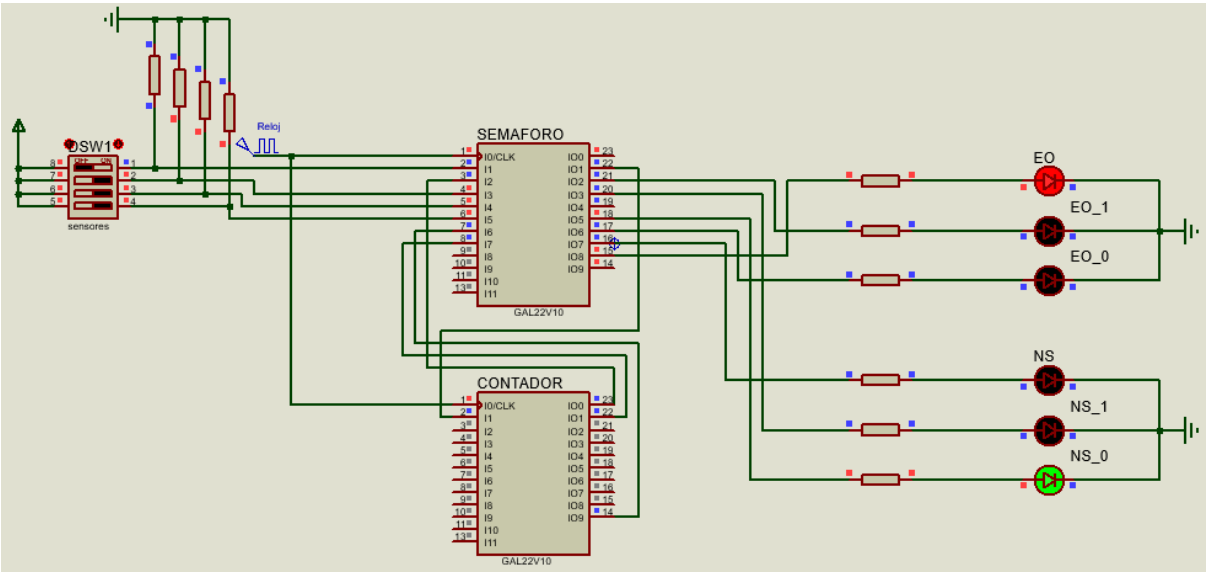


Figura 15. Crucero vial con semáforo EO rojo y NS verde con sensores en caso 4.

Luego en el siguiente ciclo de reloj los estados de los dos semáforos cambian a como se muestra en la figura 15, pasando el semáforo EO de amarillo a rojo y el semaforo NS de rojo a verde, permaneciendo asi por dos ciclos de reloj para asi simular los 10 segundos.

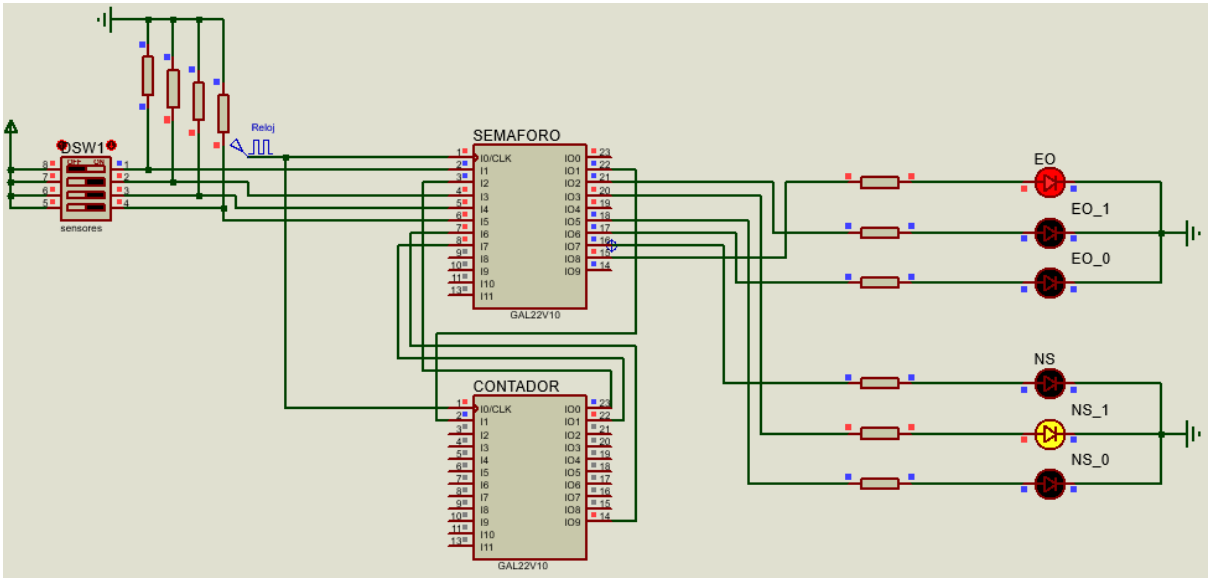


Figura 16. Crucero vial con semáforo EO rojo y NS amarillo con sensores en caso 4.

En la figura 16 se muestra como, en este caso, el semáforo NS es el que esta realizando su transición hacia rojo, por lo que debe pasar por amarillo. En esta figura se observa como la secuencia se continúa repitiendo hasta que ocurra un cambio en los sensores, indicando que ya no hay más autos que requieran esta configuración.

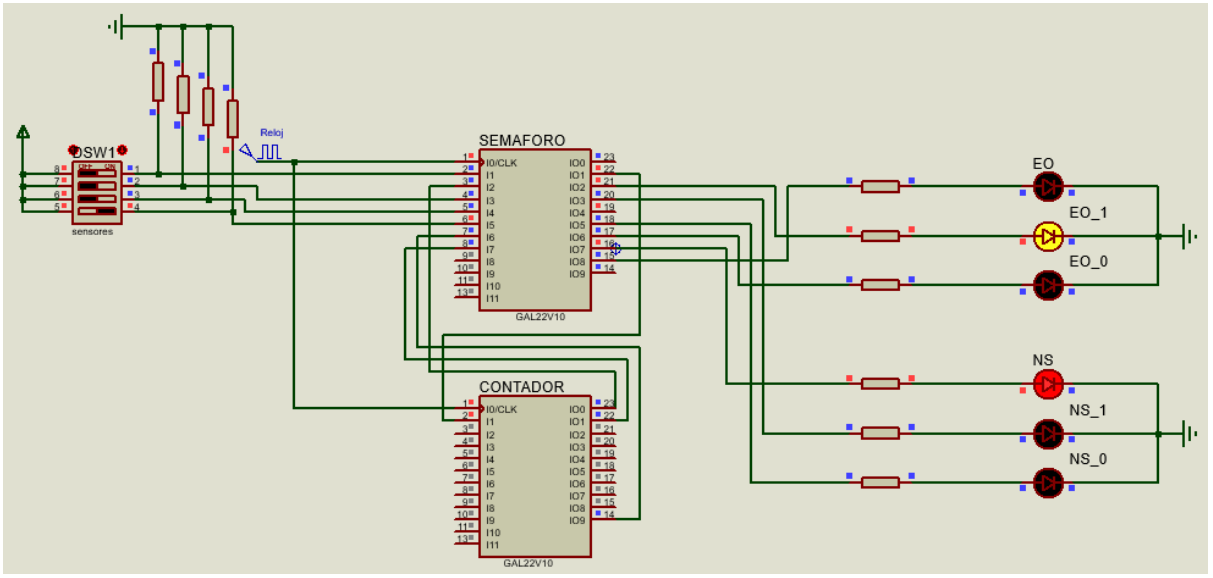


Figura 17. Crucero vial con semáforo EO amarillo y NS rojo con sensores en caso 6.

En la figura 17 se puede observar que, mediante los sensores de entrada, se ingresa la secuencia “0001” indicando que el semáforo EO debe estar en rojo y que el semáforo NS debería estar en verde. En la misma figura se muestra como el semáforo EO esta realizando su transición de verde a amarillo para llegar a rojo; de la misma forma, el otro semáforo permenece en rojo hasta el siguiente ciclo de reloj donde puede simplemente cambiar a verde.

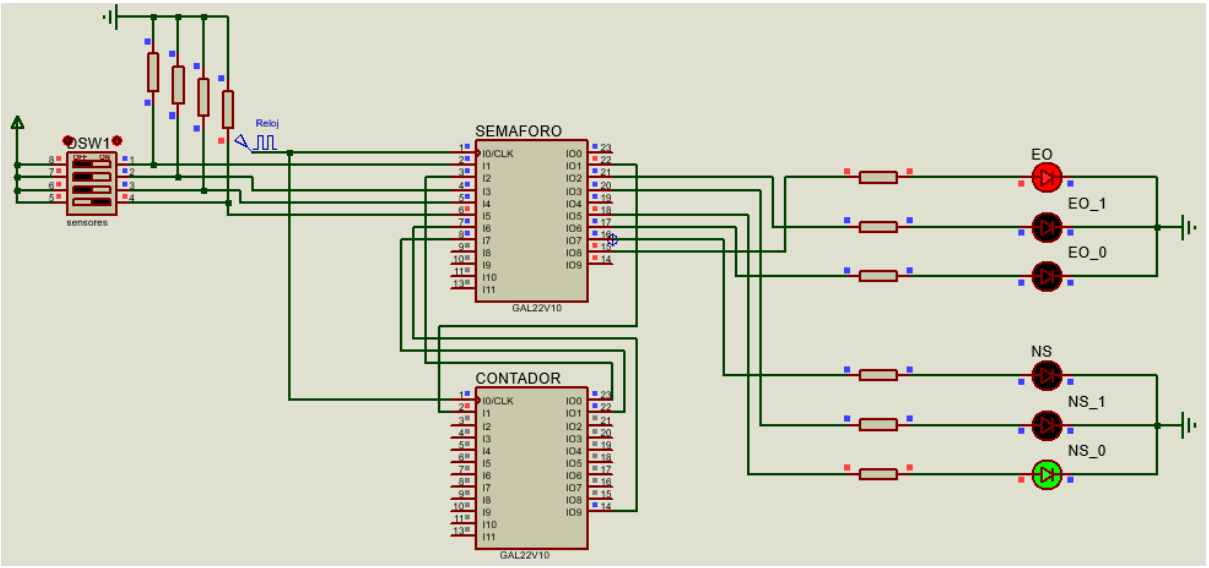


Figura 18. Crucero vial con semáforo EO rojo y NS verde con sensores en caso 6.

En la figura 18 se muestra esta transición completa donde el semáforo EO ha alcanzado el estado rojo y permanecerá allí hasta que ocurra un cambio en los sensores; así como el semáforo NS se quedará en verde hasta presenciar dicho cambio, comprobando que este caso funciona correctamente.

## Discusión o interpretación

A partir del desarrollo del proyecto, se entendió más claramente la forma en que VHDL estructura los códigos así como las restricciones que existen al utilizar ciertas instrucciones ya que, por ejemplo, la instrucción *wait* que no puede ser usada en estructuras *case*. Por otro lado, se infiere que el uso de señales (variables declaradas como *signals*) causa problemas al tomar ciertos puertos de dispositivo GAL y provocar que el dispositivo no pueda reservar suficientes puertos para las entradas y salidas declaradas.

Asimismo, el desarrollo e implementación del crcucero vial sufrió de algunos cambios, especialmente en los casos donde se especificaba que ambos semáforos debían encontrarse en verde. En este caso, se optó por una opción similar a la realidad donde un semáforo que intersecta con otro no debe presentar el mismo estado y se decidió alternar los estados de forma que se siga cumpliendo lo establecido en el planteamiento del problema y, al mismo tiempo, no se presenten ambos estados en verde.

Finalmente, el resultado es similar a los vistos anteriormente en otros sistemas digitales aplicados a ciudades modernas como un medio para reducir la congestión del tráfico, proporcionando una solución que involucra el diseño digital. El crucero vial no presenta discrepancias con un semáforo ordinario, es una versión perfeccionada del mismo que conlleva a mejores resultados. Sin embargo, es importante resaltar que el diseño fue realizado en base a condiciones específicas, si se desea recrear el diseño, este debe ajustarse a las necesidades del entorno y los resultados estarían sujetos a interpretación.

# Conclusiones

## Alondra Carrasco

Al elaborar este proyecto pude poner en práctica los conocimientos que adquirí durante el curso para lograr llevar a cabo un circuito secuencial más complejo que los que se había trabajado con anterioridad, sin embargo el proceso de realización no fue sencillo ya que se tuvieron que elaborar varias pruebas con distintos diseños para al final llegar a la solución más óptima y con mejores resultados; esto ya que si bien se había llegado a otra solución, implicaba el uso de más PLDs y no eran tan precisos los cambios en los semáforos, por lo tanto se busco otra manera de implementarlo y que fuera mas optima y precisa.

Por otro lado, el proyecto me permitió reforzar y aprender aún más sobre el lenguaje VHDL ya que al tratar de llegar a la mejor solución surgieron dudas y la necesidad de buscar otras formas para aplicar los requisitos del tiempo en cada luz de semáforo y tal vez mucha de esta información investigada no fue aplicada en este proyecto sin embargo puede servir para futuros proyectos. En general este proyecto sirvió para reforzar y ampliar mis conocimientos sobre los temas de circuitos secuenciales y el lenguaje VHDL, además de ver donde se pueden aplicar en la vida real y aplicarlo en otros casos o futuros proyectos.

## Tamara Rico

El desarrollo del proyecto me permitió entender de una forma práctica cómo funciona el lenguaje VHDL así como las posibles aplicaciones que su uso puede tener; así como sus limitaciones y los casos en que podría ser mejor hacer uso de un método diferente. Por otro lado, este proyecto presentó una oportunidad para poner en práctica lo que hemos aprendido a lo largo del semestre para desarrollar algo que es práctica y funcional así como efectivo. Asimismo, el trabajar con el simulador a lo largo del semestre y con la nueva herramienta de software hizo que el trabajo fuera más amigable al momento de cometer errores y la experiencia de aprender de dichos errores mucho más placentera.

Sin embargo, este proyecto también presentó múltiples desafíos al tener que realizar diferentes versiones que solucionaran el problema para llegar a la indicada, siempre tratando de analizar los problemas con el diseño actual y la posible forma de solucionarlos; sin embargo, aprendimos de cada uno de los intentos y cada versión era más eficiente.

Finalmente, el desarrollo del proyecto fue una oportunidad para concluir el semestre de una forma práctica y entretenida, que aporta habilidades a nuestra formación como ingenieros y que nos muestra cómo podemos hacer eso de esta clase de sistemas en el futuro.

# Referencias


1. Benites Morales, A., & León Jiménez, N. J. (s. f.). 4.6 Circuitos Secuenciales. Universidad Autónoma del Estado de Hidalgo. Recuperado 8 de diciembre de 2021, de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro16/46_circuitos_secuenciales.html
2. Torres Valle, F. J. (s. f.). CAPÍTULO I: LENGUAJES DE DESCRIPCIÓN DE HARDWARE. Universidad Autónoma de Guadalajara. Recuperado 8 de diciembre de 2021, de https://xdoc.mx/preview/lenguajes-de-descripcion-de-hardware-5c2d1aa29da5a
3. Martinez Medina, A. (s. f.). Dispositivo Lógico Programable (PLD) - Arquitectura de Computadoras. Arquitectura de computadoras. Recuperado 8 de diciembre de 2021, de https://sites.google.com/site/arquitectura1488/news/introduccion
4. Dispositivos lógicos programables (PLD). (2010, 8 marzo). Electrónica Digital. Recuperado 8 de diciembre de 2021, de https://ecadigitaliiequipo7.wordpress.com/2010/03/08/dispositivos-logicos-programables-pld/