«SSL: Smart-Stop-Light»

Juan David Bernal Sánchez, Alejandro Díaz Córdoba
Juan David Sánchez Quintero
Docentes: Johan Sebastián Eslava y Diana Natali Maldonado Ramirez

1. Introducción

El tráfico urbano es una de las principales problemáticas en movilidad, afectando tanto a conductores como a peatones y ciclistas. Un miembro de nuestro equipo, quien se desplaza a diario en bicicleta, ha identificado que algunos semáforos presentan tiempos ineficientes, ya sea porque permanecen demasiado tiempo en rojo o porque cambian rápidamente sin tener en cuenta la demanda vehicular inmediata. Este problema provoca congestión y tiempos de espera innecesarios.

Para abordar esta situación, proponemos un sistema de semáforos inteligentes a pequeña escala que ajuste sus tiempos en función del flujo vehicular, medido mediante sensores. Este modelo se implementará en una maqueta que simula un cruce entre una avenida y una calle de doble sentido, con semáforos sincronizados en la avenida y asincrónicos en la calle.

El proyecto permitirá visualizar el efecto de la regulación del tráfico a través de una pantalla LCD y el uso de un motor con aspas que simulará el flujo vehicular mediante control por modulación de ancho de pulso (PWM). Con esta implementación, se busca desarrollar una solución escalable para optimizar los tiempos semafóricos en entornos urbanos.

2. Objetivos

El principal objetivo de este proyecto fue consolidar los aprendizajes adquiridos, tanto teóricos, discutidos ampliamente en las clases magistrales, como prácticos, desarrollados durante los laboratorios. Esto se logró a través del proceso de diseño, desarrollo e implementación de un dispositivo digital con la complejidad esperada, resultado de la aplicación de todos los temas tratados en el curso de Electrónica Digital I. Aunque el dispositivo se presenta a nivel de prototipo, también busca ser una solución a una problemática real abordada desde la ingeniería.

Durante las etapas iniciales del planteamiento y desarrollo del proyecto, surgieron varios objetivos específicos que debían cumplirse para alcanzar el éxito del proyecto:

- Dado que se buscó finalizar el proyecto antes de la fecha de entrega del reto, fue necesario estudiar el tema de máquinas de estado antes de que fuera tratado en la clase magistral.
- Elaboración de un diagrama de máquina de estado para cada característica del proyecto. Las características incluyen la visualización, el control del tráfico de los semáforos y la modulación de ancho de pulso de los motores.
- Generación de una descripción de hardware que permitiera controlar los semáforos con tiempos constantes. Esta etapa se realizó como una de las primeras fases de la solución, para luego escalar la descripción de hardware a una que permitiera modificar esos tiempos.
- Definición de intervalos de flujo vehicular asociados a un tiempo fijo para el estado activo de los semáforos e implementación de una descripción de hardware en Verilog en la FPGA, que permitiera cambiar dichos tiempos.
- Selección de los sensores que se utilizarán para medir el flujo vehicular.
- Elección del motor y módulo PWM asociado a la simulación del flujo vehicular.
- Establecimiento de la pantalla LCD que se utilizará para presentar el conteo de vehículos, realizando una descripción de hardware en Verilog para controlarla.

- Creación de una descripción de hardware que integre todos los elementos del proyecto.
- Elaboración de una maqueta para la visualización y disposición de todos los elementos del proyecto.
- Generación de un diseño PCB para implementar los componentes del proyecto en una placa de baquelita.

3. Descripción de la solución realizada

3.1. Funcionamiento principal

En primer lugar, es importante resaltar la base del manejo lógico secuencial del proyecto. Todo parte de la definición de un bloque semáforo. Este tiene como entradas: una señal que le indica cuáles diodos prender y como salidas las señales que encienden o apagan los leds.

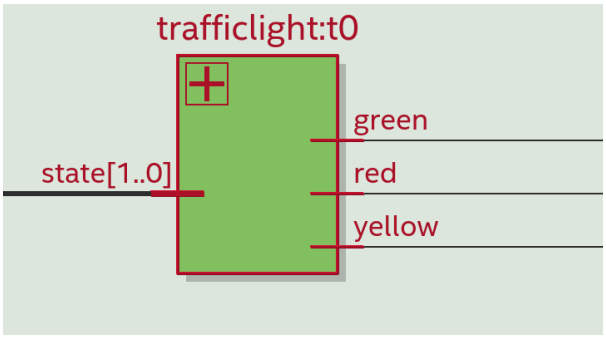

Figura 1: Bloque de semáforo

De esta manera, la siguiente lógica combinacional muestra el comportamiento de los semáforos:

state	green	yellow	red
00	0	0	1
01	0	1	0
10	1	0	0

Esta codificación de dos bits permitió controlar los 3 semáforos de una manera simplificada con un registro de 6 bits llamado *tfst*. Los bits del registro estarían distribuidos de tal manera que los dos más significativos controlan los semáforos sincronos, es decir los semáforos de la avenida. Los siguientes dos bits controlan el semáforo con flujo de este a oeste, y los dos bits menos significativos controlan el semaforo con flujo oeste a este.

Por lo anterior las siguientes combinaciones del registro permiten conocer cual semáforo está en estado activo. NS corresponde al flujo de norte-sur, SN sur-norte, EW este-oeste y WE oeste-este.

tfst	NS	SN	EW	WE
100000	1	1	0	0
001000	0	0	1	0
000010	0	0	0	1

Ahora, para llevar a cabo la solución general del proyecto se planteó una secuencia de pasos que se debían de seguir para construir la maquina de estados finita. Esta secuencia inicia colocándole condiciones iniciales a dos registros $green = 6b'001000$ y $yellow = 6b'000100$. En un siguiente ciclo de reloj, se pasa a un estado donde se realiza un desplazamiento de 2 bits a la izquierda de estos registros, pero los dos bits más significativos pasan a ser los

menos significativos, es decir que $green = 6b'100000$ y $yellow = 6b'010000$ con el fin de mostrar el funcionamiento del proyecto desde el estado activo de la avenida (NS y SN sincronos).

Como la solución general del proyecto se centra en contar el flujo vehicular para decidir el tiempo activo de los semáforos, se decidió dar un tiempo de espera de $3{,}5s$ para empezar a contar carros por primera vez porque los conductores pueden estar distraídos ó se pueden demorar en arrancar. Una vez que ha pasado este tiempo se inicializa un registro de **n = 3'd4**, este indica el numero de ciclos que se van a contar carros. La idea principal fue contar carros durante un periodo de $t = 2 * n + 1[500ms]$. En el momento que pasa este tiempo de conteo, dependiendo de la cantidad de vehículos que pasaron, el sistema decide darle un tiempo adicional para que siga en verde el semáforo. A continuación se presentan los intervalos escogidos:

Count_car	t_add $[s]$
0-3	4
4-7	6
8-11	8
12-15	10
>15	14

Una vez que el tiempo de espera adicional termina, entonces se vuelven a contar carros, pero esta vez al registro n se le resta 1. Lo anterior quiere decir que en este nuevo ciclo no se cuentan carros en un intervalo de $2n + 1$ sino que se cuenta $2(n - 1) + 1$. De esta manera el proceso de conteo y decisión de tiempo adicional se repite n veces hasta que $n = 0$. En este caso, al registro *tfst* se le asignan los valores almacenados en el registro *yellow*, y se da un tiempo de espera de $3s$ para cambiar el registro a $tfst = 6'b000000$ con el fin de indicarle que debe de colocar todos los semáforos en rojo durante $10s$. Finalmente, una vez se acaba de contar el tiempo del semáforo en rojo, se vuelve al segundo estado donde se le hacía un corrimiento de bits al registro *green* y registro *yellow*. De esta manera se puede notar que cada vez que se pase por todos los estados y se haga un corrimiento de bits de los registros, se está cambiando el semaforo que se prenderá.

3.2. Conteo de flujo vehicular

En este caso se decidió utilizar una configuración de diodos emisor y receptor para contar el flujo por cada semáforo. Inicialmente se pensó que con solo contar un flanco positivo de la variación de la señal era suficiente para incrementar el contador, pero en la práctica el contador estaba detectando entre 1000 y 6000 variaciones por lo que se decidió implementar un reloj para realizar un procesamiento digital de señales. Con el uso de un registro auxiliar y auxiliar previo de 14 bits, se comparan por cada ciclo de reloj de 2Hz. El registro auxiliar tiene asociado un contador de variaciones de la señal del diodo receptor y por cada ciclo de reloj se le asigna al registro auxiliar previo el valor del registro auxiliar. Si en el siguiente ciclo de reloj los dos registros no tienen el mismo valor, entonces se le suma al contador una unidad.

3.3. Tratamiento del conteo por semáforo

El modulo principal del manejo de tiempo de los semáforos tiene como entradas los contadores netos en el tiempo de cada semáforo, estos contadores son de 14 bits para poder llegar a representar hasta 10000 carros por semáforo. Es por esto que se realizó un tratamiento especial para poder conocer el numero de vehículos en los intervalos de conteo.

Solo se utilizaron 4 bits de cada contador para poder contar hasta 15 vehículos por cada semáforo, es decir que los semáforos de la calles pueden llegar a tener un tiempo adicional por ciclo de conteo de $10s$, pero la avenida $14s$ porque se suman los 4 bits del contador sur y el contador norte.

Recordando la importancia del registro *tfst* del proyecto. Este permite conocer cual o cuales de los 4 semáforos están prendidos. En el momento que se pasa al estado de conteo, se compara el valor de tfst y dependiendo de los 3 posibles valores definidos se almacena en un registro de 5 bits el conteo que lleva el determinado semáforo. En la siguiente tabla se resume las asignaciones:

tfst	count_car
100000	count_ns_4b + count_sn_4b
001000	count_ew_4b
000010	count_we_4b

En el ciclo que termina el conteo, se le asigna al registro *count_car* la resta entre el valor actual de los contadores y el primer valor almacenado en el registro:

tfst	count_car
100000	count_ns 4b + count_sn 4b - count_car
001000	count_ew 4b - count_car
000010	count_we 4b - count_car

3.3.1. Tratamiento de overflow

Dado que las variables definidas para el proceso de conteo son todas de un tamaño de 5 bits, cuando se realiza la resta con la variable *counter_car* debemos sumar un 32 decimal en la ecuación en el caso de que en la resta se compruebe que hay desbordamiento. En el módulo de verilog donde se describe el conteo, la magnitud de la resta se comprueba mediante un if que compara la magnitud de toda la resta con la de *counter_car*.

Esto se hace porque la resta pude generar un número fuera del rango permitido (desbordamiento u overflow), el cual es de -16 hasta 15 decimal. El procedimiento de sumar 32 decimal es válido considerando que las variables de 5 bits se están tratando como números binarios de complemento a 2. Por ejemplo, si el registro tfst es 100000, y además se cumple que $count\_ns\_4b + count\_sn\_4b - count\_car < counter\_car$, la expresión para *counter_car* se convierte en:

$$counter\_car <= count\_ns\_4b + count\_sn\_4b \\ -counter\_car + 32$$

Más específicamente, el número 32 aparece en particular porque es el número decimal que en binario es de tamaño de 6 bits con un único 1 en su MSB: 100000.

3.4. Visualización

Para la visualización del conteo de carros, así como del tiempo adicional ajustado en cada intervalo de conteo, se optó por implementar una pantalla LCD 16x2.

En primer lugar, es necesario crear un protocolo de comunicación que permita configurar el modo de visualización de la información, así como el número de bits que se envía, que en este caso fue de 8 bits. Posteriormente es necesario hacer una transición entre el mensaje enviado en la primera línea y el que se envía para la segunda. Por último, se envía un comando que retorna la LCD al inicio en donde se actualizan los mensajes y se inicia nuevo el proceso. A continuación se muestra el diagrama equivalente.

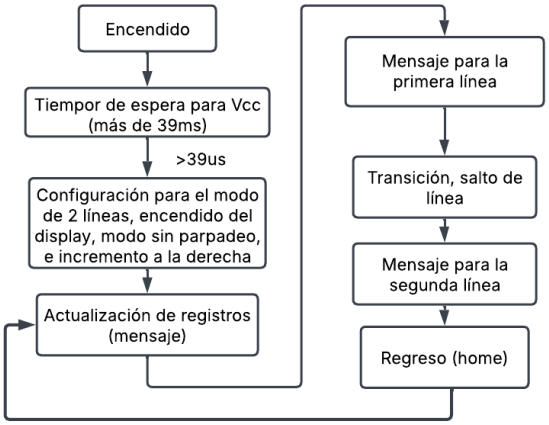

Figura 2: Funcionamiento de la LCD

En este caso, se investigó a fondo el funcionamiento de la pantalla a partir del datasheet del fabricante. Esta pantalla requiere 11 pines para su control principal, entre los cuales se encuentran *RS*, *RW*, *Enable* y *D* (8 bits). Cuando $RS = 0$ y $RW = 0$, la pantalla recibe una instrucción de configuración, definida por los 8 bits de la señal *D*. En cambio, cuando $RS = 1$ y $RW = 0$, se escribe un carácter en la pantalla en código *ASCII* a través de la misma señal *D*. Cabe destacar que la señal *Enable* funciona como un reloj, ya que las nuevas instrucciones se activan en el flanco positivo de dicha señal [1].

Para gestionar la pantalla, se establece una secuencia de operación que comienza con un estado inicial encargado de su configuración. En este estado, se envían una serie de instrucciones, como se ilustra en la figura anterior. Estas instrucciones incluyen la limpieza de la pantalla, la activación del display, la desactivación del cursor y su parpadeo, así como la definición del movimiento del cursor sin desplazamiento del texto hacia la izquierda o derecha.

Dado que la pantalla dispone de 32 casillas de caracteres, se implementa un banco de 32 registros de 8 bits para almacenar la información a desplegar en código *ASCII*. En la descripción de *hardware*, se define un estado específico para actualizar cada posición del banco de registros a partir de una entrada *mensaje_in* de 256 bits, donde cada conjunto de 8 bits representa un carácter en código *ASCII*. Una vez completada la actualización del banco de registros, la secuencia avanza al siguiente estado.

Este nuevo estado se encarga de enviar cada carácter en código *ASCII* a través de la salida *D* en cada ciclo de reloj, hasta completar la primera fila de caracteres. Posteriormente, se activa un estado que desplaza el cursor a la segunda línea en un solo ciclo de reloj. Luego, otro estado transmite los caracteres restantes correspondientes a la segunda línea. Finalmente, el último estado envía una instrucción para mover el cursor a la primera posición y reiniciar el proceso desde la actualización de registros.

Es importante mencionar que se utiliza un reloj con un periodo de $16ms$ para garantizar la sincronización con los tiempos de respuesta de la pantalla y evitar posibles inconvenientes.

3.4.1. Mensaje

Para decidir cuál mensaje enviarle al módulo de la *LCD* fue necesario establecer dos tipos de mensajes según el estado. El funcionamiento es simple, si el estado de la máquina de estados del semáforo es «espera_aditiva» se desplegara en la LCD el siguiente mensaje:

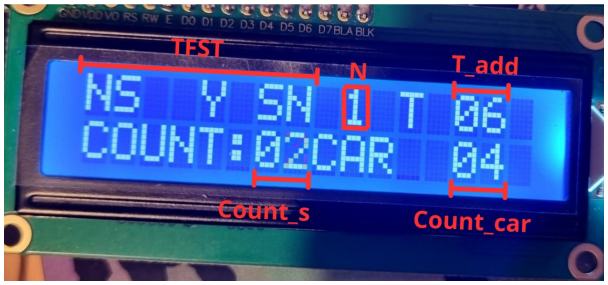

Figura 3: Visualización en display *LCD* - $state = espera\_aditiva$

Donde se muestra el registro n para conocer cuántos ciclos le restan al sistema. También se muestra un contador de tiempo en segundos para conocer el progreso de la espera y de la misma manera se evidencia el tiempo total añadido. Adicionalmente se muestra la cantidad de vehículos que fueron contados, y en los primeros 8 caracteres se despliega el nombre del semáforo el cual está en estado activo.

En caso contrario que no se presente el estado «espera_aditiva», se desplegará un mensaje con el contador total neto de cada semáforo como se muestra a continuación:

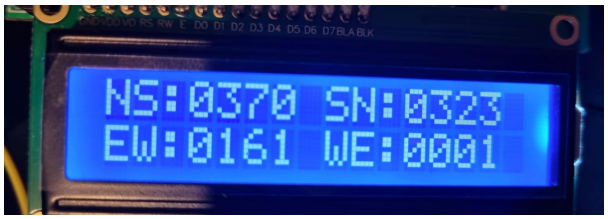

Figura 4: Visualización en display *LCD* - $state != espera\_aditiva$

Para construir el mensaje es importante mencionar que se reciben los números que se desean desplegar en *BCD*. Como segundo criterio, se debe considerar que la *LCD* maneja código *ASCII*, por lo que la señal del mensaje contiene un número de bits de $8bits/char * 32char = 256bits$. En este módulo de *mensaje* se estableció una lógica combinacional que evalúa primero la condición del estado del control de semáforos. Con base en lo anterior, se escribió en código *ASCII* en grupos de 8 bits carácter por carácter según la condición del estado de «espera_aditiva» y *tfst*.

3.5. Servomotor

Un servomotor es un dispositivo eléctrico y electrónico: por una parte necesita una alimentación (en este caso, 5 V), que suministra la energía. Por otro lado, recibe una señal cuadrada que le indica al servomotor a qué ángulo moverse. Este mecanismo se denomina PWM (modulación por ancho de pulso). Las características de esta señal de entrada debe seguir las siguientes características:

- Frecuencia: 50Hz. Es decir, el periodo de la señal son 20ms.
- El *duty cycle* de la señal no debe superar el 10 %. Es decir, la señal debe ser enviada con un tiempo máximo en alto de 2ms.
- Para que el servomotor se mueva en determinada dirección, es necesario mandarle un pulso continuo con un *duty cycle* constante. El servomotor se moverá a la posición asignada para ese pulso, en caso de que llegue a ella, se detendrá.

El servomotor utilizado en el proyecto es la referencia *SG90*. Este servomotor se controla por ángulos.

3.5.1. Principio de funcionamiento

El objetivo del servomotor en el proyecto fue el de simular el flujo de carros. Para este objetivo, no es necesario que él de la vuelta completa, simplemente se necesita que oscile en un rango de ángulos lo suficientemente amplio como para que sea reconocido por la configuración de diodos.

A continuación se muestra el ancho de cada señal en alto, con su respectiva posición asociada.

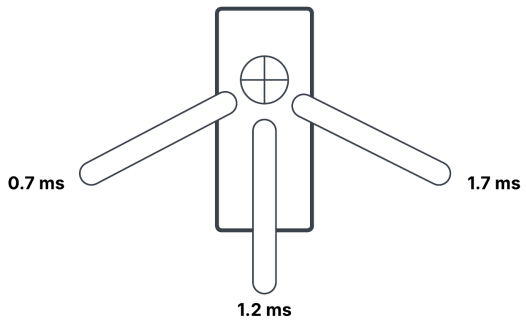

Figura 5: Posición del servomotor para distintas señales

En principio, el proyecto iba a utilizar un circuito integrado que controlara el ancho de pulso del servomotor, pero al ser un problema que se podía resolver desde el diseño de una lógica secuencial en verilog, el equipo decidió que podía generar la señal de control del servomotor.

Para abordar este desafío, se diseñó un módulo que recibe como entradas un reloj de $10KHz$ y otro de $10Hz$. Además, dado que los servomotores no deben moverse cuando los semáforos no están activos, el módulo encargado de controlar los cuatro servomotores incluye una señal de entrada llamada *tfst*. Esta señal se utiliza para habilitar o deshabilitar individualmente cada servomotor según sea necesario.

La solución propuesta emplea un banco de registros que almacena tres valores de tiempo en unidades de cientos de μs, correspondientes a tres ángulos específicos: 7, 12 y 17 [$100\mu s$]. El método consiste en utilizar el reloj de $10KHz$ para contar intervalos de 100 μs y mantener la señal en $HIGH$ hasta alcanzar uno de los valores almacenados en el banco. Posteriormente, la señal se mantiene en LOW hasta completar un total de 200 intervalos de 100 μs ($20ms$). Finalmente, el contador se reinicia y el ciclo se repite.

Para controlar el ángulo deseado y la velocidad del motor, se emplea un registro de velocidad y un contador que determina el ángulo actual. Cada vez que se presiona un pulsador, el registro de velocidad incrementa en $1'b1$. Además, utilizando el flanco de subida del reloj de $10Hz$, se cuenta *2(speed+1)* ciclos antes de incrementar el contador de ángulos. Esto implica que, al aumentar el valor del registro *speed*, el sistema debe esperar más tiempo antes de cambiar de ángulo, lo que efectivamente reduce la velocidad de movimiento del servomotor.

3.6. Diodo receptor y emisor

Por último, es importante especificar la configuración utilizada como sensor, que consistió en dos diodos, uno receptor y otro emisor.

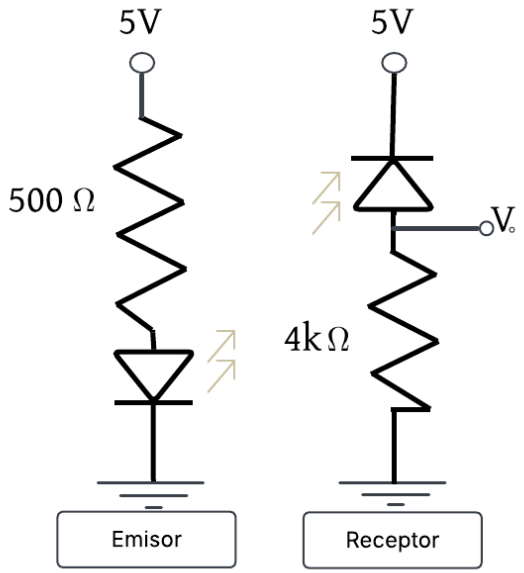

Figura 6: Configuración para diodo emisor y receptor

4. Estructura de la solución

4.1. Control de semáforos y detector de flujo vehicular

Desde un enfoque más gráfico, a partir de un análisis *RTL* se puede evidenciar que el controlador de tráfico recibe como entradas los contadores de flujo vehicular y una señal de *reset*, utilizada para restablecer las condiciones iniciales del sistema.

Por otro lado, cada detector de vehículos cuenta con una instancia de un módulo diseñado para medir el flujo vehicular y realizar un procesamiento digital de la señal generada por el diodo receptor.

El módulo de control de tráfico genera diversas señales de salida, entre las que se incluyen:

- **Estado actual del sistema.**
- **Contador de vehículos detectados.**
- **Contador de segundos transcurridos.**
- **Variable** n, que indica $(n-1)$ ciclos restantes.
- **Tiempo adicional** asignado a cada fase del semáforo.

Estas salidas se utilizan principalmente con fines de visualización, como se mencionó en la sección anterior. No obstante, las señales más relevantes son aquellas que controlan el encendido y apagado de los *LEDs* de cada semáforo: *tf0*, *tf1* y *tf2*.

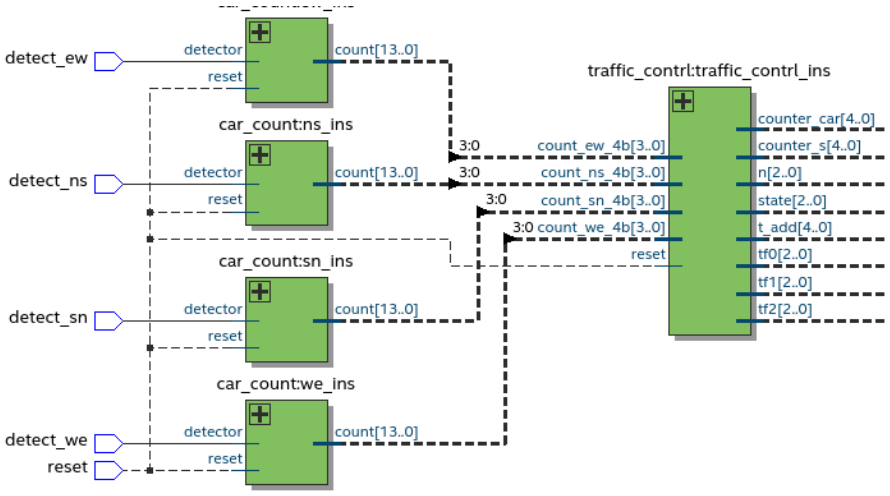

Figura 7: Diagrama de bloques del control de semáforos y detección de flujo vehicular

4.1.1. Diagrama de maquina de estados principal

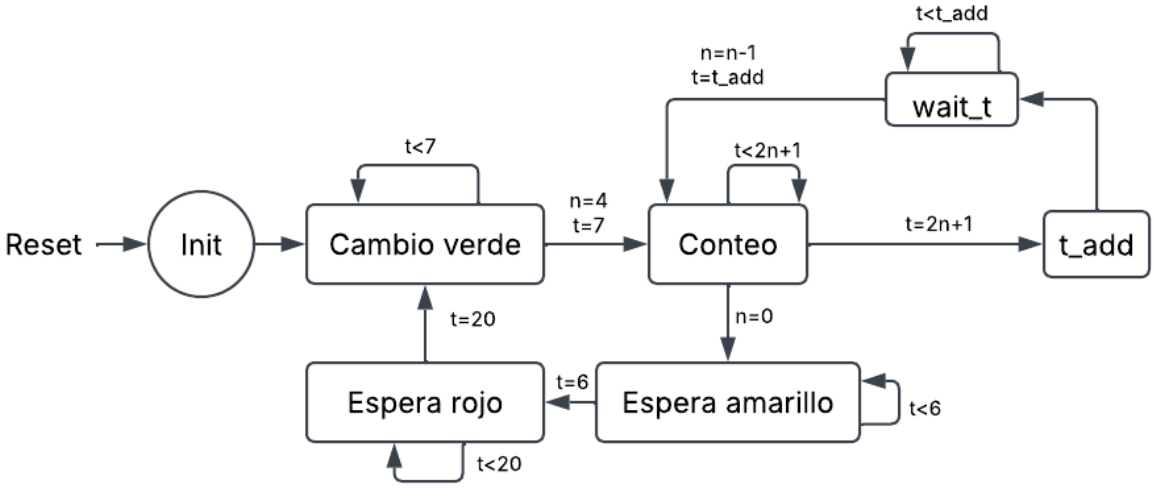

Figura 8: Diagrama de maquina de estados para control de luces de trafico

A continuación se presenta una descripción detallada de lo que ocurre en cada estado:

- **Init:** Se establecen condiciones iniciales para los registros:
  - Contador de medios segundos: $counter\_s <= 5'd0$.
  - Tiempo adicional: $t\_add <= 4'd0$;

- Número de ciclos: $n <= 4$.
- Conteo de carros: $counter\_car <= 5'd0$.
- Registro que indica la señal que debe tener *tfst* para colocar amarillo: $yellow <= 6'b000100$.
- Registro que indica la señal que debe tener *tfst* para colocar verde: $green <= 6'b001000$.
- $tfst <= 6'b000000$;

- **Cambio_verde:** Allí se realiza el desplazamiento de bits mencionado en la descripción de la solución. Este consiste en hacer un desplazamiento de dos bits a la izquierda los bits de los registros *green* y *yellow*, pero moviendo los dos bits más significativos a los dos bits menos significativos; lo anterior se realiza en el primer ciclo de reloj. En el siguiente ciclo de reloj el registro que controla la señal de los semáforos adquiere el nuevo valor del registro *green*. Una vez que se cuentan 7 ciclos ciclos de reloj, se pasa al estado de conteo inicializando el registro $n = 4$.

- **Conteo:** En este estado se registra en *counter_car* el valor del conteo neto del semáforo o los semáforos activos en el primer ciclo de reloj. Una vez se llega a $2n + 1$ ciclos de reloj, se le asigna al contador de carros la resta entre el valor presente del contador neto y el primer valor registrado. Es decir, se realiza la siguiente operación dependiendo del semáforo activo: $counter\_car = count\_counter\_car$

tfst	count_car
100000	count_ns 4b + count_sn 4b - count_car
001000	count_ew 4b - count_car
000010	count_we 4b - count_car

Cabe resaltar que si el registro n se encuentra en 0, se pasa al estado de espera amarillo.

En este mismo ciclo se da la transición al proceso de asignación de tiempo adicional.

- **T_add:** En un solo ciclo de reloj se le asigna a un registro *t_add* un tiempo adicional dependiendo del número de carros contados. En el siguiente ciclo de reloj se pasa a un tiempo de espera.

Count_car	t_add $[s]$
0-3	4
4-7	6
8-11	8
12-15	10
>15	14

- **wait_t:** En este estado se cuenta el número de ciclos de reloj asignado en el registro *t_add* para volver a pasar el estado de conteo. En la transición al proceso de conteo, se le resta una unidad al registro n.

- **Espera_amarillo:** En el primer ciclo de reloj se le asigna al registro *tfst* el valor de *yellow*, y se esperan 6 ciclos de reloj para pasar a la espera en rojo.

- **Espera_rojo:** Allí se asigna $tfst = 6'b000000$ para colocar en rojo todos los semáforos durante 20 ciclos de reloj para realizar el desplazamiento de bits nuevamente en el estado de cambio verde.

4.2. Visualización

Tal y como se explicó anteriormente, el protocolo de comunicación, así como la secuencia de muestreo de datos siguen un flujo específico y repetitivo. A continuación se muestra el diagrama presentado anteriormente, que ilustra el funcionamiento de la LCD.

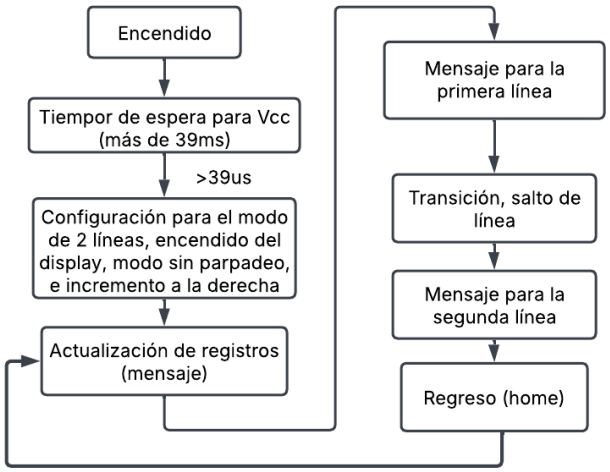

Figura 9: Diagrama de visualización en la LCD

Tal y como se muestra en la anterior imagen, en primer lugar es necesario dejar un tiempo de espera de por lo menos 39ms. Posterior a esto, se envía una serie de comando que permite la disposición de display en 2 líneas, además de su configuración con 8 bits.

Naturalmente, es necesario que se actualicen los registros, es decir, los datos a proyectar en la LCD. A la hora de transmitir los datos, primero se muestran los de la primera línea. Para saltar de línea, es necesario enviar una señal especifica de configuración con $RS = 0$ para indicarle a la pantalla que mueva el cursor a la segunda linea, y terminar de enviar los caracteres que van en la segunda linea.

El diagrama de este modulo *display* se resume en estas entradas y salidas:

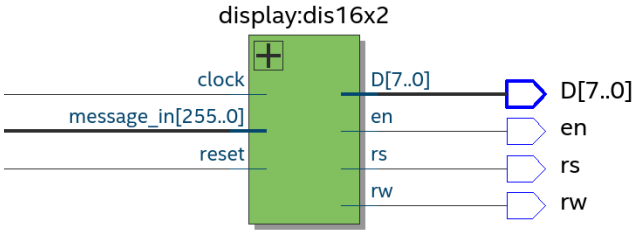

Figura 10: Diagrama de entradas y salidas del modulo de display

En la anterior imagen se observa que el display tiene como entradas:

- **Clock**: Reloj con frecuencia de 62Hz.
- **Message_in:** Una señal que por cada conjunto de 8 bits contiene codificado un carácter en código *ASCII*.
- Una señal reset para volver a inicializar la tarjeta.

La descripción de *hardware* del módulo de *display* se creó considerando el anterior diagrama y el siguiente diagrama de máquina de estados:

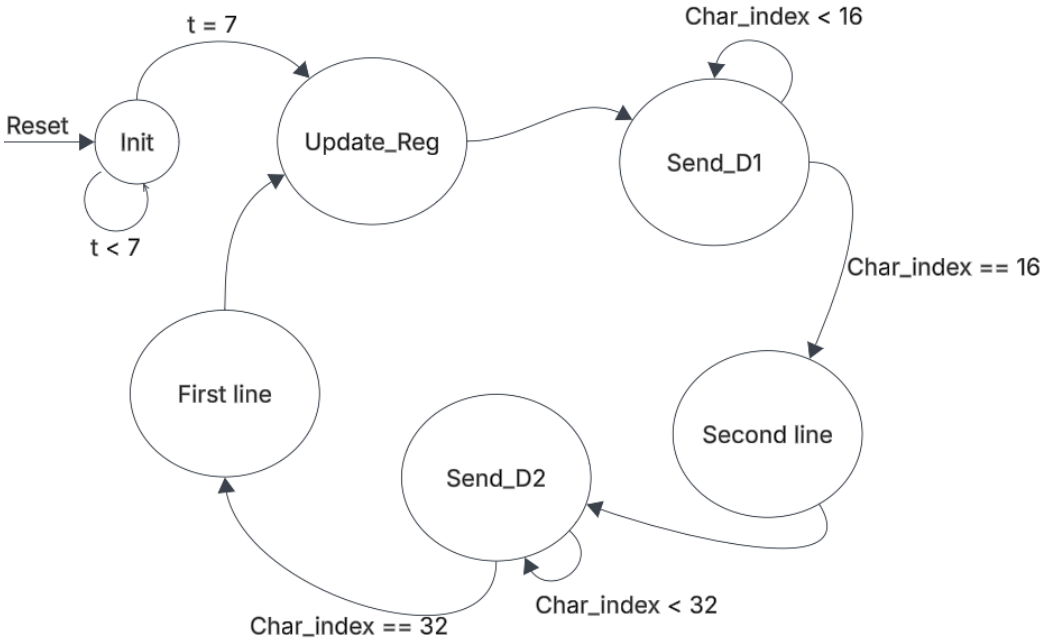

Figura 11: Diagrama de maquina de estados para *LCD*

A continuación se presenta una descripción detallada de lo que ocurre en cada estado:

- **Init:** Se esperan dos ciclos de reloj para inicializar la pantalla. Luego, por cada ciclo de reloj se envía a través de la señal D una instrucción de configuración con *rs =0*. Se envían las siguientes 4 instrucciones de configuración:
  - $D <= 8'b00111100$: Envío de instrucciones utilizando los 8 bits de entrada de la pantalla, uso de las dos lineas de la pantalla y la resolución de la fuente.
  - $D <= 8'b00001110$: Se enciende el display y se apaga el cursor junto con su parpadeo.
  - $D <= 8'b00000001$: Se limpia la pantalla.
  - $D <= 8'b00000110$: Se deshabilita el corrimiento de los caracteres a izquierda o derecha, y el cursor se incrementa una posición por carácter desplegado.
- **Update_Reg:** Se actualizan los registros del banco de registros interno del módulo con la señal que contiene el mensaje codificado en código *ASCII*.

  Cuando pasan 7 ciclos de reloj ocurre la transición al estado *Send_D1* y se inicializa el registro *char_index* = $5d0$.
- **Send_D1:** En este estado se envía un carácter del banco de registros por cada ciclo del reloj hasta enviar los primeros 16 caracteres de la primera línea. Para acceder a la enésima dirección del banco de registro se debe utilizar un contador de índice de caracteres *char_index* que aumenta por cada ciclo de reloj hasta que es igual a 16, allí ocurre la transición al estado *Secondline*.

- **Secondline:** La salidas se configura como $RS = 0$ y $D = 8'b11000000$ para indicarle a la pantalla que debe ubicar el cursor en la primera posición de la segunda línea.
- **Send_D2:** Se vuelve a configurar $RS = 1$ y se sigue incrementando el índice de carácter para continuar mostrando los caracteres restantes en la segunda línea de la pantalla. Una vez el contador llega a 32, se realiza la transición al estado *firstline*.
- **Firstline:** Se configura $RS = 0$ para realizar una instrucción de configuración y se envía en $D <= 8'b10000000$ para retornar el cursor en la primera posición de la primera linea. En el siguiente ciclo de reloj se actualizan nuevamente los registros que se mostrarán.

4.3. Mensaje

A continuación se muestra el diagrama correspondiente al módulo del mensaje que debe aparecer en la LCD.

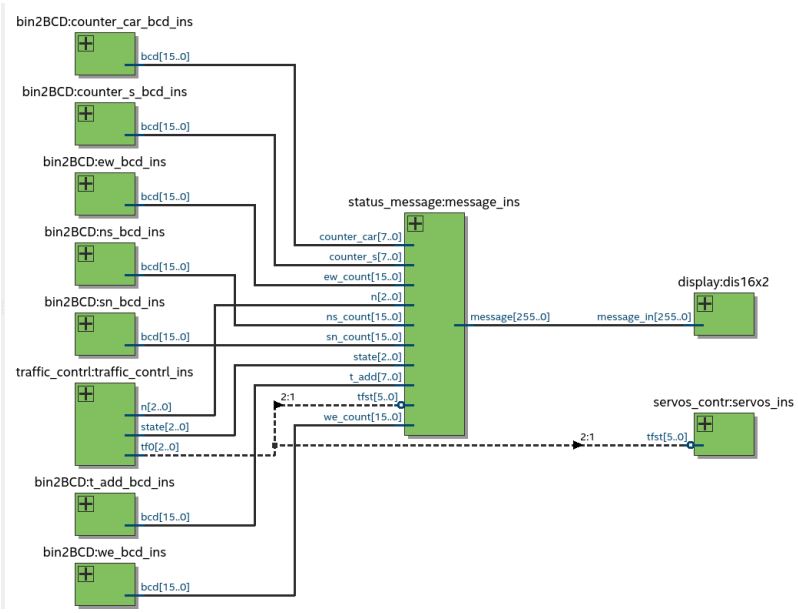

Figura 12: Diagramas de entradas y salidas del módulo para mensaje

Este módulo recibe como entradas los contadores netos totales por cada semáforo, el número de vehículos que se contaron en el estado de conteo, un contador en segundos para visualizar el progreso de la espera y asimismo el tiempo adicional que debe alcanzar este contador para volver a contar. Por otro lado, con el fin de mostrar información personalizada en el estado de «espera_aditiva» se recibe como entrada el valor de *tfst* y *state* del controlador de tráfico.

4.4. Implementación servomotor

Para implementar el servomotor se crearon dos módulos. El primero, cuyo diagrama se muestra a continuación, recibe un clock de $10KHz$, es decir, una señal con un periodo igual al de la señal PWM que se desea crear, y también recibe un clock de $10Hz$ que se encarga de cambiar el ángulo del servo. Por otro lado, una señal vel, que físicamente, consiste en un pulsador. La idea del pulsador es que, al ser oprimido este, produzca un *posedge* que es detectado como un aumento en la velocidad. La velocidad, internamente, estaría dada por un registro, y la función de la entrada vel sería, simplemente, aumentar este registro con el fin de cambiar la velocidad, pero realizando un procesamiento digital de señales.

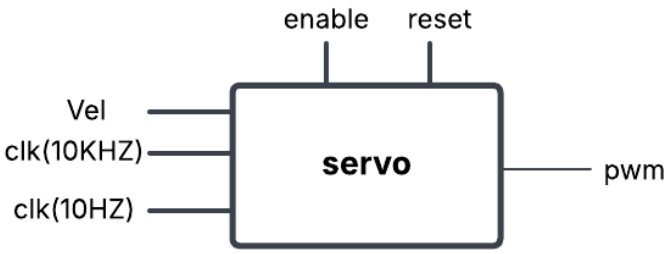

Figura 13: Módulo para la señal PWM

Por otro lado, se implementó un módulo superior, cuyo objetivo consiste en habilitar o deshabilitar los motores asociados a cada semáforo según el estado del mismo. Como se mencionó anteriormente, esta información está almacenada en la variable tfst, por lo tanto, esta será una de las variables del módulo. Se tienen 2 bits de velocidad porque se requiere como máximo controlar dos servomotores al mismo tiempo, y los 4 bits de pwm corresponde a la señal de control de cada servomotor.

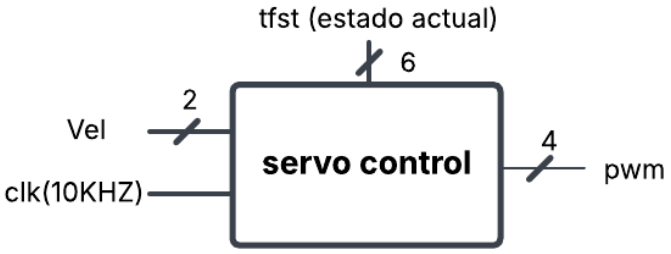

Figura 14: Módulo superior para servomotor

En el módulo del servo se tiene una máquina de estados encargada de cambiar el ángulo. Esta funciona con el clock de $10Hz$:

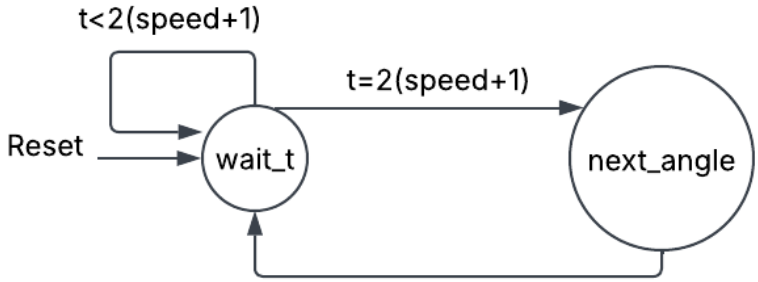

Figura 15: Diagrama de maquina de estados para control de velocidad del servomotor

Internamente se creó un contador de los flancos positivos del reloj de $10KHz$ y hasta que no se supere la condición de que el contador sea igual a $t = 2(speed + 1)$, no se cambiará el angulo del servomotor. En el momento que se cumple la condición, se aumenta un registro *angle_count*. Este contador es el indice de un banco de registros que contienen los ángulos seleccionados. Por otro lado, como el servomotor necesita de una señal con el mismo ancho de pulso de manera continua para confirmar su desplazamiento, se requirió utilizar otra máquina de estados que no interviniera el periodo de la señal. Esta máquina utiliza el clock de $10kHz$ para generar la señal de $20ms$.

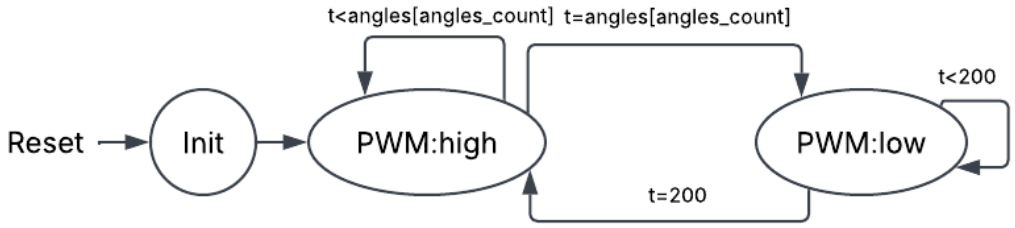

Figura 16: Diagrama de maquina de estados para control de PWM del servomotor

Como se mencionó en la sección previa del documento, se tiene un banco de registros que almacenan 3 tiempos en unidades de $100\mu s$ correspondientes a 3 ángulos distintos. Con el control del registro *angle_count*, la señal de ancho de pulso estará activa durante el tiempo correspondiente al ángulo deseado. Una vez se supere este tiempo, se realiza una transición para colocar en LOW la señal hasta alcanzar un tiempo de $200(100\mu s) = 20ms$ y nuevamente realizar un transición para colocar en $HIGH$ la señal durante el ángulo seleccionado.

4.5. Módulo superior

A continuación se presenta el módulo superior y sus conexiones con los módulos presentados anteriormente.

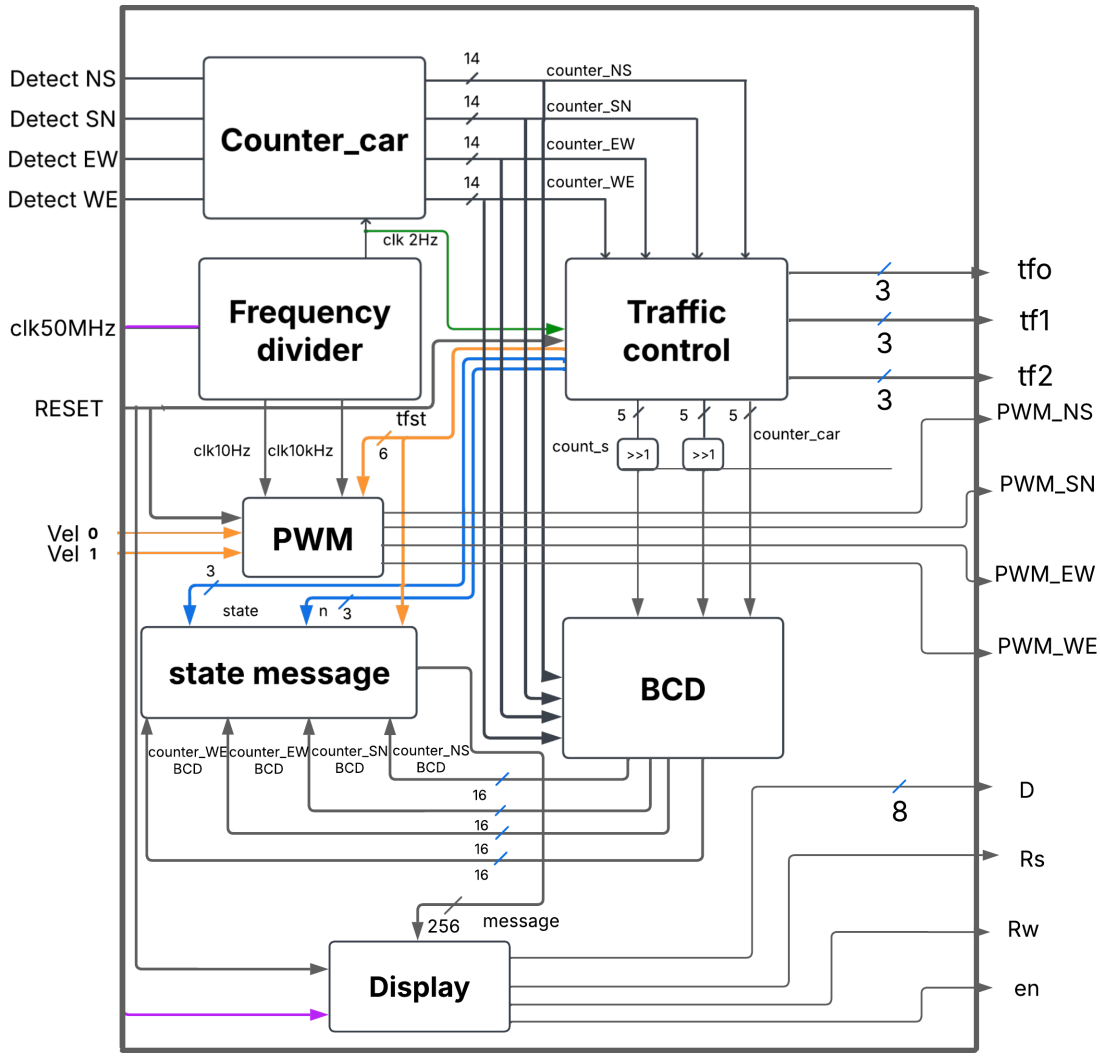

Figura 17: Módulo superior

Como se puede observar, a modo general, las entradas del sistema consisten en las señales detectadas por los diodos. Asimismo, hay tres botones adicionales: uno para el RESET, que reinicia el sistema a sus condiciones iniciales; vel0 y vel1, cuya función es aumentar o disminuir la velocidad de los servomotores.

Las señales captadas por los diodos son procesadas directamente por el módulo counter car, cuya función es guardar la información de cuántos carros pasan en determinado tiempo, de ahí que su otra entrada sea un clock del módulo frecuency divider. La información de cuántos carros pasan es procesada luego por el módulo de control de tráfico, es decir, es este módulo el que determina el tiempo adicional por flujo de carros.

En este punto es importante señalar la importancia del módulo frecuency divider, cuya función consiste en proveer a todo el sistema de distintas frecuencias de clock, dependiendo la necesidad del módulo.

Por otro lado, para desplegar la información en el display es necesario que, primero, se genere un mensaje según el estado en el que se encuentra el sistema. El módulo encargado de recibir la información de los carros contados y generar dicho mensaje es el state message. Asimismo, el módulo auxiliar de este es el BCD, cuya función es separar

los dígitos del conteo de carros, pues, es necesario desplegarlos uno por uno en la LCD.

Por último, el módulo display se encarga de recibir el mensaje generado por el módulo state message y desplegarlo en la LCD por medio de la salida D de 8 bits.

5. Pruebas del funcionamiento

Una vez terminado el proceso de diseño y desarrollo del proyecto, pasamos a su implementación y comprobamos el correcto funcionamiento para cada uno de los semáforos. En la presentación del proyecto se registró el contador de çarros"por los diodos emisor y receptor con el respectivo tiempo que el sistema añade al ciclo. Estos datos los organizamos en las siguientes tablas:

Semáforos NS y SN	
C_car	T [s]
11	10
9	10
3	4
1	4
2	4
25	14
7	6
5	6
2	4

Semáforo EW	
C_car	T [s]
11	10
24	14
6	6
5	6
3	4

Semáforo WE	
C_car	T [s]
7	6
5	6
3	4
11	10
23	14

Considerando el diseño realizado, el tiempo adicional asociado a cada conteo fue el indicado.

6. Tamaño de la solución

A continuación se muestran los recursos utilizados por la *FPGA* con el análisis de *quartus*.

```
Flow Status
Successful - Tue Feb 11 22:05:31 2025
Quartus Prime Version
23.1std.1 Build 993 05/14/2024 SC Lite Edition
Revision Name
top
Top-level Entity Name
top
Family
Cyclone IV E
Device
EP4CE10E22C8
Timing Models
Final
Total logic elements
```

```
2,699 / 10,320 ( 26 % )
Total registers                      467
Total pins                     32 / 92 ( 35 % )
Total virtual pins                   0
Total memory bits
0 / 423,936 ( 0 % )
Embedded Multiplier 9-bit elements 0 / 46 ( 0 % )
Total PLLs                         0 / 2 ( 0 % )
```

En términos del circuito externo, el proyecto utiliza una gran cantidad de pines debido a que utiliza un display, y también tiene 9 salidas para diodo led. Por otro lado se requieren 4 pines para el control pwm y las señales de entrada de detección. Por otro lado, debido a que se tienen mensajes variables en la pantalla, se requiere un uso de gran cantidad de registros. Además, por la lógica combinacional y secuencial asociado a cada módulo, se está utilizando una gran parte de unidades lógicas.

7. Conclusiones

- A la hora de implementar distintos tipos de clocks en un mismo proyecto, pero para distintos módulos, es conveniente que la entrada de todos ellos sea una misma, es decir, que estén sincronizados según un mismo clock de origen.
- Las señales percibidas por los sensores captan variaciones muy pequeñas en un simple conteo, de manera que es necesario implementar una lógica comparativa, en lugar de una simplemente aritmética, es decir,
- La metodología de desarrollo modular y escalable mantenida a lo largo del proyecto nos permitió que a la hora de realizar algún cambio estos no tuvieran un impacto negativo en el flujo de trabajo. Así como también nos facilitó la comunicación entre nosotros y la organización de las diferentes funciones realizadas para el dispositivo.
- La cantidad de unidades lógicas utilizadas en el proyecto utilizó casi un tercio de la capacidad de la FPGA. Asimismo, la cantidad de pines utilizados superó este porcentaje. Estos datos indican una tendencia de uso de recursos grande en comparación con la envergadura y tipo de proyecto.

8. Trabajos Futuros

Gracias a la naturaleza modular del desarrollo que se mantuvo a lo largo de todo el proyecto (metodología sobre la cual se hizo hincapié en los laboratorios debido a sus ventajas), es posible pensar en formas de escalar el proyecto para realizar funciones más complejas o simplemente un mayor manejo de datos, vías y semáforos.

Un escenario posible es el de añadir funciones de memoria, para después recopilar datos y con esa información apoyar estrategias de movilidad para la ciudad. Esta ventaja en la adquisición de datos a su vez podría realimentar al equipo o desarrolladores del sistema para analizar y descubrir nuevas formas de hacerlo más eficiente. Estos datos podrían correlacionar diferentes variables como hora del día, tipos de autos (tráfico pesado, ligero), temperatura o clima (días de lluvia o soleados) y así tener un espectro amplio de información que alimentara a los modelos de movilidad y estadísticos de la ciudad.

Sin embargo, para lograr esas funciones adicionales también sería necesario cambiar el sistema de detección de paso vehicular (en nuestro prototipo, el empleado por medio de diodos receptor y emisor) por uno mucho más sofisticado, como cámaras programables mediante modelos de IA. Un ejemplo de este tipo de tecnología es el sensor TrafiCam AI desarrollado por la empresa Teledyne Flir.

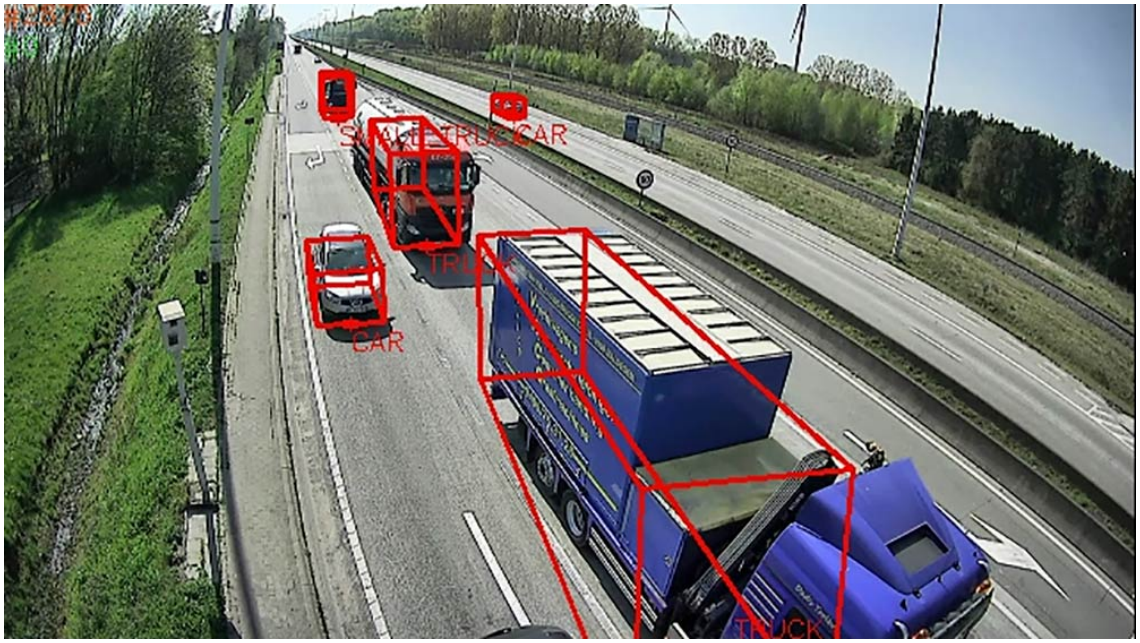

Figura 18: Ejemplo de funcionamiento de cámaras de detección de vehículos mediante inteligencia artificial. En este caso TrafiCam AI.