UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

ESCUELA DE INGENIERÍAS INDUSTRIALES

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Diseño de un semáforo a escala con componentes de bajo coste para utilizarlo en prácticas de Automatización Industrial

Design of a scaled down traffic lights with low-cost components to be used in Industrial Automation practices

Realizado por

Daniel Costa da Silva

Tutorizado por

Vicente Manuel Arévalo Espejo

Área de Conocimiento

Ingeniería de Sistemas y Automática

Departamento

Ingeniería de Sistemas y Automática

UNIVERSIDAD DE MÁLAGA

MÁLAGA, NOVIEMBRE DE 2024

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Resumen

Una gran parte de la automatización en la industria se hace mediante autómatas programables industriales o PLCs, cuyo lenguaje de programación es común y está estandarizado. Sin embargo, el elevado coste de estos dispositivos industriales limita su uso fuera del Centro, e incluso dentro, dado que el número de equipos es muy pequeño. Esto nos lleva a buscar una alternativa más económica.

Por eso, en este proyecto se ha diseñado y fabricado un semáforo a escala que se ha utilizado para simular un cruce semafórico realista. Para ello se ha conectado el cruce semafórico a un Arduino Mega, usándolo como pasarela ModBus que a su vez se controla desde un ordenador emulando el funcionamiento de un PLC.

Finalmente, se han diseñado varias prácticas que utilizan uno o varios semáforos con el fin de poner en práctica los conocimientos adquiridos en clase sobre programación de automatismos.

Palabras clave:
Semáforo, Arduino, *perfboard*, Modbus, CoDeSys, GRAFCET, SFC, electrónica, automática, PLC, autómata, modelo, escala, cruce semafórico.

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Abstract

A large part of automation in industry is done using industrial programmable logic controllers or PLCs, whose programming language is common and standardized. However, the prohibitive cost of these industrial devices limits their use both outside and inside the Center, since the number of devices is very small. This leads us to look for a cheaper alternative.

For this reason, in this project a scaled down traffic light has been designed and manufactured and has been used to simulate a realistic traffic light interchange. To accomplish this, the interchange has been connected to an Arduino Mega, using it as a ModBus gateway which is controlled from a computer emulating the operation of a PLC.

Finally, several practices have been designed to use one or more traffic lights to put into practice the knowledge acquired in class on programming PLCs.

Keywords:
Traffic light, Arduino, perfboard, Modbus, CoDeSys, GRAFCET, SFC, electronic, automation, PLC, controller, model, scaled down, traffic interchange.

UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Índice

Resumen 1

Abstract 1

Índice 1

Introducción 3

1.1 Motivación 3

1.2 Objetivos 3

1.3 Requisitos de diseño 4

1.4 Fases del proyecto 6

1.5 Herramientas y componentes 6

1.5.1 Introducción 6

1.5.2 Software 6

1.5.3 Hardware 8

1.6 Organización del documento 9

Diseño electrónico 11

2.1 Introducción 11

2.2 Arduino Mega 2560 11

2.3 Semáforo 15

2.3.1 LEDs 15

3.3.2 Pulsador 16

2.3.3 Bocina 16

2.3.4 Resumen 18

Desarrollo de prototipos 21

3.1 Introducción 21

3.2 Primer prototipo 21

3.3 Segundo prototipo 23

3.4 Prototipo final 24

3.4.1 Placa *perfboard* 24

3.4.2 Semáforo 25

3.4.3 Base de semáforo 28

3.4.4 Montaje 28

4.5 Presupuesto y tiempos de montaje 32

Prácticas 35

4.1 Introducción 35

4.2 Practica 1 - Introducción a CoDeSys 35

4.2.1 Introducción 35

4.2.2 Pasos previos en CoDeSys 35

4.2.3 "Hola Mundo" en CoDeSys 37

4.2.4 Ventana de visualización virtual 40

4.3 Practica 2 - Programación de semáforo para vehículos 43
4.3.1 Introducción 43
4.3.2 Semáforo 43
4.3.3 Arduino Mega 2560 45
4.3.4 Conexión ModBus 46
4.3.5 Semáforo completo 49
4.4 Practica 3 - Lectura de señales, ramificaciones y acciones activas 50
4.4.1 Introducción 50
4.4.2 Lectura de señales 50
4.4.3 Acciones activas 51
4.4.4 Ramificaciones 52
4.5 Practica 4 - Programación de cruce semafórico 53
4.5.1 Introducción 53
4.5.2 Cruce semafórico 53

Conclusiones y Mejoras 55
5.1 Conclusiones 55
5.2 Mejoras 57

Bibliografía 59

Código fuente 61
7.1 Código de Arduino del primer prototipo 61
7.2 Código de comunicación ModBus 64
7.3 Código de CoDeSys para cruce semafórico completo 67

Planos 75
8.1 Plano de Perfboard 76
8.2 Plano de Placa de LEDs de Coches 77
8.3 Plano de Placa de LEDs de Peatones 78
8.4 Plano de Cuerpo Principal de Semáforo 79
8.5 Plano de Base de Semáforo 80
8.6 Plano de Placa Trasera de Semáforo 81

1

Introducción

1.1 Motivación

Una gran parte de la automatización en la industria se hace mediante autómatas programables industriales o PLCs, cuyo lenguaje de programación es común y está estandarizado. Sin embargo, el elevado coste de estos dispositivos industriales limita su uso fuera del Centro, e incluso dentro, dado que el número de equipos es muy pequeño. Esto nos lleva a buscar una alternativa más económica.

1.2 Objetivos

Actualmente, el acceso a prácticas con PLCs en el Grado en Ingeniería en Tecnologías Industriales en la Escuela de Ingenierías Industriales de la Universidad de Málaga está reservado a la asignatura de Sistemas Robotizados, perteneciente al módulo de competencias específicas de electrónica industrial (METRON) [1]. Este módulo es uno de los 3 módulos de competencias especificas con los que cuenta el plan de estudios del Grado en Ingeniería en Tecnologías Industriales siendo de requisito indispensable para completar 2 de los 8 posibles itinerarios del Grado o se puede cursar de manera opcional en el itinerario de organización, cursándose en cualquier caso durante el cuarto curso.

La alternativa presentada en este proyecto tendrá un uso esperado en las clases prácticas de Automática, esta asignatura, o asignaturas con distintas denominaciones pero con contenidos lectivos similares, se encuentra presente en todos los Grados impartidos en la Escuela de Ingenierías Industriales de la Universidad de Málaga, siendo de carácter obligatorio en casi todos a excepción de los Grados en Ingeniería de la Energía y en Ingeniería de Organización Industriales [2], en los cuales sería una asignatura optativa. En el caso del Grado en Ingeniería en Tecnologías Industriales, y en la mayoría de los demás grados, es una asignatura que se imparte en el segundo curso.

En esta asignatura se imparten conocimientos de GRAFCET, un modelo de representación gráfica en el cual se puede describir el control de máquinas mediante etapas y transiciones y que cuenta con su lenguaje de programación equivalente denominado como *Sequential Function Chart (SFC)* en la norma IEC 61131-3:2013 [3].

El objetivo principal de este proyecto consiste en desarrollar una solución *hardware-software* de bajo coste que permita desarrollar prácticas introductorias de Automatización industrial sin tener que recurrir a costosos sistemas. Un objetivo secundario de este proyecto es el diseño de un conjunto de prácticas tutorizadas que ayuden al alumno a comprender el funcionamiento de la planta diseñada y programarla con *software* comercial ampliamente utilizado en la Industria.

Para terminar, se ha de mencionar que la planta que se ha propuesto desarrollar consiste en un semáforo a escala con un funcionamiento que imite el comportamiento de un semáforo real con la siguiente funcionalidad: luces para vehículos, luces para peatones, luz de giro a derecha, bocina para avisar de verde para peatones y un pulsador de reducción de cuenta de rojo para vehículos.

1.3 Requisitos de diseño

En este apartado se comentarán los requisitos de diseño del *hardware* que se desarrollará en este proyecto.

- Se utilizará 1 único Arduino Mega 2560. Con el fin de abaratar costes se requiere que con 1 único microcontrolador se pueda controlar el cruce semafórico completo.

- Se alimentará todo desde el puerto USB del Arduino. Para facilitar el uso del cruce semafórico se evitará usar fuentes de alimentaciones externas.

- El Arduino se usará exclusivamente como tarjeta de comunicación, haciendo que su reemplazo por cualquier otro dispositivo de E/S digitales con capacidad de comunicación por ModBus no requiera rediseñar la electrónica.

- Se podrá usar el cruce semafórico en las clases docentes prácticas sin herramientas adicionales como destornilladores, soldadores, etc.

- Cada cruce tendrá 4 semáforos. Con el fin de simular un cruce semafórico realista se usará 1 semáforo para cada vía que se incorpora el cruce.

- Los semáforos tendrán un diseño realista con un tamaño adecuado. Con el fin de acotar el tamaño del cruce para facilitar su almacenaje, transporte y/o uso se requiere que los semáforos estén en una escala entre 1:16 y 1:32 (ver figura 1.1).

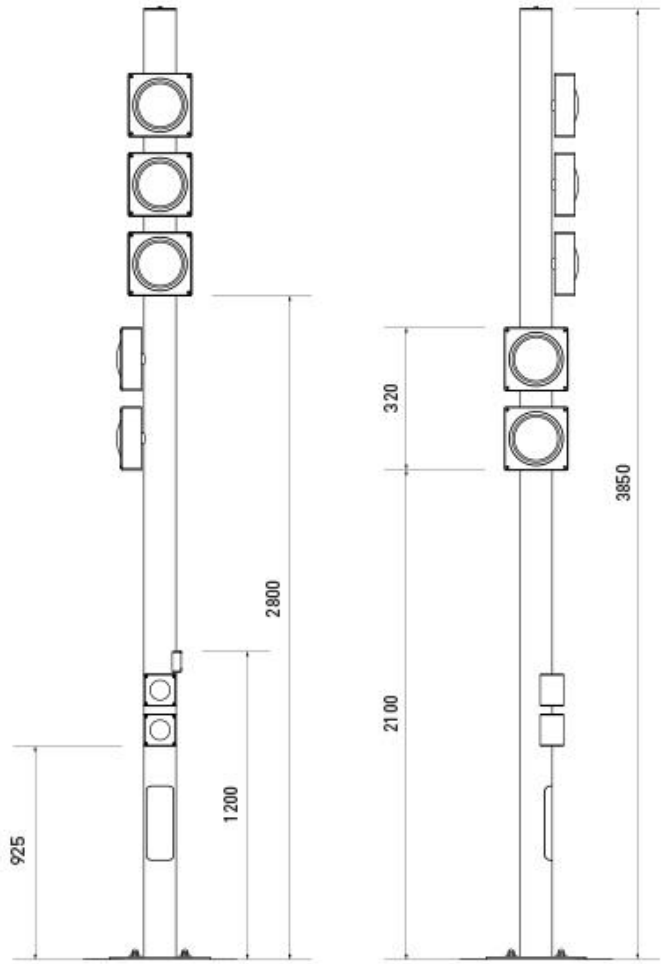


Figura 1.1 - Ejemplo de semáforo moderno [1].

- Los semáforos tendrán 5 elementos:
  - Señales luminosas para vehículos.
  - Señal luminosa para giro de vehículos.
  - Señales luminosas para peatones.
  - Señal acústica.
  - Pulsador.

- Las señales luminosas para vehículos estarán compuestas por las señales roja, ámbar y verde, dispuestas en vertical y en el mismo orden empezando por arriba. Físicamente se situarán en lo alto del semáforo.

- La señal de giro se colocará a la derecha de las señales para vehículos y, en caso de ser posible, tendrá un indicador de giro a derecha.

- Las señales luminosas para peatones estarán compuestas por las señales roja y verde, dispuestas en vertical y en el mismo orden empezando por arriba. Físicamente se situarán a media altura en el semáforo y giradas 90 grados en sentido horario con relación a las señales para vehículos.

- La señal acústica se diseñará para simular un avisador acústico para personas con discapacidad visual.

- El pulsador se diseñará para acortar el tiempo de prohibición de paso para los peatones.

1.4 Fases del proyecto

El proyecto se ha dividido en las siguientes fases:

- **Fase 1.** Obtención de materiales, herramientas de desarrollo y adquisición de los conocimientos necesarios:
  - Selección de componentes electrónicos necesarios para construir un semáforo a escala.
  - Elección de los programas a usar para el desarrollo del proyecto.
  - Aprendizaje de los lenguajes de diseño y programación de automatismos (GRAFCET/SFC, respectivamente) utilizados habitualmente en clase.

- **Fase 2.** Desarrollo del modelo a escala del semáforo:
  - Diseño del sistema de comunicación entre el Arduino y el ordenador.
  - Impresión y montaje de modelo 3D.
  - Integración completa del sistema.

- **Fase 3.** Diseño de prácticas y preparación de la memoria:
  - Redacción de diferentes supuestos prácticos que utilicen 1, 2 o múltiples semáforos.
  - Diseño e implementación de los automatismos para que los semáforos funcionen de acuerdo con lo establecido en los supuestos planteados.
  - Documentación del trabajo realizado durante el proyecto.

1.5 Herramientas y componentes

1.5.1 Introducción

En este apartado se comentará brevemente el *software* y *hardware* empleados en el desarrollo del proyecto además de los componentes electrónicos que se emplearán en el semáforo.

1.5.2 Software

1.5.2.1 Arduino IDE

El entorno de desarrollo integrado (IDE) de Arduino [5] es un *software* de código abierto que permite la programación de los microcontroladores de la familia Arduino con un lenguaje basado en C++ con una capa de funciones que facilitan el acceso a las capacidades de los microcontroladores que se encuentran en las placas de desarrollo a la par que facilitan la legibilidad del código.

Además de las librerías propias de la familia Arduino que ya vienen incluidas en la IDE, una de las grandes ventajas que presenta la IDE es la gran variedad de librerías publicadas y mantenidas por la comunidad para todo tipo de tareas.

1.5.2.2 CoDeSys

CoDeSys [6] es un entorno de desarrollo para la programación de controladores conforme con el estándar internacional IEC 61131-1.

Para el desarrollo de este proyecto se ha decidido usar CoDeSys ya que nos permite simular un PLC en un ordenador con Windows. Carece de capacidades en tiempo real y de pines de entrada/salida, teniendo que recurrir a la comunicación con placas externas encargadas de esto último.

CoDeSys cuenta con varias licencias, dando un alto grado de flexibilidad en cuanto a los costes necesarios para utilizarlo, empezando por la licencia gratuita que permite usar todo el entorno de programación, permitiendo el uso de un PLC virtual durante 2 horas seguidas de cada vez, haciéndola perfecta para el desarrollo de prototipos.

1.5.2.3 SolidWorks

AUTODESK SolidWorks es uno de los programas de diseño de piezas mecánicas más utilizados en entornos profesionales. Es un programa muy versátil ya que no solo tiene capacidades de diseño 3D, sino que cuenta con librerías de simulación estructural, de fluidos, eléctrica, etc.

Aunque la licencia de SolidWorks es muy costosa, 3480 €/año, la Escuela de Ingenierías Industriales proporciona a los alumnos licencias de estudiantes con las cuales se tiene acceso a las capacidades del programa para fines educativos.

En este caso se ha prescindido de hacer uso de las capacidades de simulación, limitándose en el proyecto a diseñar los modelos y exportarlos para a continuación prepararlos para la impresión 3D.

1.5.2.4 PrusaSlicer

PrusaSlicer [7] es un programa de código abierto publicado por la compañía de impresoras 3D, Prusa3D que contiene decenas de perfiles probados para distintos filamentos y resinas para impresoras 3D de Prusa además de haber infinidad de perfiles creados por la comunidad para impresoras 3D de otras marcas.

Este programa permite convertir un modelo, con extensión STL, en una serie de capas indicando así a la impresora que tiene que imprimir en cada capa.

1.5.3 Hardware

1.5.3.1 Arduino Mega 2560

El Arduino Mega 2560 es una placa de desarrollo de la empresa Arduino. Esta placa es la segunda más potente de la misma contando con un gran número de pines de entrada y salida, varias implementaciones hardware de protocolos de comunicación distintos y, al estar dentro de la familia Arduino, cuenta con un entorno de desarrollo de *software* propio que facilita el desarrollo de aplicaciones.

Características técnicas:

- Microcontrolador de 8 bits ATmega2560.
- 54 entradas/salidas digitales.
- 15 salidas PWM.
- 16 entradas analógicas.
- puertos de comunicación serial por *hardware.*
- Cristal oscilador de 16 MHz.
- Conexión USB.
- Conector de corriente.
- Salida ICSP.
- Pulsador de *reset.*
- Compatible con placas de expansión diseñadas para Uno, Duemilanove y Diecimila.

1.5.3.2 Componentes electrónicos

Se emplean una serie de componentes electrónicos, entre ellos:

- **Resistencias.** Las resistencias son parte de los componentes básicos de la electrónica, junto a los condensadores y los inductores. En el caso de las resistencias reducen la intensidad que circula por un circuito.

- **Condensadores.** Como se comentó en las resistencias, los condensadores son componentes básicos. En este caso son dispositivos que almacenan energía eléctrica pudiéndose usar como dispositivos de filtrado o de histéresis.

- **LEDs.** Los diodos emisores de luz (LED) son dispositivos electrónicos compuestos por un material semiconductor que emite luz cuando se aplica el voltaje correcto en sus extremos.

- **Pulsador.** Los pulsadores son dispositivos eléctricos que permiten que pase o no corriente según estén pulsados o no.

- **Bocina.** Las bocinas son dispositivos que permiten emitir un sonido cuando activadas, se suelen usar en aplicaciones industriales como señales acústicas.

- **Circuitos integrados.** Los circuitos integrados son pequeños *chips* que cuentan con la electrónica necesaria para tareas concretas, en el caso de este proyecto se ha usado un circuito integrado temporizador capaz de generar una señal periódica cuadrada.

1.5.3.3 Impresora Prusa i3 MK3s

La Prusa i3 MK3s es una impresora 3D de la marca Prusa, que se dedica al desarrollo de impresoras y programas de código abierto.

Dentro de las características técnicas tenemos:

- Volumen de impresión de 25x21x21 cm.
- Altura de capa de 0.05 a 0.35 mm.
- Boquilla de 0.4 mm.
- Filamento de 1.75 mm.
- Compatible con PLA, PETG, ASA, ABS, PC, CPE, PVA/BVOH, PVB, HIPS, PP, Flex, nGen, Nylon, *Carbon filled, Woodfill* y otros materiales.
- Velocidad máxima de desplazamiento de más de 200 mm/s.
- Temperatura máxima de la boquilla de 300 ºC.
- Temperatura máxima de la base de 120 ºC.
- Extrusor de accionamiento directo, engranajes Bontech, hotend V6.
- Superficie de impresión con láminas de acero magnético desmontable con diferentes acabados de superficie y base calefactable con compensación de esquinas frías.
- Dimensiones totales de 500x550x400 mm.
- Peso 7 kg.
- Consumo de 80 W imprimiendo con PLA y 120 W con ABS.

1.6 Organización del documento

En este apartado contiene una explicación breve de la estructura del documento, junto a una pequeña descripción de cada capítulo.

- **1. Introducción:** Descripción de la motivación, objetivos y planificación del proyecto. Introducción de los componentes, herramientas y programas necesarios para la realización del proyecto.
- **2. Diseño electrónico:** Explicación de las decisiones de diseño de la electrónica del proyecto basada en criterios industriales.
- **3. Desarrollo de prototipos:** Seguimiento del desarrollo de prototipos desde la validación de componentes hasta el montaje final.
- **4. Prácticas:** Propuestas de prácticas a desarrollar por los alumnos en las cuales podrán poner en práctica los conocimientos adquiridos sobre GRAFCET.
- **5. Conclusiones y futuras mejoras:** Se comentarán las conclusiones personales y posibles vías de desarrollo para mejorar los modelos.
- **6. Bibliografía:** Resumen de referencias a otros documentos.
- **7. Códigos fuente:** Códigos de Arduino y programa de CoDeSys desarrollados durante el proyecto.
- **8. Planos:** Planos de las piezas diseñadas para el proyecto.

2

Diseño electrónico

2.1 Introducción

En este capítulo se describirán los pasos seguidos en la elección de componentes electrónicos utilizados en el diseño del semáforo. También se describen las comprobaciones necesarias a la seguridad (en lo relativo a que no se superen sus límites de trabajo) de todos los componentes.

2.2 Arduino Mega 2560

Puesto que uno de los requisitos de diseño establecidos al inicio del proyecto era que el Arduino alimentase los cuatro semáforos del cruce, se ha empezado por analizar el diseño interno del Arduino Mega 2560 para asegurarnos que en ningún momento habrá peligro de sobrepasar los límites de potencia de este. En la figura 2.1 se puede observar la intensidad máxima que puede pasar por cada componente del Arduino Mega 2560.

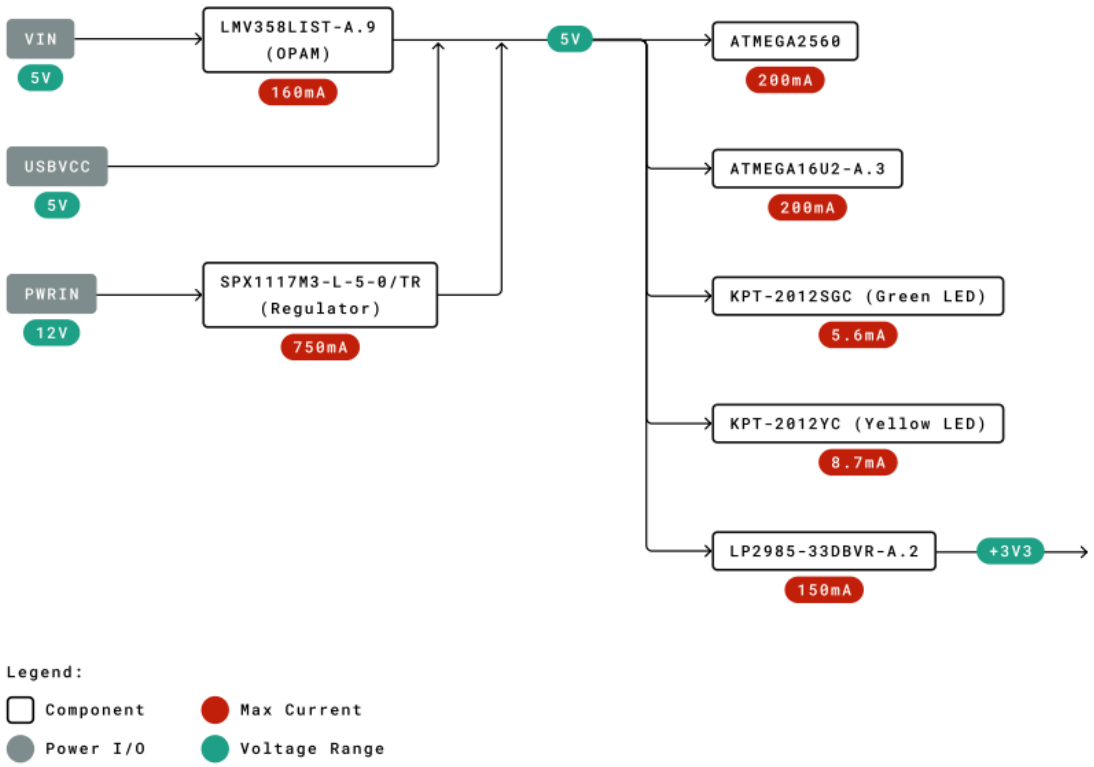


Figura 2.1 - Árbol de potencia de Arduino Mega 2560 [8].

La comunicación con el ordenador se hará mediante el puerto Serial que está integrado en la conexión USB, usando el protocolo ModBus por lo que se ha empezado por ver el consumo máximo permitido por el estándar de USB 2.0 que como se puede observar en la figura 2.2 es de 500 mA.

Parameter	Symbol	Conditions	Min.	Max.	Units
Supply Voltage:					
High-power Port	V_{BUS}	Note 2, Section 7.2.1	4.75	5.25	V
Low-power Port	V_{BUS}	Note 2, Section 7.2.1	4.40	5.25	V
Supply Current:					
High-power Hub Port (out)	I_{CCPRT}	Section 7.2.1	500		mA
Low-power Hub Port (out)	I_{CCUPT}	Section 7.2.1	100		mA
High-power Function (in)	I_{CCHPF}	Section 7.2.1		500	mA
Low-power Function (in)	I_{CCLPF}	Section 7.2.1		100	mA
Unconfigured Function/Hub (in)	I_{CCINIT}	Section 7.2.1.4		100	mA
Suspended High-power Device	I_{CCSH}	Section 7.2.3; Note 15		2.5	mA
Suspended Low-power Device	I_{CCSL}	Section 7.2.3		500	µA

Figura 2.2 - Especificaciones eléctricas del estándar de USB 2.0 [9].

El chip ATMEGA2560 es el chip principal en el cual correrá el programa de Arduino y cuenta con un consumo máximo de operación de 14 mA (ver figura 2.3).

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
I_{CC}	Power Supply Current[5]	Active 1MHz, V_{CC} = 2V (ATmega640/1280/2560/1V)		0.5	0.8	mA
		Active 4MHz, V_{CC} = 3V (ATmega640/1280/2560/1L)		3.2	5	
		Active 8MHz, V_{CC} = 5V (ATmega640/1280/1281/2560/2561)		10	14	
		Idle 1MHz, V_{CC} = 2V (ATmega640/1280/2560/1V)		0.14	0.22	
		Idle 4MHz, V_{CC} = 3V (ATmega640/1280/2560/1L)		0.7	1.1	
		Idle 8MHz, V_{CC} = 5V (ATmega640/1280/1281/2560/2561)		2.7	4	
	Power-down mode	WDT enabled, V_{CC} = 3V		<5	15	µA
		WDT disabled, V_{CC} = 3V		<1	7.5	

Figura 2.3 - Especificaciones eléctricas del chip ATmega2560 [10].

El chip ATMEGA16U2-A.3 es el encargado de la comunicación entre el Arduino y el puerto USB. Al estar usando el Arduino como pasarela ModBus, este chip también estará en funcionamiento continuo.

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted) (Continued)

Symbol	Parameter	Condition	Min.[5]	Typ.	Max.[5]	Units
R_{PUDP}	USB D+ Internal Pull-Up	Idle mode	900		1500	Ω
		Streaming mode	1425		3090	Ω
I_{CC}	Power Supply Current[6]	Active 8 MHz, V_{CC} = 3V regulator disabled		4	6	mA
		Active 16 MHz, V_{CC} = 5V regulator enabled		13.5	21	mA
		Idle 8 MHz, V_{CC} = 3V regulator disabled		0.8	1.2	mA
		Idle 16 MHz, V_{CC} = 5V regulator enabled		3.2	4.0	mA
	Power-down mode	WDT disabled, regulator disabled,V_{CC} = 3V		5	10	µA
		WDT enabled, regulator disabled,V_{CC} = 3V		10	15	µA
		WDT, BOD, regulator enabled, Vcc = 5V		40	65	µA
	Standby mode - 8MHZ XTAL	WDT disabled, BOD Enabled, regulator disabled, Vcc = 3V		250		µA
		WDT disabled, BOD, regulator enabled, Vcc = 5V		350		µA

Figura 2.4 - Especificaciones eléctricas del chip ATmega16U2 [11].

Así mismo, en la figura 2.4 se puede observar el consumo máximo de operación del chip ATmega16U2 de 21 mA.

El LED verde (KPT-2012SGC) indica que el Arduino está encendido y consume 5.6 mA y el LED amarillo (KPT-2012YC) indica que el Arduino está comunicando a través del puerto USB y consume 8.7 mA.

Finalmente, el LP2985-33DBVR-A.2 es un regulador de tensión para la salida de 3.3V que no se usará.

Although each I/O port can source more than the test conditions (20mA at VCC = 5V, 10mA at VCC = 3V) under steady state conditions (non-transient), the following must be observed:
ATmega1281/2561:
1)The sum of all IOH, for ports A0-A7, G2, C4-C7 should not exceed 100mA.
2)The sum of all IOH, for ports C0-C3, G0-G1, D0-D7 should not exceed 100mA.
3)The sum of all IOH, for ports G3-G5, B0-B7, E0-E7 should not exceed 100mA.
4)The sum of all IOH, for ports F0-F7 should not exceed 100mA.
ATmega640/1280/2560:
1)The sum of all IOH, for ports J0-J7, G2, A0-A7 should not exceed 200mA.
2)The sum of all IOH, for ports C0-C7, G0-G1, D0-D7, L0-L7 should not exceed 200mA.
3)The sum of all IOH, for ports G3-G4, B0-B7, H0-H7 should not exceed 200mA.
4)The sum of all IOH, for ports E0-E7, G5 should not exceed 100mA.
5)The sum of all IOH, for ports F0-F7, K0-K7 should not exceed 100mA.

Figura 2.5 - Especificaciones de consumo de pines de ATmega2560 [10].

Continuando el análisis se ha buscado la corriente máxima que podría consumir cada *pin* que como se ve en la figura 2.5 varia no solo según la corriente singular del *pin* sino también según el consumo de algunos grupos de pines.

Descripción	Numero de Pines	Consumo Máximo	Consumo Máximo Medio
Cada pin	1	20 mA	20 mA
J0-J7, G2, A0-A7	17	200 mA	11.764 mA
C0-C7, G0-G1, D0-D7, L0-L7	26	200 mA	7.692 mA
G3-G4, B0-B7, H0-H7	18	200 mA	11.111 mA
E0-E7, G5	9	100 mA	11.111 mA
F0-F7, K0-K7	16	100 mA	6.25 mA

Tabla 2.1 - Tabla de requisitos de consumo de pines de chip ATmega2560.

Se puede ver en la tabla 2.1 que el caso más desfavorable seria que los dieciséis pines que más consumen estuvieran en los pines internos F0-F7 y K0-K7, por lo que en futuros análisis se tendrá esto en cuenta para asegurarnos del buen funcionamiento y seguridad eléctrica del proyecto.

Componente	Consumo
ATmega2560	14 mA
ATmega16u2-A.3	21 mA
KPT-2012SGC	5,6 mA
KPT-2012YC	8.7 mA
Total	**49.3 mA**

Tabla 2.2 - Tabla resumen de consumos de corriente básicos.

En resumen, como se puede ver en la tabla 2.2, el chip ATmega2560 consume 14 mA de 200 mA de máximo con los que puede trabajar quedando 186 mA libres para componentes que se alimenten desde los pines de este. Así mismo, el Arduino tendrá un consumo básico de 49,3 mA de 500 mA de máximo que puede alimentar la interfaz USB 2.0, quedando libre un total de 450,7 mA.

2.3 Semáforo

Como se ha comentado en el capítulo introductorio, el objetivo principal de este proyecto es diseñar un cruce semafórico con 4 semáforos donde cada semáforo se desea que tenga las siguientes características:

- Luces indicadoras para coches: roja, ámbar y verde.
- Luz indicadora de giro: ámbar.
- Luces indicadoras para peatones: roja y verde.
- Pulsador para adelantar el verde para los peatones.
- Bocina indicadora de verde para peatones.

En total cada semáforo contará con 6 LEDs, 1 pulsador y 1 bocina.

2.3.1 LEDs

Los formatos de LEDs más comunes son de 3 y 5 mm de diámetro, para este proyecto se ha elegido trabajar con LEDs de 5 mm lo que permitirá hacer modelos ligeramente más grandes sin que se pierda las proporciones entre los distintos elementos sin requerir varios LEDs para cada señal luminosa del semáforo.

De la misma forma, se ha intentado acortar la posible lista de candidatos según criterios de consumo, difusividad y color del encapsulado, habiéndose elegido, finalmente, los LEDs de bajo consumo de AVAGO TECHNOLOGIES, más específicamente los modelos HLMP-4700, HLMP-4719 y HLMP-4740, para los colores rojo, ámbar y verde respectivamente.

Estos LEDs están diseñados para un funcionamiento típico consumiendo 2 mA, pudiendo llegar hasta 7 mA en trabajo continuo. Además, cada LED viene con un encapsulado difuso del mismo color que la luz del LED.

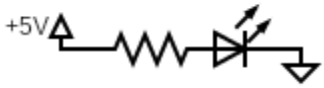


Figura 2.6 - Circuito típico de LED.

Para acompañar cada uno de estos LEDs hay que colocar una resistencia, como se ve en la figura 2.6, con la cual se puede controlar la corriente que pasa por el mismo.

$$R = \frac{V_{cc} - V_F}{I_F} \quad (1)$$

Siendo V_{cc} el voltaje suministrado en el *pin*, 5 V en este caso, R la resistencia elegida, I_F la corriente que pasa en el circuito y V_F el salto de tensión en el LED mientras está emitiendo luz, pudiéndose ver los valores de esto último para cada color de LED en la figura 2.7.

Electrical/Optical Characteristics at $T_A = 25°C$

Symbol	Description	T-1 3/4	T-1	Min.	Typ.	Max.	Units	Test Conditions
V_F	Forward Voltage	4700	1700		1.7	2.0	V	2 mA
		4719	1719		1.8	2.5		
		4740	1790		1.9	2.2		

Figura 2.7 - Características eléctricas de los LEDs [12].

El caso más desfavorable sería con la tensión de salto más baja que se encontraría en el LED rojo con una tensión típica de 1.7 V, sustituyendo la tensión de alimentación por 5 V y la intensidad de paso en 2 mA obtenemos que la resistencia mínima seria de 1650 Ω cuyo valor normalizado inmediatamente superior es 1.8 kΩ siendo esta la resistencia finalmente elegida y dando los consumos que se ven en la tabla 3.3.

Referencia	Color	Consumo
HLMP-4700	Rojo	1,833 mA
HLMP-4719	Ámbar	1,778 mA
HLMP-4740	Verde	1,722 mA

Tabla 2.3 - Consumo en cada LED.

3.3.2 Pulsador

Para el pulsador se ha decidido colocar una resistencia de *pull-up* (ver figura 2.8) externa para que todo el funcionamiento electrónico sea externo al Arduino que solo debería de funcionar como pasarela ModBus.

Figura 2.8 - Circuito típico de pulsador con resistencia *pull-up*.

Además, esta resistencia será la encargada de limitar corriente en el pin que esté conectado el pulsador. Por otro lado, valores elevados de esta resistencia reducen la fuerza de *pull-up*, haciendo con que la lectura con el pulsador abierto sea errónea.

Las resistencias internas del chip ATmega2560 están entre 20 y 50 kΩ, por lo que se ha elegido 47 kΩ que está cercano al valor máximo de las resistencias internas. Esta resistencia conlleva un consumo mientras se pulsa el pulsador de 0,1 mA.

2.3.3 Bocina

Para la bocina indicadora de verde para peatones se ha elegido la bocina CPT-2207-90T de Same Sky [13] con un consumo de 1.5 mA y frecuencia de trabajo entre 3.5 y 4.5 kHz. En la figura 2.9 se enumeran sus principales características:

SPECIFICATIONS

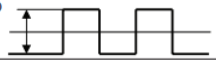
parameter	conditions/description	min	typ	max	units
rated voltage	Vp-p  0V		5		Vp-p
operating voltage				30	Vp-p
current consumption	at rated voltage			1.5	mA
rated frequency		3,500	4,000	4,500	Hz
sound pressure level	at 10 cm, rated voltage	90			dB

Figura 2.9 - Especificaciones técnicas de la bocina [13].

Para controlar la frecuencia de trabajo, se ha optado por usar un circuito integrado LM555 cuyo consumo de funcionamiento es de 6 mA (ver figura 2.10).

(T_A = 25°C, V_{CC} = 5 V to 15 V, unless otherwise specified)[1][2]

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Supply Voltage		4.5		16	V
Supply Current	V_{CC} = 5 V, R_L = ∞		3	6	mA
	V_{CC} = 15 V, R_L = ∞ (Low State) [3]		10	15	

Figura 2.10 - Especificaciones técnicas de LM555 [14].

El LM555 es un circuito temporizador que cuenta con 2 modos operativos, el primero es el modo monoestable en el cual se lanza un único pulso cuando se activa el circuito. En el segundo caso, el modo astable, el circuito oscila entre 2 señales pudiéndose controlar el tiempo en alto y en bajo con 2 resistencias y 1 condensador colocadas según la figura 2.11.

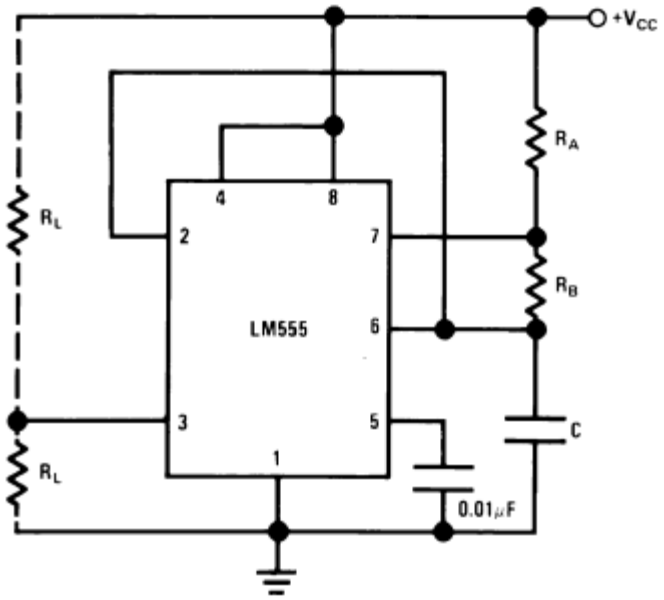


Figura 2.11 - Esquema para funcionamiento astable del dispositivo LM555 [14].

El tiempo de carga (salida alta) se calcula con la siguiente formula:

$$t_1 = 0.693\ (R_A + R_B)\ C \tag{2}$$

El tiempo de descarga (salida baja) con:

$$t_2 = 0.693\ R_A\ C \tag{3}$$

Por lo que el periodo total es:

$$T = t_1 + t_2 = 0.693\ (R_A + 2R_B)\ C \tag{4}$$

Y la frecuencia de oscilación:

$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)\ C} \tag{5}$$

Quedando el ciclo de trabajo finalmente:

$$D = \frac{R_B}{R_A + 2R_B} \tag{6}$$

En un principio se había diseñado el circuito con R_A = 0 Ω para así obtener una onda cuadrada con el mismo tiempo en alto como en bajo. Pero tras comprobar el circuito, este no funcionaba, así que finalmente se colocó una resistencia.

Para el cálculo inicial se usó 1 nF para el condensador y 4 kHz para la frecuencia en la ecuación (5) obteniendo así el valor de R_B= 180 kΩ. No obstante, una vez se ha colocado la R_A con valor 1.8 kΩ la frecuencia final es de 3.9801 kHz.

Para terminar, por las resistencias R_A y R_B también hay corriente que se puede calcular según la ley de ohm y cuyo valor máximo es de 0.028 mA.

2.3.4 Resumen

Se ha descrito los componentes necesarios y se ha analizado el consumo eléctrico de cada uno. En total se tendrán 4 semáforos, cada uno con 2 LEDs rojos, 2 LEDs ámbar, 2 LEDs verde, 1 pulsador con resistencia *pull-up* y 1 bocina con un temporizador LM555.

Referencia	Cantidad	Consumo Individual	Consumo Total
HLMP-4700	8	1.833 mA	14.667 mA
HLMP-4719	8	1.778 mA	14.222 mA
HLMP-4740	8	1.722 mA	13.778 mA
Pulsador	4	0.106 mA	0.426 mA
Bocina	4	1.5 mA	6 mA
LM555	4	6.028 mA	24,112 mA
		Total	**73.204 mA**

Tabla 2.4 - Resumen de corriente consumida por elementos externos al Arduino.

Como se puede ver en la tabla 2.4, el consumo de componentes externos máximo es de 73.2 mA, además, todos se alimentarán desde los pines del chip ATmega2560 que aun contaba con 186 mA hasta llegar al máximo permitido, y cuya limitación en algún

conjunto de pines más baja era de 100 mA, por lo que se ha cumplido con los requisitos de potencia para controlar los 4 semáforos desde 1 solo Arduino Mega 2560.

Finalmente, se diseña el esquema completo de cada semáforo, indicando los pines necesarios para controlarlo (ver figura 2.12).

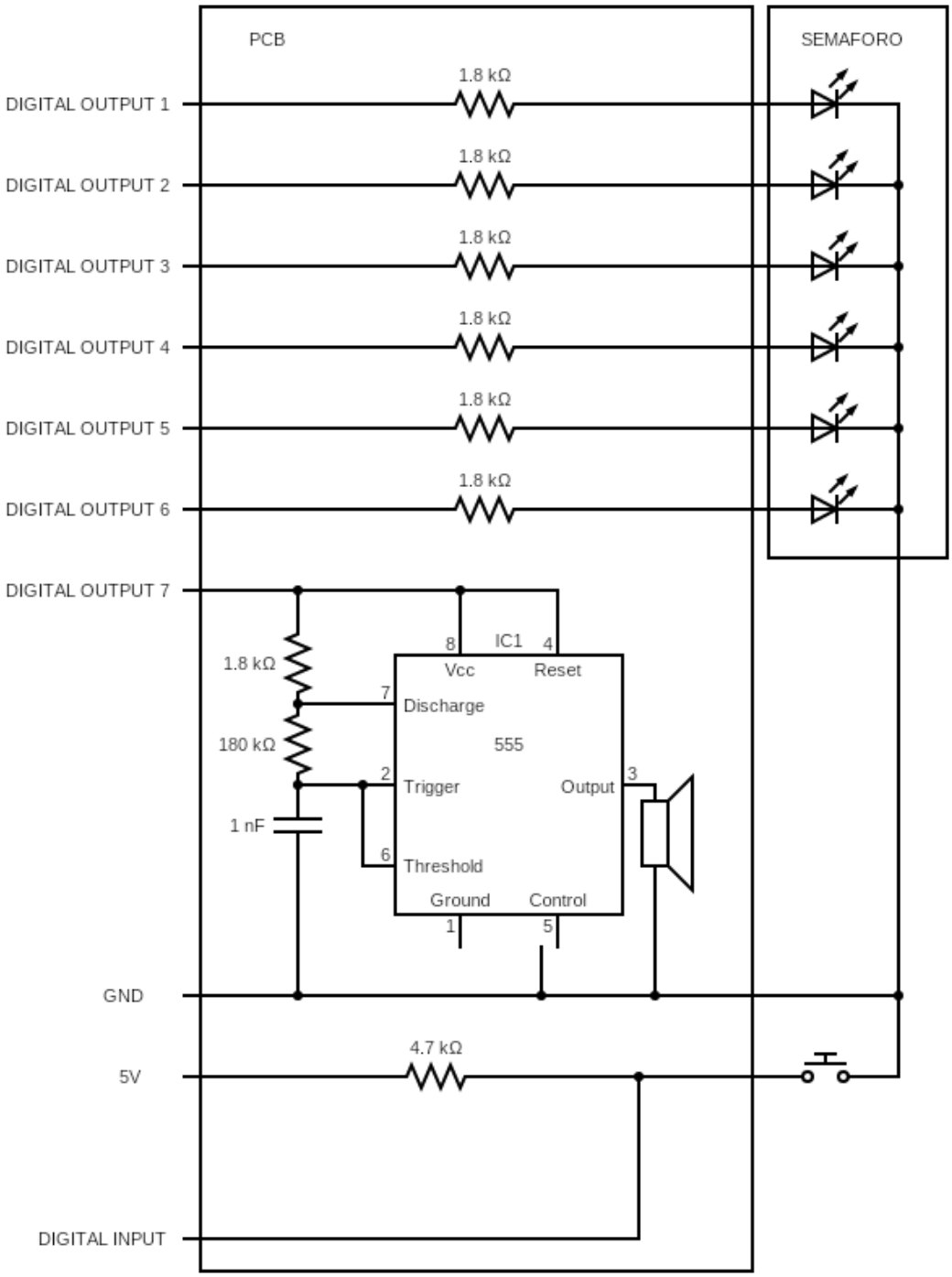


Figura 2.12 - Esquema eléctrico de un semáforo.

3

Desarrollo de prototipos

3.1 Introducción

En este capítulo se detalla el proceso llevado a cabo para el desarrollo del prototipo. Para facilitar la explicación se comentarán brevemente las distintas iteraciones llevadas previas a la versión final.

3.2 Primer prototipo

El primer prototipo se montó sobre una placa de desarrollo *breadboard* (ver figura 3.1) que permite la conexión de componentes sin necesidad de soldaduras reduciendo así el tiempo entre iteraciones.

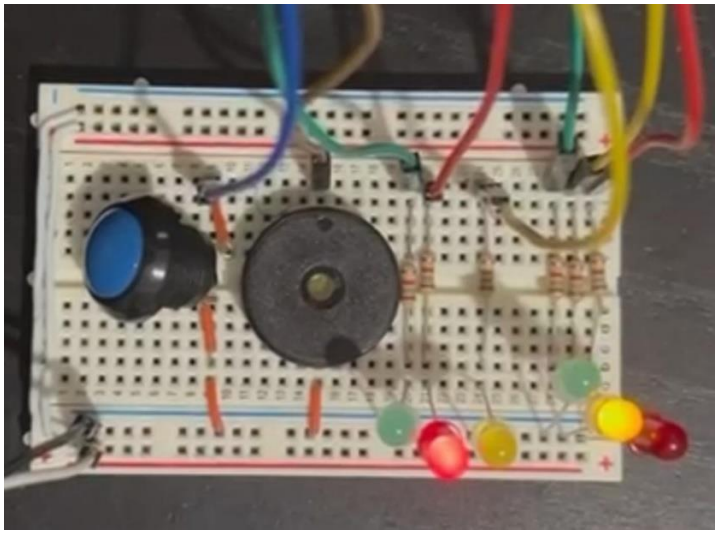


Figura 3.1 - Fotografía del primer prototipo.

Durante las primeras fases de este prototipo se comprobó el correcto funcionamiento de los LEDs, asegurando que son visibles en un entorno con amplia claridad, el correcto funcionamiento de la bocina, controlándola desde el Arduino usando el comando Tone() que permite generar una señal PWM simétrica en la frecuencia deseada entre 31 y 65535 Hz, y el correcto funcionamiento del pulsador, haciendo uso de las resistencias *pull-up* internas del microcontrolador.

```
91  void Run_State4() {
92    if (millis() - stateTimer > CAR_RED_TIME) {
93      stateVar = 5;
94      stateTimer = millis();
95    } else {
96      digitalWrite(PEDESTRIAN_RED, LOW);
97      digitalWrite(PEDESTRIAN_GREEN, HIGH);
98      if ((millis() - stateTimer) % 400 < 200) {
99        digitalWrite(CAR_RIGHT, HIGH);
100       tone(PEDESTRIAN_BEEP,4000);
101     } else {
102       digitalWrite(CAR_RIGHT, LOW);
103       noTone(PEDESTRIAN_BEEP);
104     }
105   }
106 }
```

Figura 3.2 - Control de bocina.[1]

En la figura 3.2 se puede observar un fragmento de código en la IDE de Arduino en el cual se coloca la luz de peatones en verde y se tiene el pitido para usuarios invidentes y la señal luminosa que permite a los coches girar a la derecha con la intermitencia cada 200 ms. En este código de prueba se decidió usar el mismo periodo de intermitencia para ambos.

```
29  pinMode(PEDESTRIAN_BUTTON, INPUT_PULLUP);
30  attachInterrupt(digitalPinToInterrupt(PEDESTRIAN_BUTTON), Run_ButtonPressed, FALLING);
```

Figura 3.3 - Declaración de una interrupción con el pulsador.

En el mismo prototipo se comprobó el correcto funcionamiento del pulsador (ver figura 3.3) haciendo con que el programa pasara del estado 1 (verde para los vehículos y rojo para peatones) al estado 2 (transición hacia rojo para vehículos y verde para peatones) (ver figura 3.4).

```
120 void Run_ButtonPressed() {
121   if (stateVar == 1) {
122     stateVar = 2;
123     stateTimer = millis();
124   }
125 }
```

Figura 3.4 - Subfunción con la lógica para cambiar de estado al presionar el pulsador.

[1] El código fuente completo de este prototipo se encuentra en el apartado 7.1.

3.3 Segundo prototipo

Una vez terminado el primer prototipo y con el fin de validar el funcionamiento electrónico antes de diseñar los modelos físicos de los semáforos se hizo una prueba con CoDeSys, prueba que no se superó ya que la bocina no funcionaba. Tras investigar el problema y determinar que era por la baja velocidad de comunicación entre el PLC virtual y el Arduino que está limitada a 100 ms, y necesitando que fuera de 125 ns para codificar una señal de 4 kHz se decidió incluir el dispositivo temporizador LM555 descrito en la sección 2.3.3.

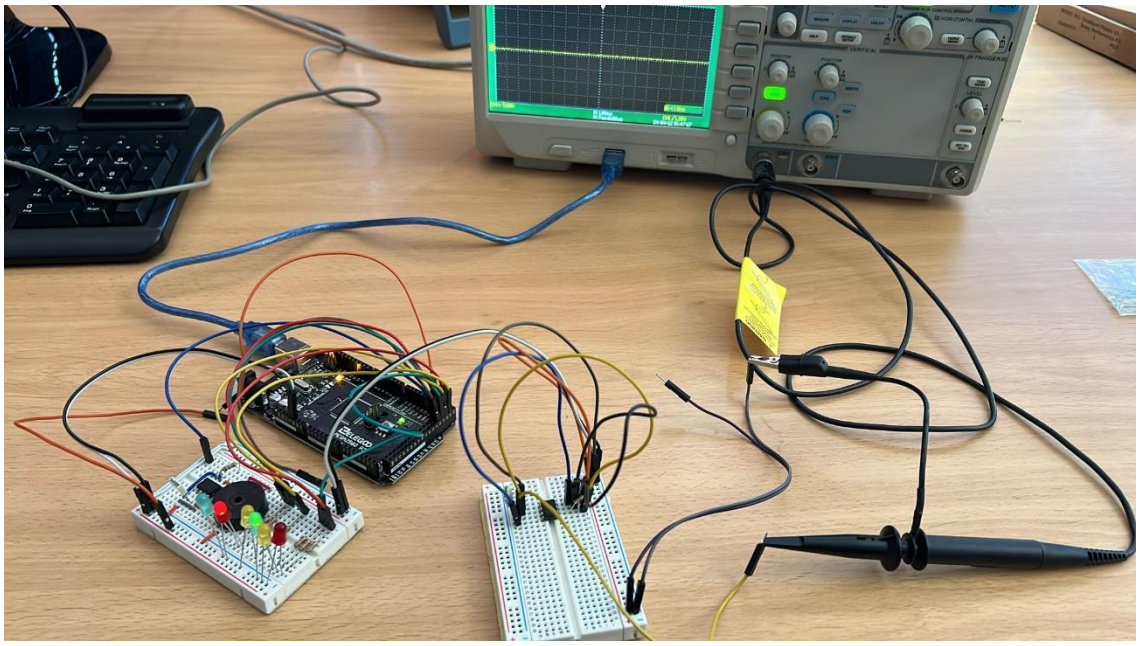


Figura 3.5 - Depuración de error con circuito LM555.

Como se comentó en el capítulo anterior, en las primeras pruebas realizadas con el LM555 se pudo comprobar que la bocina seguía sin funcionar. En la figura 3.5 se muestran algunas de las pruebas realizadas con la segunda variante del sistema electrónico.

Finalmente, después de numerosas pruebas. se determinó que habría que colocar una resistencia cuyo valor mínimo no viene especificado en el *datasheet* del circuito, pero que es necesaria para el correcto funcionamiento del circuito.

Una vez colocada esa segunda resistencia la bocina volvía a funcionar, pudiendo así continuar el proceso de desarrollo. En la figura 3.6 se incluye una fotografía del segundo prototipo.

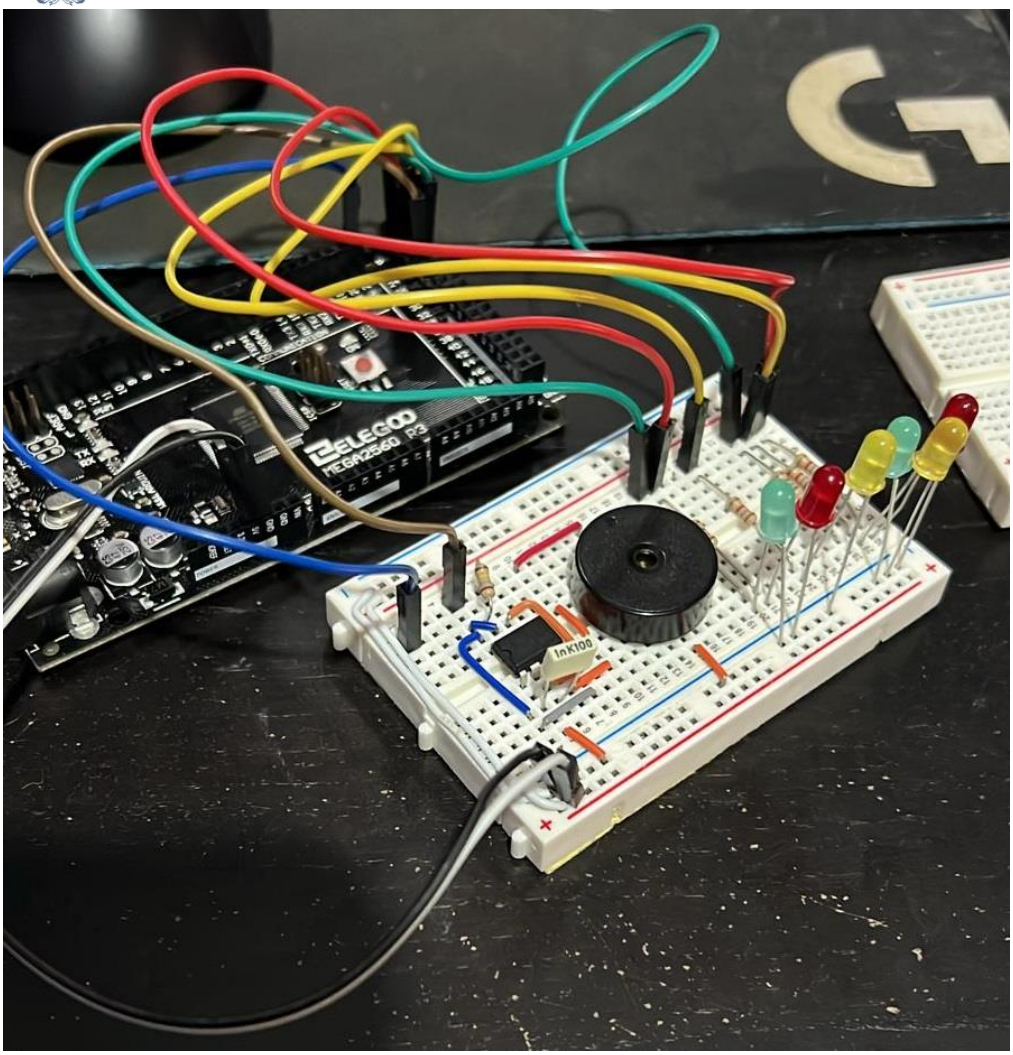


Figura 3.6 - Fotografía del segundo prototipo.

3.4 Prototipo final

Una vez validado el funcionamiento del subsistema electrónico se procedió a montarlo en una *perfboard* para evitar malos contactos cuando se ensamble en el chasis y/o se manipule durante las sesiones de docencia prácticas. El desarrollo del prototipo final se divide en 3 partes:

- Colocación de los componentes en una placa *perfboard* de 60x40 mm.
- Diseño 3D del semáforo con software específico.
- Fabricación de base de semáforo para acomodar la placa *perfboard.*

3.4.1 Placa *perfboard*

Las placas *perfboard* son placas PCBs específicas de prototipado que vienen con una matriz de agujeros con la separación estándar de componentes electrónicos de 2.54 mm (1/10 inch). En este caso se compró una placa que también tiene agujeros en las cuatro esquinas para su montaje fijo con tornillos.

A la hora de montar la *perfboard* se ha tenido que conectar los distintos pines usando cables tanto aislados cuando era necesario como sin aislar en otros casos más sencillos.

Además de esto, se añadió una clavija de 12 pines con conectores hembra DuPont que facilitarán la conexión de la placa con el Arduino. En este punto se midieron los agujeros y la posición del conector y de la bocina para poder acomodar la base del semáforo a los mismos (ver figuras 3.7 y 3.8).

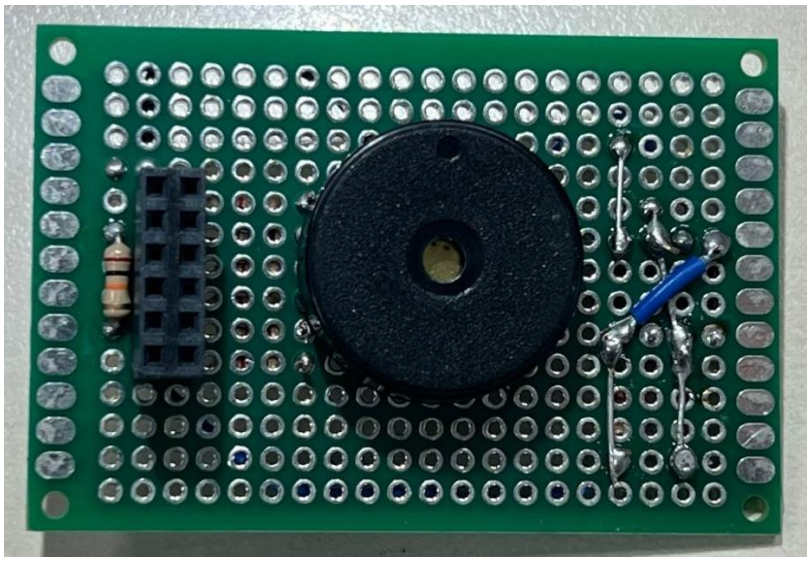


Imagen 3.7 - *Perfboard* (Lado A).

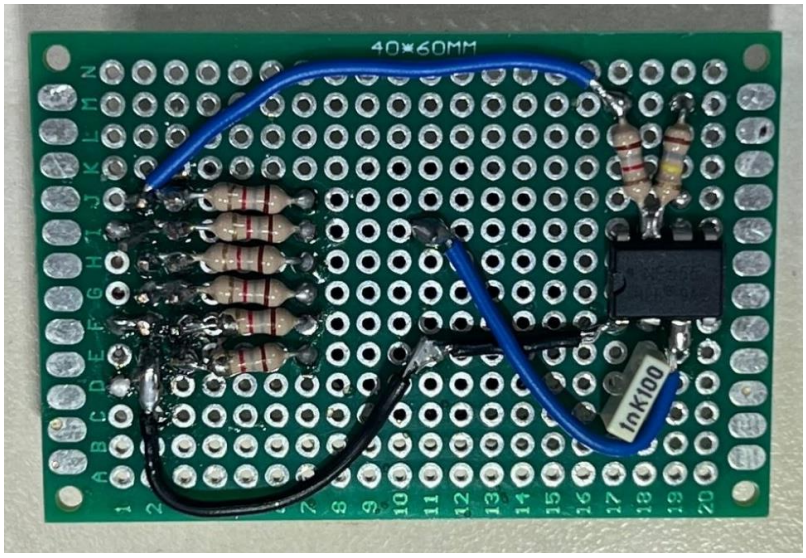


Imagen 3.8 - *Perfboard* (Lado B).

3.4.2 Semáforo

3.4.2.1 Introducción

En este apartado se empezará por detallar las iteraciones llevadas en el diseño del semáforo y finalmente se detallarán los pasos seguidos en el montaje de este.

3.4.2.2 Primer diseño

En el desarrollo del proyecto se han presentado varios diseños, empezando por este que solo contaba con las señales luminosas para los coches y cuyo montaje estaba completamente diseñado para ejecutarse con tornillería (ver figura 3.9).

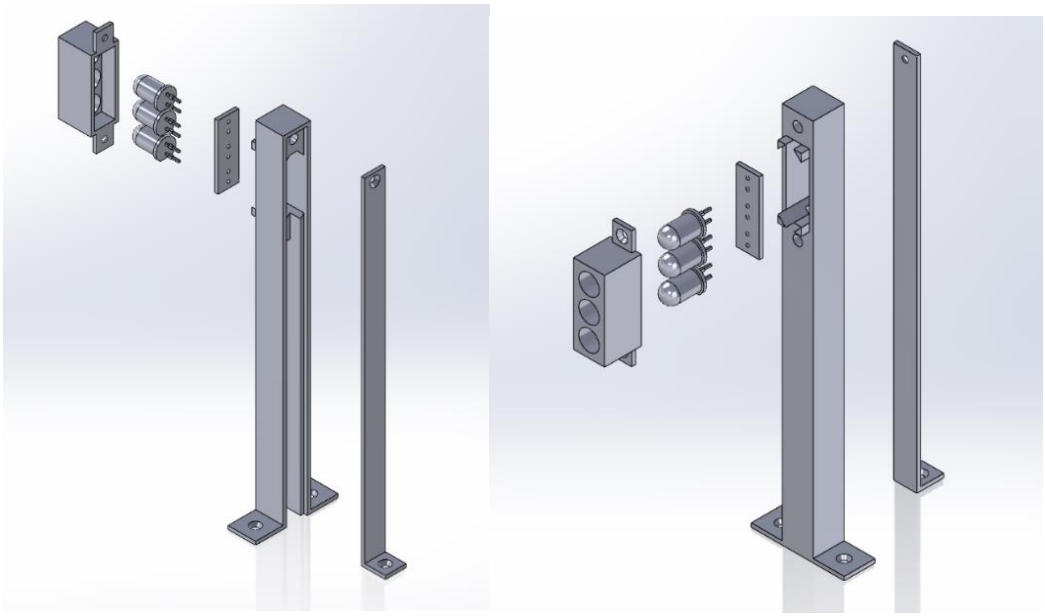


Figura 3.9 - Primer diseño del semáforo.

Algunas ventajas por destacar de este diseño seria la posibilidad de usar la misma base para configuraciones semafóricas distintas, la facilidad de mantenimiento de los LEDs y la fijación de los LEDs en su sitio mediante una placa perforada.

4.4.2.3 Segundo diseño

En el segundo diseño se ha optado por abandonar el uso masivo de tornillería optando por un diseño monolítico en casi su totalidad dejando solo la pletina trasera con montaje por tornillería para tener acceso al cableado del semáforo. Además de esto, se añadieron las señales luminosas de giro a derecha para los vehículos y de peatones (ver figura 3.10).

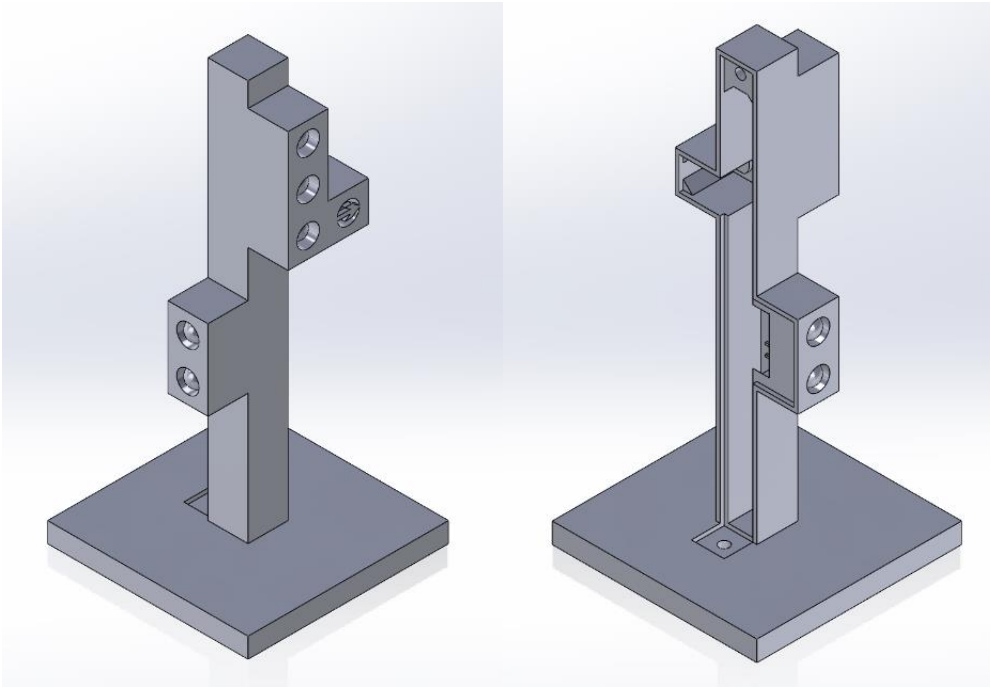


Figura 3.10 - Segundo diseño del semáforo.

Sin embargo, este diseño suponía un gran reto tanto a la hora de imprimirse en 3D como a la hora de soldar los cables a los LEDs.

4.4.2.4 Diseño final

Para el diseño final se han tomado las medidas necesarias para facilitar tanto la impresión como el trabajo de soldadura, además de buscar una estética más realista.

Con el fin de facilitar la impresión se ha tratado de eliminar todos los puntos que necesitarían soporte adicional a la hora de impresión. Pudiendo hacerlo para todos los puntos menos para un agujero en la pletina trasera (ver figura 4.11).

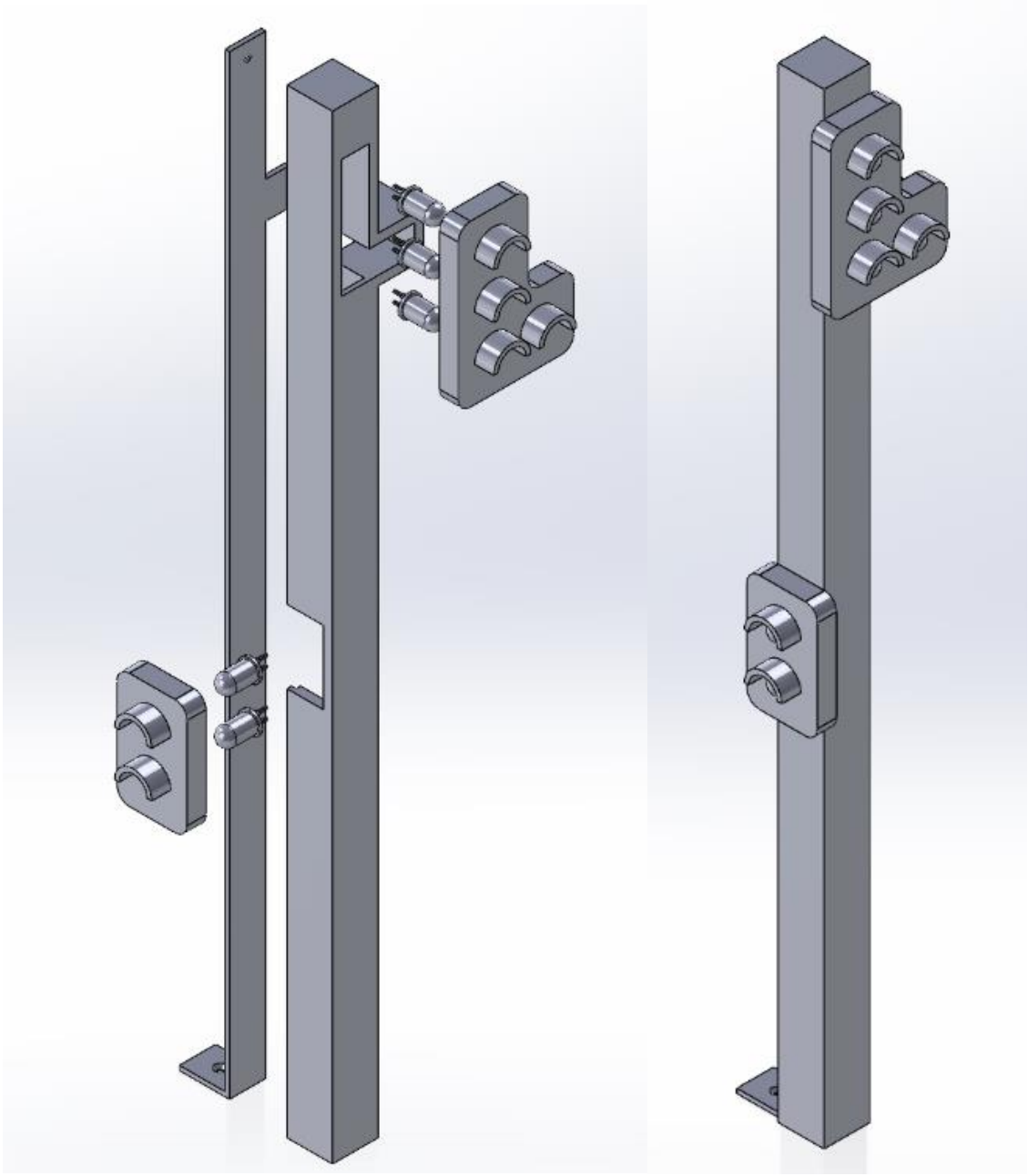


Figura 4.11 - Diseño final del semáforo.

3.4.3 Base de semáforo

La base del semáforo finalmente se diseñó con 68x68mm de base teniendo una tapa de 3.2mm de espesor y estando cerrada en todo el perímetro por una pared de 6 mm de espesor y 24 mm de alto.

Además, se dejaron los huecos correspondientes a la *perfboard,* los agujeros para la tornillería, clavija y bocina, el hueco para el semáforo, con una hendidura para que sea fácilmente colocado en su posición correcta y con un agujero para la colocación del pulsador (ver figura 3.12).

Figura 3.12 - Base del semáforo.

3.4.4 Montaje

3.4.4.1 Introducción

En este apartado se describirá el proceso de montaje de un semáforo. El montaje empieza con la impresión 3D, seguido de la unión de partes, el cableado y el ensamblaje final.

3.4.4.2 Impresión 3D

En esta fase se imprimieron las pletinas de las señales luminosas y la estructura principal usando capas de 0,3 mm y 20% de relleno. La pletina trasera requirió un 100% de relleno, ya que al ser una pieza alargada y fina necesita de mayor integridad física. En la figura 3.13 se muestra el final de la impresión del cuerpo principal y de las pletinas de las señales luminosas para los vehículos y para los peatones de 3 semáforos.

Imagen 3.13 - Foto al terminar una impresión 3D.

3.4.4.3 Montaje de componentes electrónicos y cableado

El montaje empieza por la colocación de los LEDs en las pletinas y la interconexión de las tierras como se puede ver en la figura 3.14.

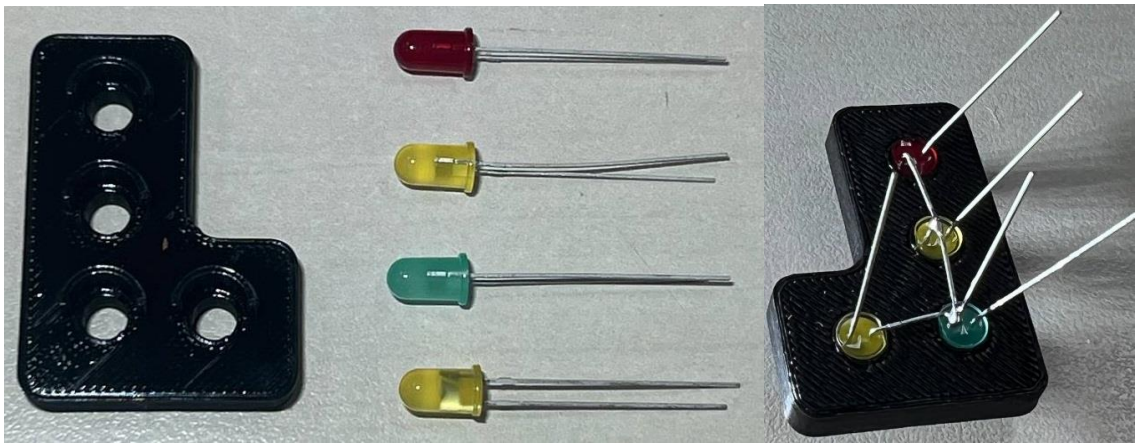


Figura 3.14 - Montaje de LEDs con tierra unificada.

A continuación, se soldaron cables con colores identificativos (ver figura 3.15) a cada punto y se pegó la pletina de los LEDs para vehículos al cuerpo principal del semáforo usando *superglue*.

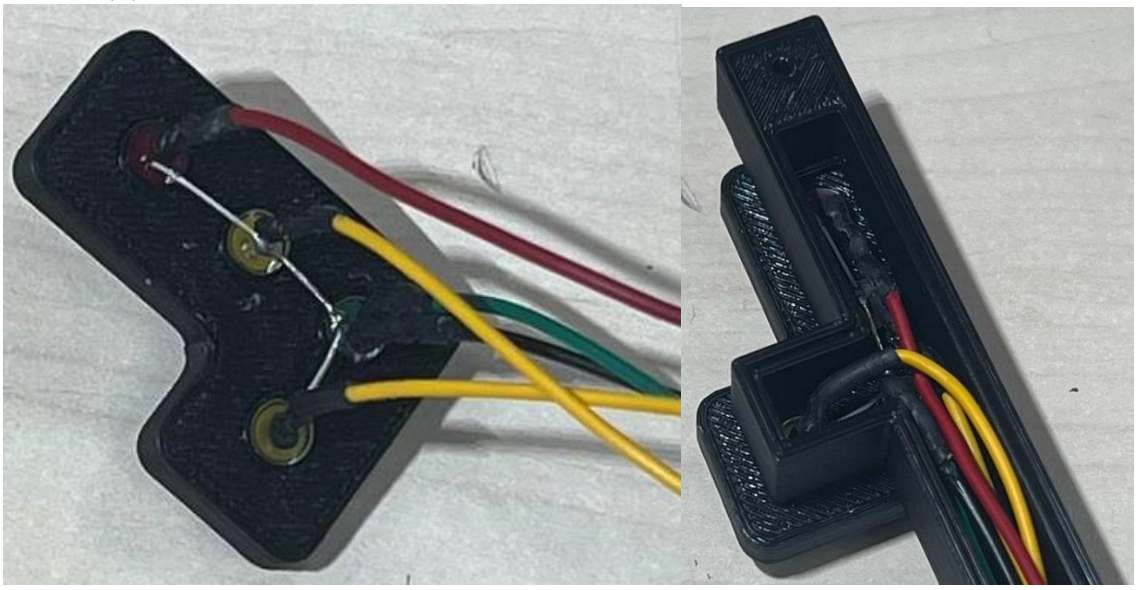


Figura 3.15 - LEDs cableados y pletina pegada en cuerpo principal.

El proceso se repite con los LEDs para los peatones, uniendo sus tierras con los LEDs de los vehículos también para reducir los cables que bajan a la placa. En la figura 3.16 se puede observar como el cable de tierra de los LEDs de los vehículos llega por la parte superior, se une con la tierra del led rojo y baja hacia la placa desde la tierra del LED verde.

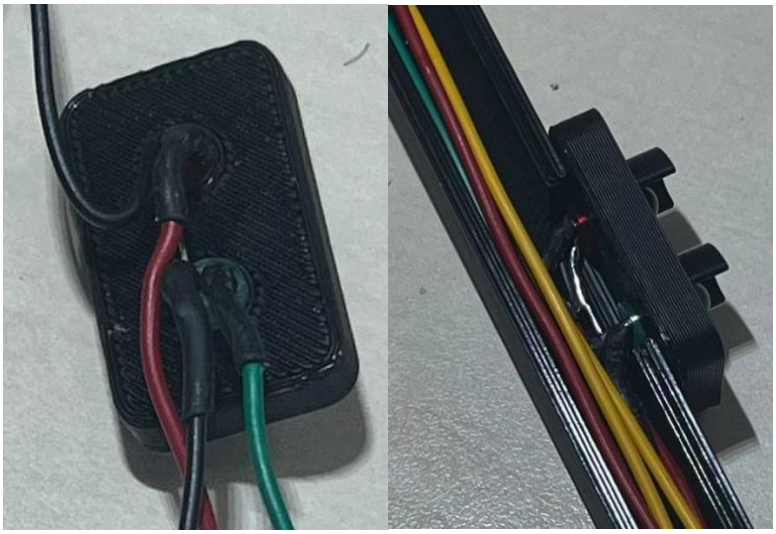


Figura 3.16 - LEDs para peatones colocados en su sitio.

3.4.4.4 Ensamblaje final

Se empezó por soldar insertos de rosca en los agujeros correspondiente tanto en la base del semáforo como en el semáforo en sí.

Una vez fijados estos insertos se procedió a fijar la *perfboard* en su posición final usando separadores de 6 mm. En la figura 4.17 se ve como los componentes en la *perfboard* encajan en los huecos de la base del semáforo y como la *perfboard* ya tenía puntas de cables soldadas cuando fue colocada para facilitar su unión con los cables del semáforo.

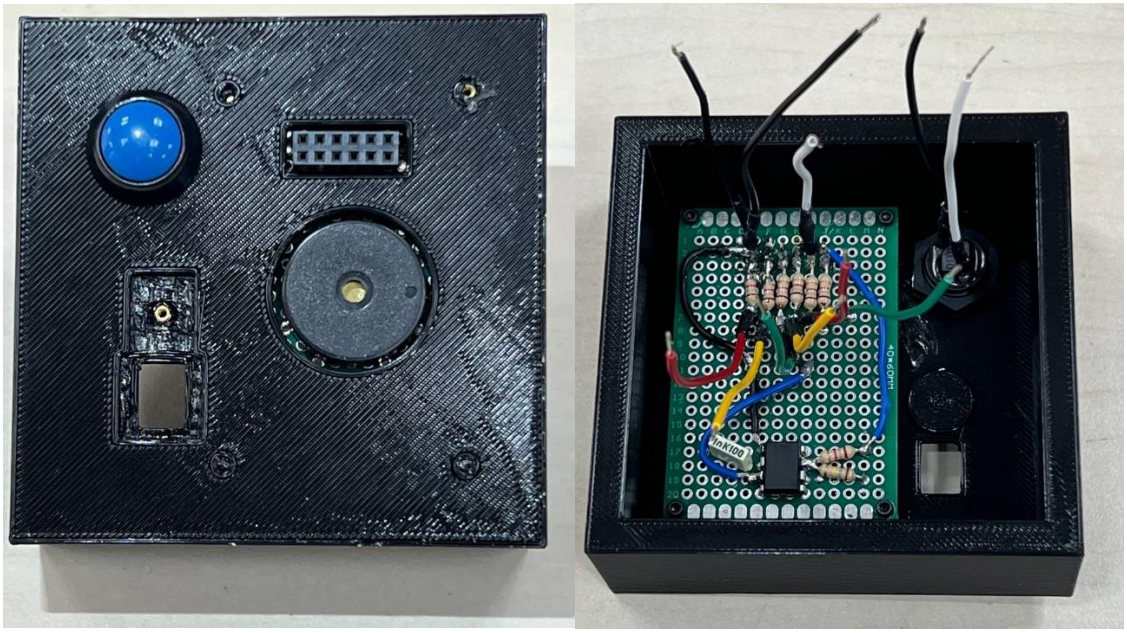


Figura 3.17 - Base de semáforo con *perfboard* montada.

Finalmente se fijó el semáforo en su sitio (ver figura 3.19) y se conectaron los cables correspondientes, soldándolos con estaño y aislando cada soldadura con aislamiento termo retráctil como se puede ver en la figura 3.18.

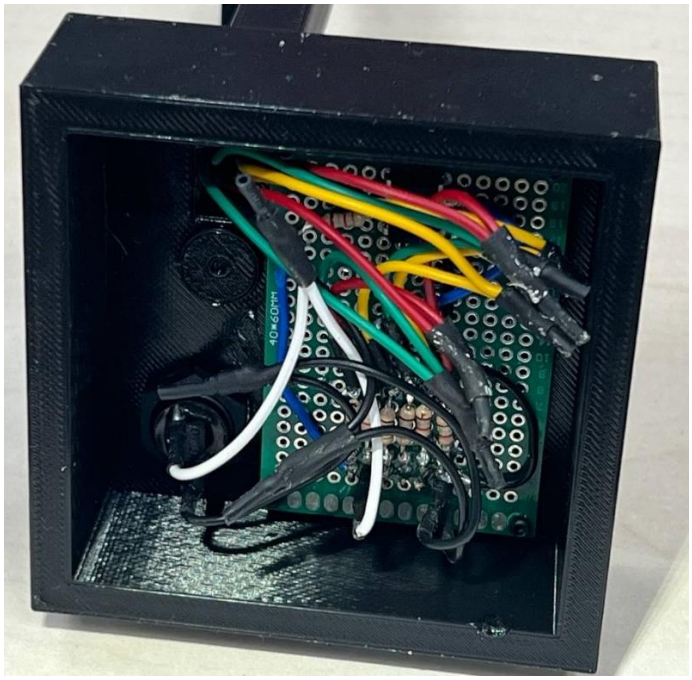


Figura 3.18 - Fotografía del cableado del semáforo terminado.

Figura 3.19 - Fotografía del semáforo terminado.

Para terminar, se hicieron pruebas unitarias comprobando que todos los LEDs y la bocina funcionan y dando por finalizado el montaje del prototipo.

4.5 Presupuesto y tiempos de montaje

En este apartado se indica el tiempo de montaje de cada semáforo y el coste de los materiales empleados en la fabricación de los 4 semáforos. Finalmente se deja una estimativa del coste total para 15 cruces semafóricos pudiendo así tener 1 cruce semafórico por cada pareja en las prácticas.

Descripción	Tiempo
Montaje de placa *perfboard*	2 horas
Impresión de semáforo	4.5 horas
Montaje de semáforo	2 horas
Colocación de semáforo y *perfboard* en la base	3 horas
Total	**11.5 horas**

Tabla 3.1 - Tiempo de montaje de cada semáforo.

Referencia	Descripción	Cantidad	Coste/Ud.	Coste Total
Arduino MEGA 2560 R3	Marca ELEGOO	1	20,79 €	20,79 €
HLMP-4700	LED rojo	8	0,356 €	2,85 €
HLMP-4719	LED ámbar	8	0,321 €	2,57 €
HLMP-4740	LED verde	8	0,320 €	2,56 €
999-P9-111126	Pulsador	4	13,63 €	54,52 €
179-CPT-2207-90T	Avisador acústico	4	1,03 €	4,12 €
LM555CN/NOPB	Temporizador	4	1,07 €	4,28 €
Perfboard	60x40 mm	4	1,045 €	4,18 €
Female 2x6 PIN PCB	Clavija hembra	4	0,354 €	1,42 €
1K8 5% 1/4W	Resistencia 1.8 kΩ	28	0,05 €	1,40 €
4K7 5% 1/4W	Resistencia 4.7 kΩ	4	0,05 €	0,20 €
180K 5% 1/4W	Resistencia 180 kΩ	4	0,05 €	0,20 €
Estaño	Alambre para soldadura	1	5,00 €	5,00 €
AWG28	Cable max 230mA	20	0,50 €	10,00 €
PLA 1.75 1 Kg	Filamento para 3D	1	15,99 €	15,99 €
DIN965 M1.6 3mm	Tornillo cabeza avellanada	8	0,034 €	0,27 €
DIN912 M1.6 8mm	Tornillo *hex socket*	16	0,041 €	0,66 €
IR M1.6 (OD2.5mm) 3mm	Inserto roscado	24	0,037 €	0,89 €
SP OD7 ID3 6mm	Espaciador de nailon	16	0,023 €	0,37 €
			Total	**132,26 €**

Tabla 3.2 - Presupuesto de materiales para cruce semafórico completo.

Descripción	Cantidad	Coste por ud.	Coste total
Cruce semafórico	15 uds.	132,26 €	1 983,90 €
Mano de obra	690 horas	21 €	14 490,00 €
		Total	**16 473,90 €**

Tabla 3.3 - Presupuesto total para 15 cruces semafóricos.

4

Prácticas

4.1 Introducción

En este capítulo se expondrán una serie de propuestas de prácticas a realizar por los alumnos utilizando los semáforos desarrollados en este proyecto.

Habrá una serie de ejercicios guiados en los que se impartirá a los alumnos los conocimientos necesarios para usar CoDeSys y el cruce semafórico, teniendo 2 ejercicios que tendrán que superar por su cuenta. Uno al final de la segunda práctica y otro en la última práctica.

4.2 Practica 1 - Introducción a CoDeSys

4.2.1 Introducción

En esta práctica se introducirá a los alumnos al entorno de programación de PLCs CoDeSys enseñándoles a crear un programa básico con pantalla de visualización virtual.

4.2.2 Pasos previos en CoDeSys

Empezamos por abrir el programa CoDeSys desde el ejecutable "*CoDeSys Without Profile*" y eligiendo la versión instalada. Una vez dentro empezaremos un nuevo proyecto estándar.

Al empezar el proyecto se tendrá que configurar el dispositivo como CoDeSys Control Win V3 y el PLC_PRG en *Sequential Function Chart* (SFC).

En este punto hay que asegurarse de que el *Gateway* y el *Control Win* se están ejecutando, pudiéndolos encontrar en la bandeja del sistema (ver figura 4.1).

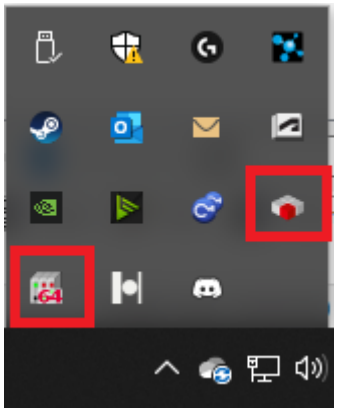


Figura 4.1 - Bandeja del sistema con Gateway y Control Win ejecutándose.

En caso de que no estén ambos iconos hay que abrir el programa correspondiente, usando el buscador de Windows y escribiendo "codesys" se puede encontrar fácilmente tanto el *CoDeSys Gateway V3* como el *CoDeSys Control Win V3*.

En caso de que los iconos estén, pero en tonos grises, hay que ponerlos en marcha pulsando el botón derecho del ratón sobre el icono que se encuentre en gris y pulsando "*Start Gateway*" en el caso del *Gateway* o "*Start PLC*" en el caso del *Control Win*.

Si es la primera vez que se está ejecutando *Control Win*, solicitará que se configure las credenciales de acceso al PLC virtual.

Finalmente tenemos que configurar el PLC virtual en el CoDeSys pulsando dos veces sobre *Device*, escaneando la red y eligiendo el PLC que vamos a usar. En nuestro caso, el propio PC que estamos usando. En la figura 4.2 se puede ver donde se encuentra cada uno de los botones a pulsar.

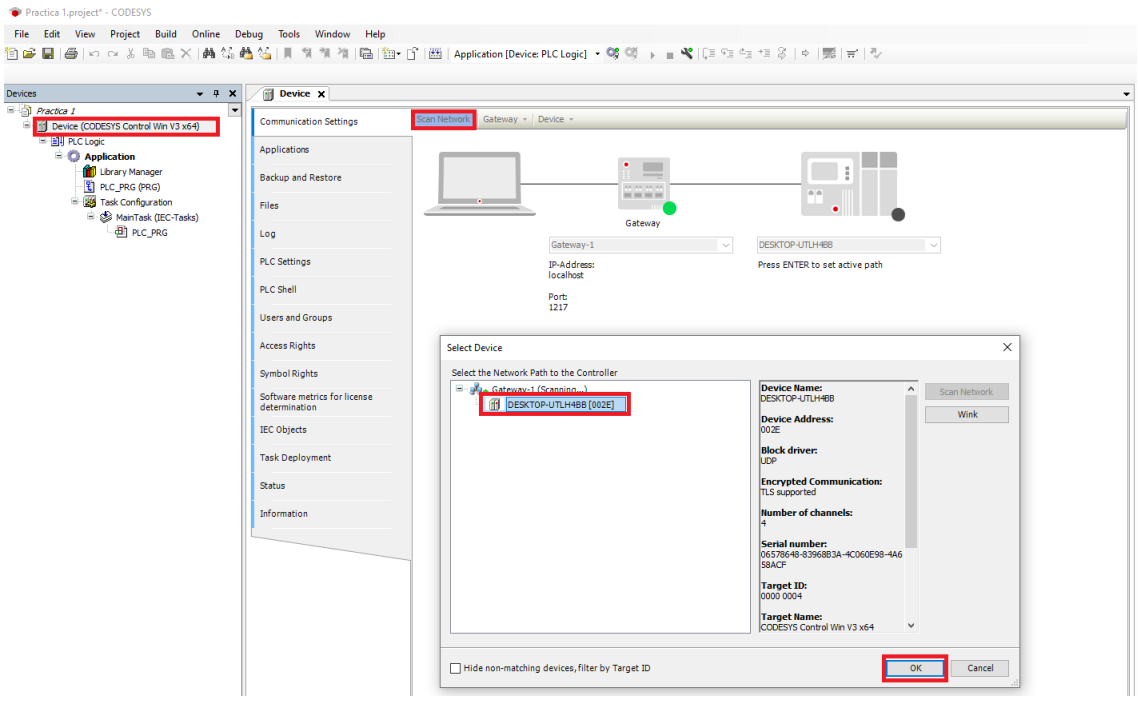


Figura 4.2 - Configurando PLC Virtual desde CoDeSys.

Al elegir el ordenador, el programa solicitará las credenciales del PLC. Al conectarse con la PLC virtual tanto el *Gateway* como el ordenador aparecerán con el circulo en verde.

Una vez estamos conectados con el PLC Virtual procedemos a escribir nuestro primer programa.

4.2.3 "Hola Mundo" en CoDeSys

Al cargar un proyecto de CoDeSys en un PLC, se ejecuta el programa PLC_PRG. Procedemos abrirlo y nos encontramos con tres zonas separadas (ver figura 4.3):

- La definición de variables.
- El programa SFC.
- Las herramientas que nos permiten modificar el programa SFC.

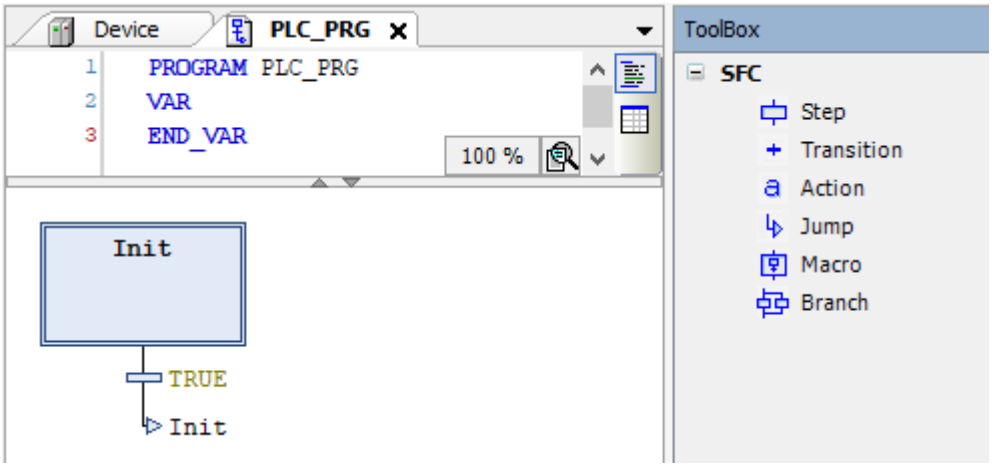


Figura 4.3 - Pantalla inicial de programación.

Ahora añadiremos una etapa más con una transición y añadimos una acción en una de ellas, arrastrando la herramienta correspondiente a donde queremos añadirla al programa. En la figura 4.4 se puede observar cómo quedaría el programa al añadir los elementos mencionados.

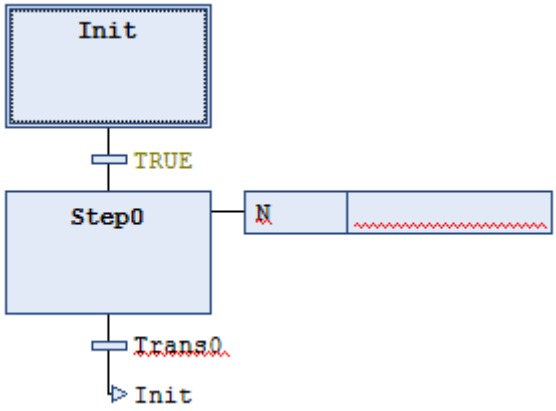


Figura 4.4 - Programa con 2 etapas y 2 transiciones.

Continuamos escribiendo las condiciones de transición entre las dos etapas, para ello recurriremos al tiempo interno del paso y pasaremos al siguiente paso cuando haya

pasado 1 segundo, para ello en la transición hay que escribir "*Init.t > t#1s*" en el caso de la primera y sustituyendo *Init* por la etapa correspondiente en la segunda. Recuerden, siempre debe que haber 1 transición entre 2 pasos en SFC.

En caso de querer usar otras cantidades de tiempo se puede hacer según las unidades que vienen en la figura 4.5, respetando siempre el orden de mayor a menor.

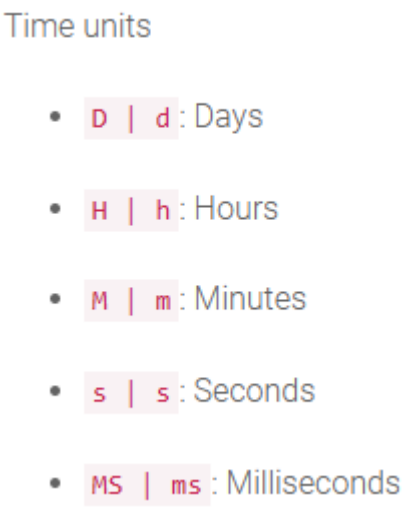


Figura 4.5 - Unidades de tiempo en CoDeSys [15].

Por lo que si queremos un retraso de minuto y medio habría que escribir t#1m30s.

A continuación, configuramos la acción escribiendo el nombre de una variable en el cuadro derecho de la misma y declarándola como BOOL (ver figura 4.6).

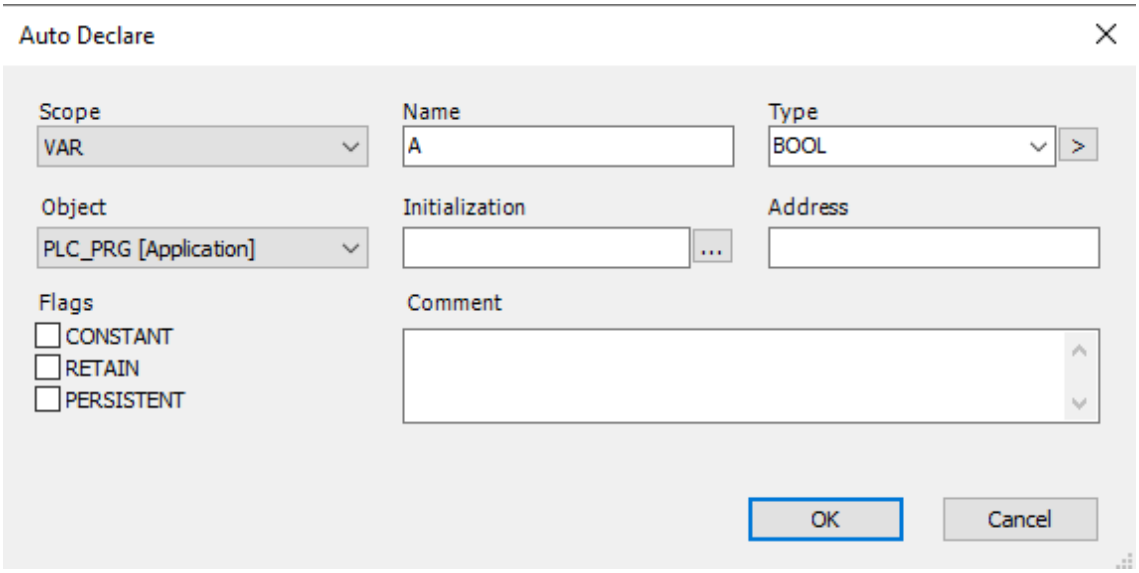


Figura 4.6 - Declaración de una variable en CoDeSys.

En este momento el programa estaría listo para ser ejecutado debiendo estar como se puede observar en la figura 4.7.

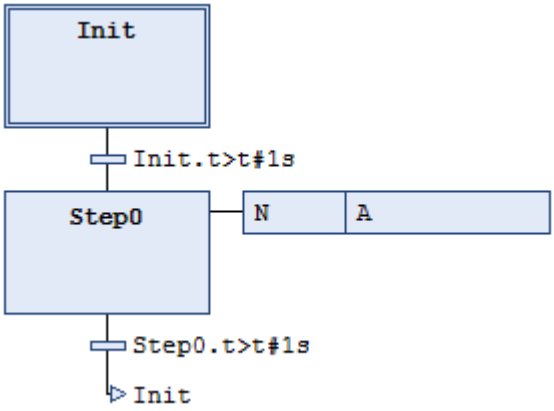


Figura 4.7 - Programa básico de CoDeSys.

Siendo la N en la acción un indicador de que tiene que hacer el programa sobre la variable A y pudiendo modificarse según la tabla que se encuentra en la figura 4.8.

N	Non-stored	The action is active as long as the step.
R	overriding Reset	The action is deactivated.
S	Set (Stored)	CODESYS executes this action as soon as the step is active. The action execution is continued even when the step has been deactivated until it gets a reset.
L	time Limited	CODESYS executes this action as soon as the step is active. The action is executed until the step is deactivated or the given time span has elapsed.
D	time Delayed	CODESYS starts executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until the step is deactivated.
P	Pulse	CODESYS executes the action exactly two times: one time when the step is activated and one time when the step is deactivated.
SD	Stored and time Delayed	CODESYS starts executing the action only after the given delay time has elapsed following step activation. The action is executed until it gets a reset.
DS	Delayed and Stored	CODESYS starts executing the action only after the given delay time has elapsed following step activation and the step is still active. The action is executed until it gets a reset.
SL	Stored and time limited	CODESYS executes this action as soon as the step is activated. It is executed until the specified time has elapsed or it gets a reset.

Figura 4.8 - Posibles *qualifiers* en una acción de SFC [16].

Para la ejecución del programa hay cuatro iconos en la barra de herramientas superior que sirven para conectarse y desconectarse del PLC y para empezar o parar el programa.

Figura 4.9 - Controles de PLC desde CoDeSys.

Empezamos por conectarnos a la PLC pulsando el primer botón (ver figura 4.9), tras lo cual CoDeSys nos avisará que el programa no se encuentra en la PLC y nos pregunta si queremos descargar el programa en la misma, a lo que tenemos que decirle que sí.

Al cargar el programa en la PLC la pantalla de programación se cambiará a una pantalla de depuración donde podemos ver qué dispositivos están conectados, el estado de las variables y en que etapa del programa nos encontramos (ver figura 4.10).

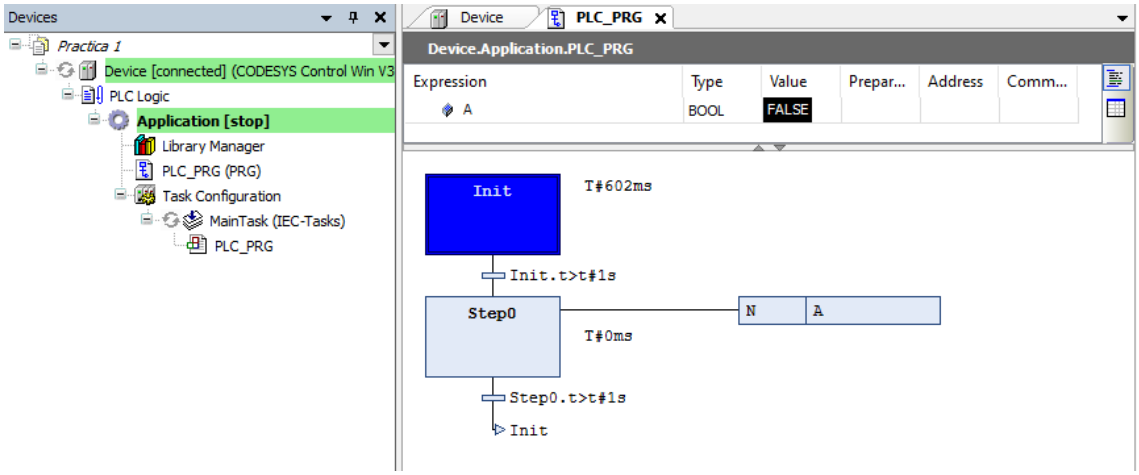


Figura 4.10 - PLC en ejecución visualizado desde CoDeSys.

Una vez en esta pantalla pulsamos al botón de "*play*", el tercer botón de los controles de PLC (ver figura 5.9), empezando así la ejecución del programa en el PLC.

Si esta todo correcto, se observa que la etapa señalizada cambia cada segundo y que la variable, en este caso "A", cambia de *FALSE* a *TRUE* y viceversa cada segundo también.

4.2.4 Ventana de visualización virtual

CoDeSys cuenta también con un entorno de visualización donde se pueden simular botoneras, paneles de control o incluso visualizar procesos más avanzados.

Para acceder a ello primero hay que crear una hoja de visualización, pulsando con el botón derecho sobre *Application* y eligiendo *Visualization* en el submenú de *Add Object* (ver figura 4.11).

En este punto aparece una ventana de elección de librerías en el que le podemos dar un nombre a la página de visualización. En este caso no elegiremos ninguna librería ya que la librería básica de CoDeSys contiene los elementos necesarios para nuestro caso.

Al igual que en la pantalla de programación, en la pantalla del entorno visual contiene las herramientas disponibles del lado derecho mientras que del lado izquierdo encontramos la pantalla sobre la cual colocaremos nuestros elementos.

Para visualizar la variable A crearemos una caja con una señal lumínica verde. Para ello, empezaremos por elegir el menú *Basic* en las herramientas de visualización, y elegiremos la herramienta de rectángulo redondeado.

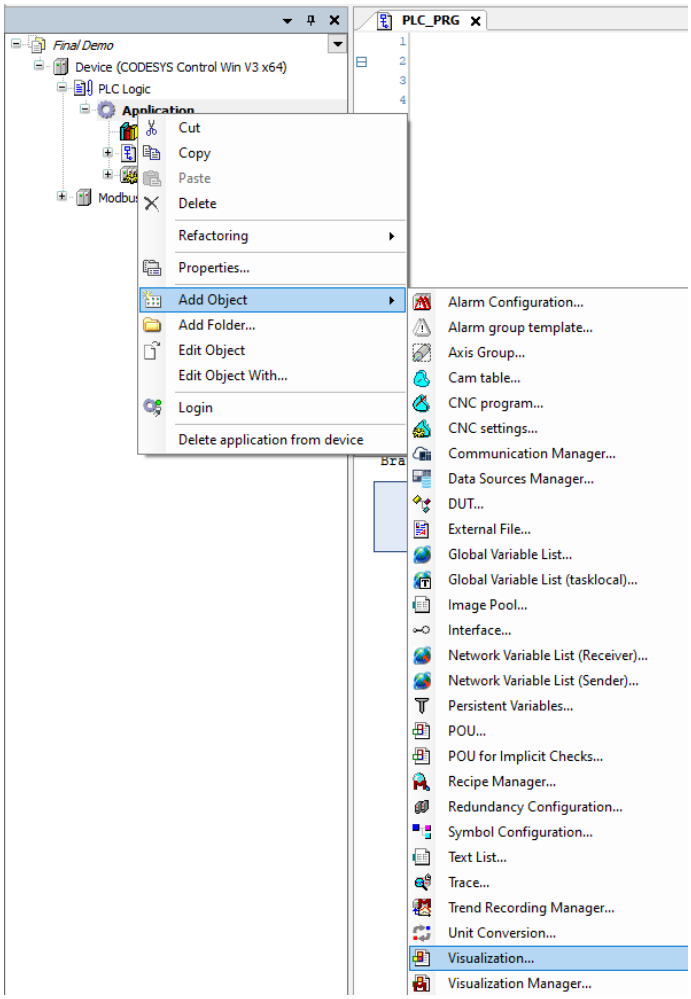


Figura 4.11 - Creación de pantalla de visualización.

Ahora pulsamos sobre la pantalla de visualización y arrastramos el ratón para crear el rectángulo. Una vez creado el rectángulo, al seleccionarlo, tendremos las propiedades de este a la derecha. Desde aquí podemos cambiar las dimensiones y la posición de la caja, además de poder cambiar el radio de curvatura de las esquinas.

Cambiamos las dimensiones para que sea de 128x128 y dejamos el radio de curvatura en 16 (ver figura 4.12). Ojo, al elegir el tipo de radio como explicito, hay que volver a seleccionar el rectángulo para que aparezca la fila de valor del radio.

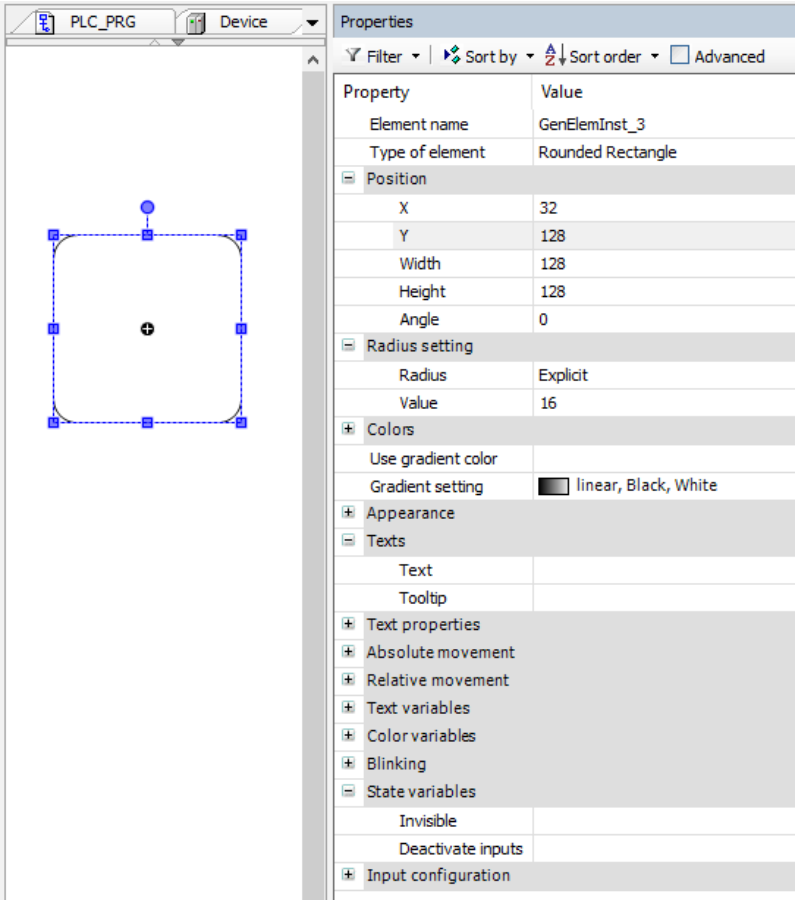


Figura 4.12 - Pantalla de propiedades de rectángulo redondeado.

Para crear la señal luminosa tenemos que volver a la ventana de herramientas eligiendo *Visualization ToolBox* en el menú que se encuentra por debajo de la ventana de propiedades. Una vez en la ventana de herramientas, elegiremos Lamps/Switches/Bitmaps y colocaremos un elemento *Lamp* dentro del rectángulo. En este punto tendremos una señal amarilla, así que entramos en las propiedades de la señal y cambiamos la propiedad imagen de *Yellow* a *Green*. Finalmente, y para que la señal se encienda con la variable "A", tendremos que escribir la variable en su campo de propiedades correspondientes, recordando hacer referencia al programa en el que se encuentra, "PLC_PRG" en ese caso. En la figura 4.13 se puede observar cómo quedarían las propiedades de la señal.

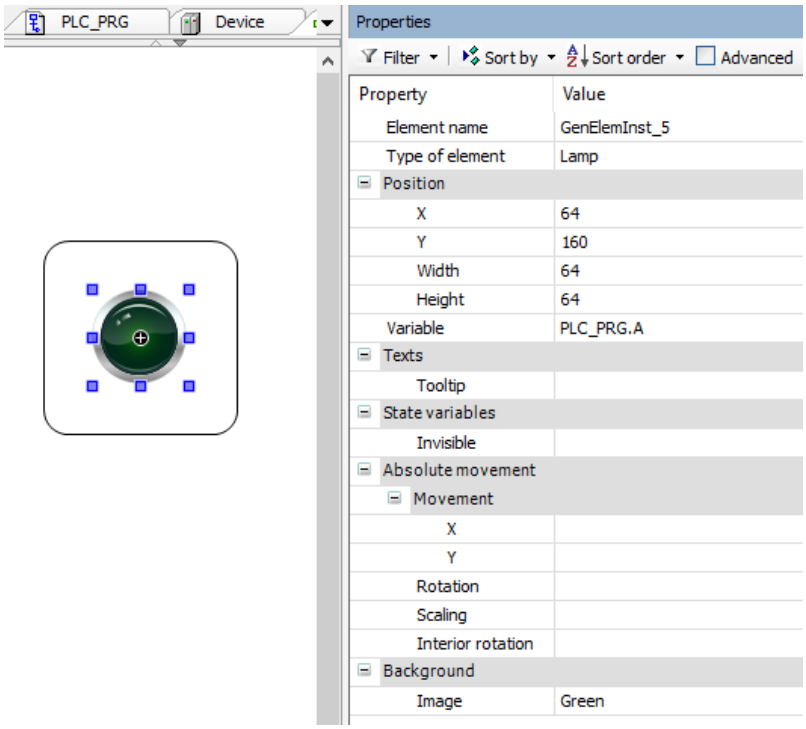


Figura 4.13 - Pantalla de propiedades de señal luminosa.

En este punto estamos preparados para cargar el programa a la PLC de nuevo y ver en la pantalla de visualización como la señal verde se enciende y se apaga cada segundo junto con la variable A.

4.3 Practica 2 - Programación de semáforo para vehículos

4.3.1 Introducción

Para esta práctica se entregarán un semáforo y un Arduino Mega 2560 con el programa de control por ModBus ya cargado a cada grupo.

El objetivo de esta práctica es que los alumnos se familiaricen con la comunicación por ModBus entre CoDeSys y el Arduino.

4.3.2 Semáforo

Los semáforos contienen 5 elementos diferenciados:

- Semáforo con:
  - Señales para vehículos.
  - Señal de giro para vehículos.
  - Señales para peatones.
- Pulsador.
- Bocina.
- Clavija con conexiones *DuPont* hembra.

En la figura 4.14 se puede ver la numeración de los pines de la clavija y en la tabla 4.1 la correspondencia entre los pines de esta y los elementos físicos del semáforo.

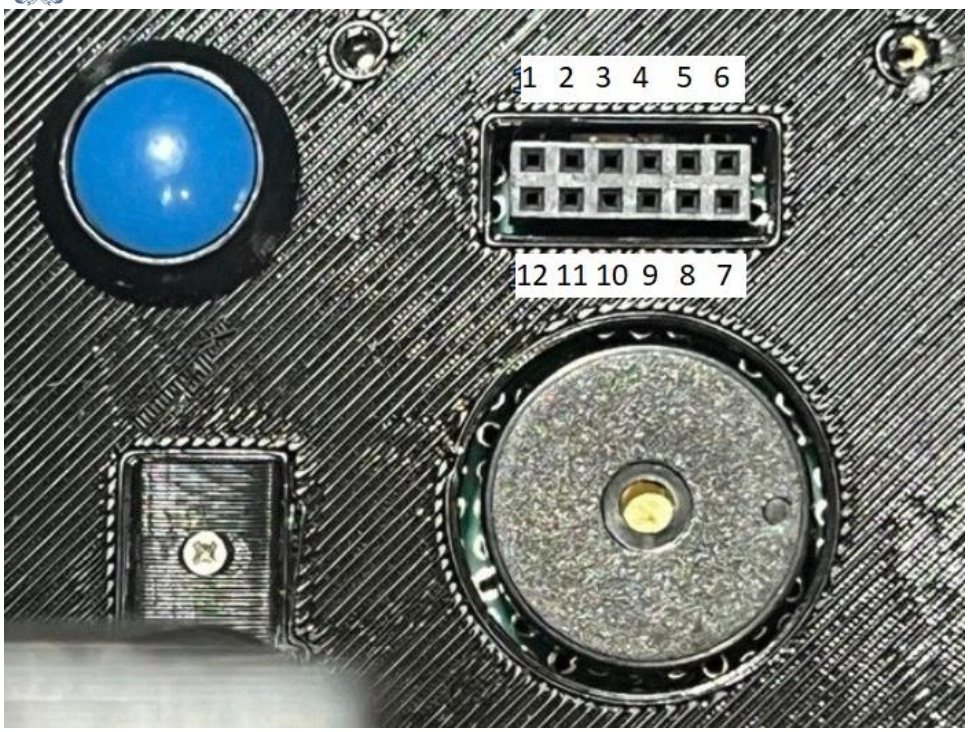


Figura 4.14 - Numeración de pines de semáforo.

Pin	Descripción
1	Bocina
2	Pulsador
3	Sin uso
4	Sin uso
5	5V [2]
6	GND [3]
7	Rojo vehículos
8	Ámbar vehículos
9	Verde vehículos
10	Giro derecha
11	Rojo peatones
12	Verde peatones

Tabla 4.1 - Correspondencia de pines y elementos físicos.

Se recomienda comprobar que todos los elementos funcionan, conectando la tierra del Arduino con el pin 6 y colocando los 5V en cada elemento excepto el pulsador. Para comprobar el funcionamiento del pulsador se coloca los 5V en el pin 5 y se mide la tensión en el pin 2 tanto en reposo como mientras se activa el pulsador. El pulsador debería de tener 5 V en reposo y 0 V mientras se está pulsando.

[2] La entrada de 5V soló es necesaria para el funcionamiento del pulsador.
[3] La entrada de GND es necesaria para el funcionamiento de todos los elementos.

4.3.3 Arduino Mega 2560

El Arduino Mega 2560 contiene un programa que permite la comunicación con CoDeSys mediante ModBus lo que permite usarlo como tarjeta de expansión de entradas y salidas (E/S) de la PLC virtual.

En la figura 4.15 se pueden ver los pines de la placa, junto a su referencia interna y a otras funcionalidades que puedan tener.

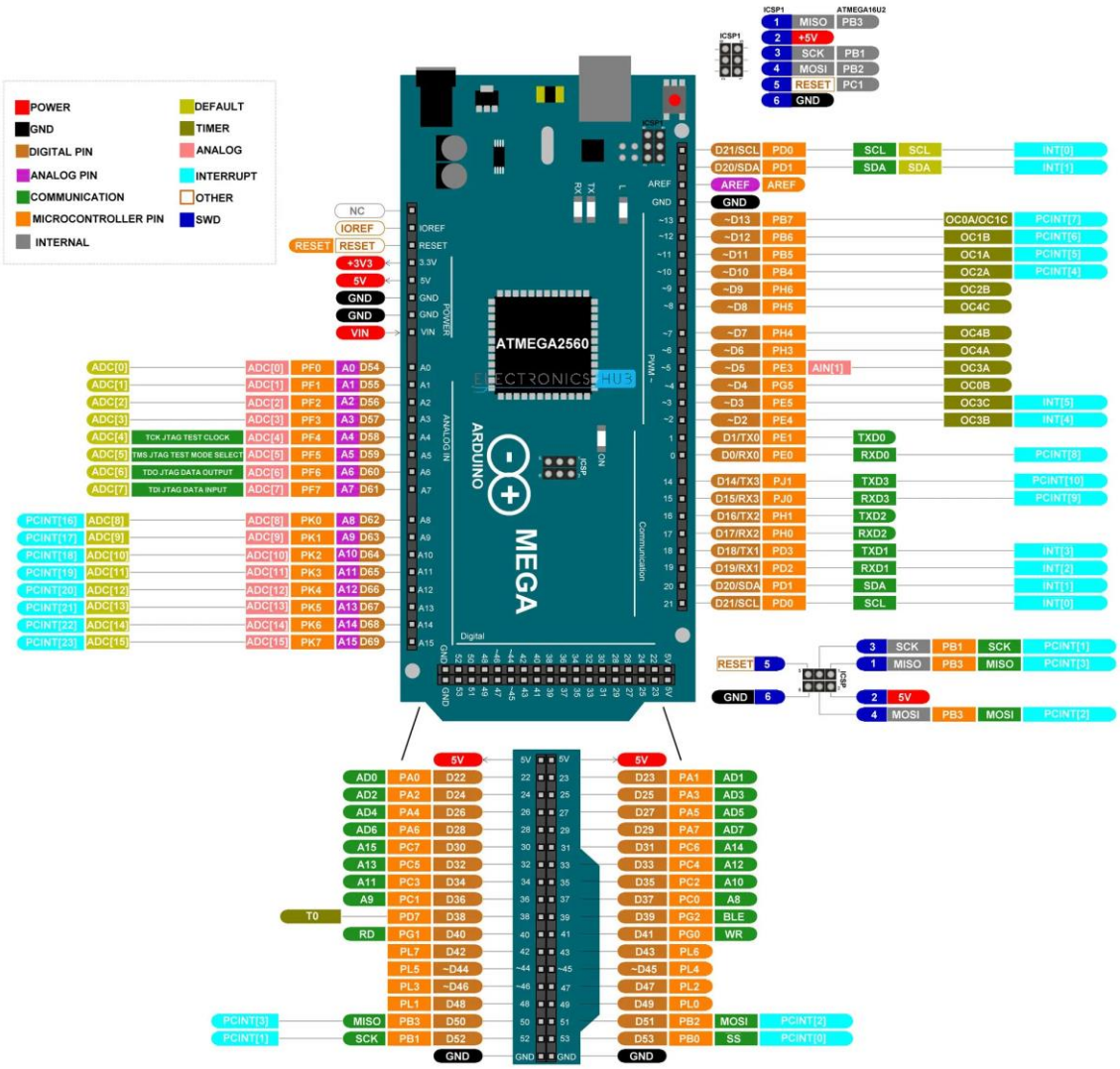


Figura 4.15 - *Pinout* de Arduino Mega 2560 [17]

En el programa antes mencionado, se han configurado una serie de entradas y salidas con una referencia para cada una con la que se tiene que acceder desde el CoDeSys, las cuales se pueden confirmar en la tabla 4.2.

Pin	Tipo	*Function Code*	Registro Decimal	Registro Hexadecimal
22	*Digital Input*	0x02	0	0x0000
23	*Digital Input*	0x02	1	0x0001
24	*Digital Input*	0x02	2	0x0002
25	*Digital Input*	0x02	3	0x0003
26	*Digital Output*	0x05	0	0x0000
27	*Digital Output*	0x05	1	0x0001
28	*Digital Output*	0x05	2	0x0002
29	*Digital Output*	0x05	3	0x0003
30	*Digital Output*	0x05	4	0x0004
31	*Digital Output*	0x05	5	0x0005
32	*Digital Output*	0x05	6	0x0006
33	*Digital Output*	0x05	7	0x0007
34	*Digital Output*	0x05	8	0x0008
35	*Digital Output*	0x05	9	0x0009
36	*Digital Output*	0x05	10	0x000A
37	*Digital Output*	0x05	11	0x000B
38	*Digital Output*	0x05	12	0x000C
39	*Digital Output*	0x05	13	0x000D
40	*Digital Output*	0x05	14	0x000E
41	*Digital Output*	0x05	15	0x000F
42	*Digital Output*	0x05	16	0x0010
43	*Digital Output*	0x05	17	0x0011
44	*Digital Output*	0x05	18	0x0012
45	*Digital Output*	0x05	19	0x0013
46	*Digital Output*	0x05	20	0x0014
47	*Digital Output*	0x05	21	0x0015
48	*Digital Output*	0x05	22	0x0016
49	*Digital Output*	0x05	23	0x0017
50	*Digital Output*	0x05	24	0x0018
51	*Digital Output*	0x05	25	0x0019
52	*Digital Output*	0x05	26	0x001A
53	*Digital Output*	0x05	27	0x001B

Tabla 4.2 - Correspondencia de pines y registros ModBus.

4.3.4 Conexión ModBus

ModBus es un protocolo de comunicaciones basado en la arquitectura maestro/esclavo diseñado en 1979 por Modicon para los PLCs de su propia marca. Hoy en día es uno de los protocolos de comunicación más extendidos en la industria.

Para utilizar este protocolo en nuestro PLC virtual empezamos por añadir un puerto ModBus pulsando con el botón derecho sobre *Device* y eligiendo la opción de añadir dispositivo (ver figura 4.16).

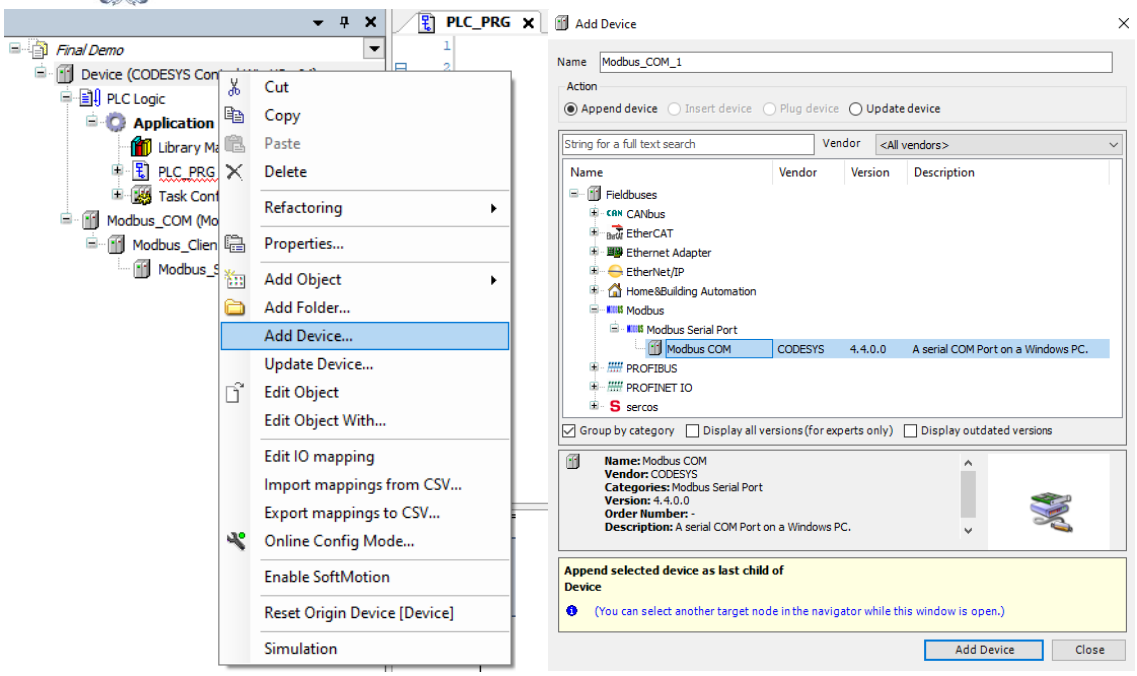


Figura 4.16 - Agregar puerto ModBus a CoDeSys.

Una vez agregado el puerto ModBus, este se tiene que configurar pulsando dos veces sobre el mismo y entrando en el apartado *General*. En este punto hay que configurar el puerto con la configuración utilizada por defecto en el firmware del Arduino (ver tabla 4.3). Hay que prestar especial atención a la hora de configurar el cliente ModBus y asegurarse que el puerto COM introducido es el puerto COM al que está conectado el Arduino. Para ello se puede abrir el administrador de dispositivo de Windows, sección "Puertos COM".

Parámetro	Valor
ID de Eslavo	1
Baudios	115200
Paridad	PAR
Numero de bits	8
Numero de bits de parada	1

Tabla 4.3 - Datos de la configuración ModBus en el Arduino.

Ahora añadimos un maestro de ModBus pulsando con el botón derecho sobre el puerto y una vez dentro de la pantalla de añadir dispositivo expandiendo el menú de *Modbus Serial Master.* Dentro del maestro de ModBus seguimos el mismo proceso para añadir un esclavo de ModBus expandiendo en este caso el menú de *Modbus Serial Slave*.

Confirmamos que el esclavo creado tiene el mismo ID (*server address* en el CoDeSys) que el Arduino y vamos al menú *Modbus Server Channel* dentro del esclavo. En este apartado podemos añadir puertos de entrada y salida (E/S), para ello añadimos un primer canal pulsando sobre *Add Channel* en la barra inferior y eligiendo el tipo de acceso y el offset en hexadecimal según la tabla 4.2. También se recomienda cambiar el nombre del canal por el nombre del PIN al que nos estaremos conectando. En la figura 4.15 podéis ver el ejemplo creando el canal para el pin D26, cuyo *Function Code* (FC) es el 0x05 y el offset de registro 0x0000.

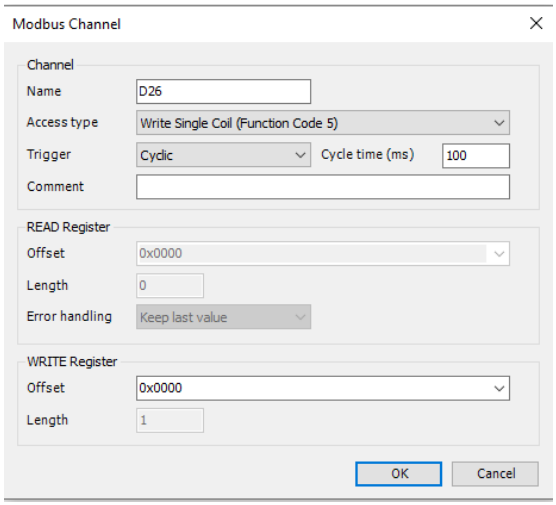


Figura 4.17 - Ejemplo de configuración de canal ModBus.

En este punto estamos listos para asignar la variable de la práctica anterior al canal que acabamos de crear, yendo al menú *ModbusGenericSerialServer I/O Mapping* dentro del esclavo y pulsando dos veces sobre el recuadro de "Variable" en el canal correspondiente. En vez de escribir la variable se puede pulsar sobre el botón con los puntos suspensivos pudiendo acceder a un árbol con la estructura del programa y pudiendo elegir la variable visualmente.

Ahora procedemos a conectar la tierra del Arduino al pin de GND del semáforo y el pin que hemos configurado en el canal al pin de la señal verde para vehículos del semáforo.

Finalmente, cargamos el programa en el PLC. Al poner el programa en funcionamiento se debería de ver el puerto, el maestro y el esclavo de ModBus con un simbolito verde a su izquierda señalizando que la conexión es correcta (ver figura 4.18).

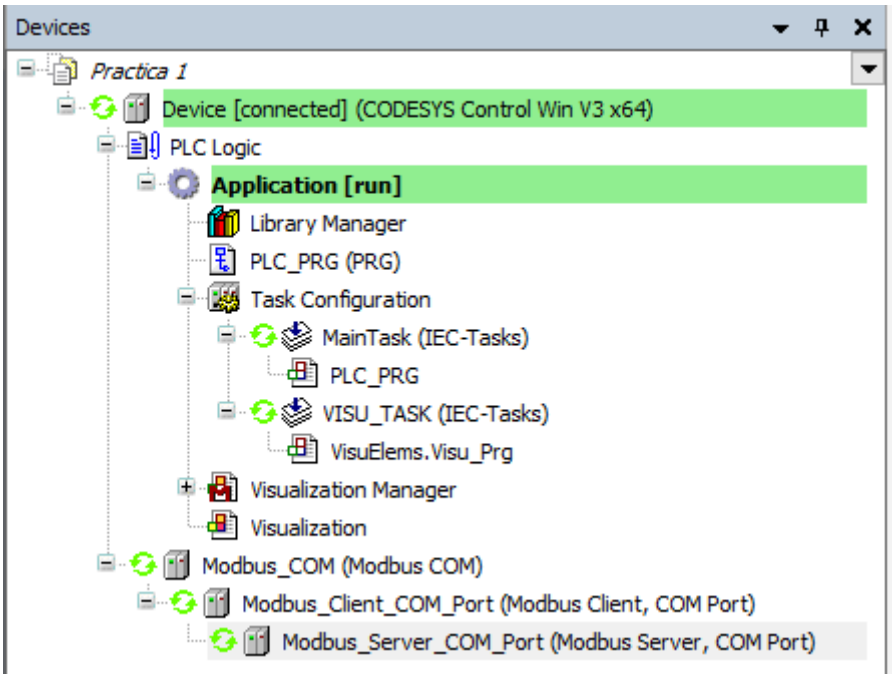


Figura 4.18 - Símbolos de estado en línea en el puerto, maestro y esclavo ModBus.

También, si es todo correcto, deberíamos de ver la luz verde vehículos encender y apagar con la periodicidad elegida en el programa además de ver lo mismo en la pantalla de visualización del CoDeSys.

4.3.5 Semáforo completo

Con los conocimientos adquiridos en clase sobre GRAFCET y en estas prácticas sobre CoDeSys y ModBus, programa un semáforo con el siguiente funcionamiento (resumen en la figura 4.19):

- El semáforo estará verde para vehículos durante 9 segundos, al mismo tiempo el semáforo estará rojo para peatones.
- Pasados los 9 segundos la parte de vehículos pasará a ámbar durante 1 segundo. Transcurrido ese tiempo se pasará a rojo.
- Pasado 1 segundo de tener ambas señales en rojo, la señal de peatones pasará a verde, manteniéndose ahí 5 segundos.
- Finalmente, la señal de peatones volverá a rojo y pasado 1 segundo volveremos al principio.

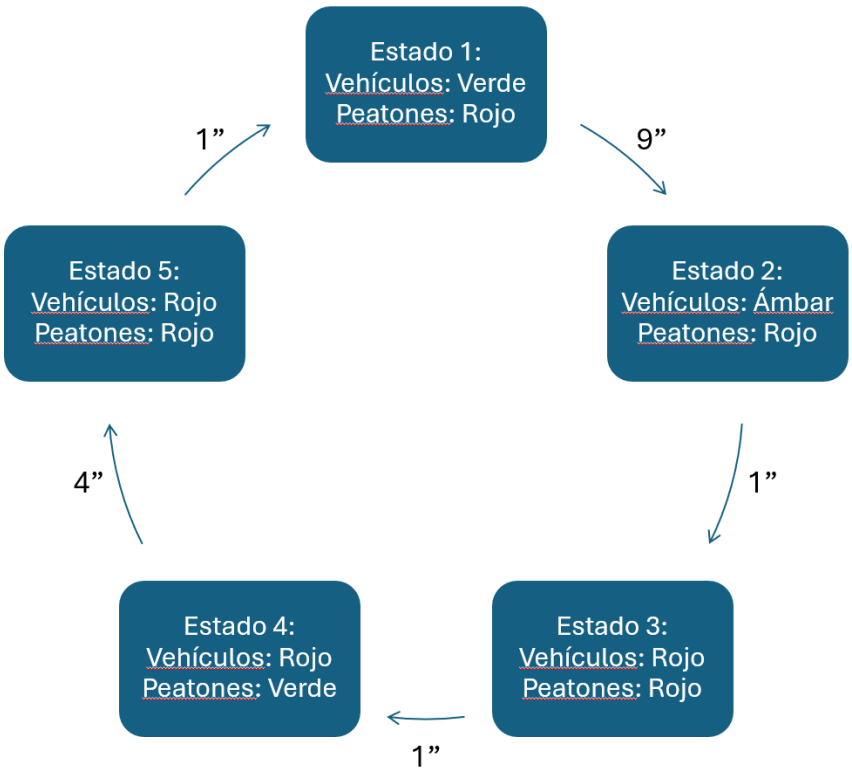


Figura 4.19 - Diagrama de funcionamiento de semáforo.

Adicionalmente, habrá que crear una pantalla de visualización en CoDeSys representando el semáforo y se tendrá que conectar el semáforo físico al CoDeSys por medio del Arduino pudiendo así comprobar que el semáforo funciona correctamente.

4.4 Practica 3 - Lectura de señales, ramificaciones y acciones activas

4.4.1 Introducción

Para esta práctica se entregarán un semáforo y un Arduino Mega 2560 con el programa de control por ModBus ya cargado a cada grupo.

El objetivo de la práctica es que los alumnos se familiaricen con el uso de entradas y flancos de subida o bajada, usar ramificaciones y aprender a crear funciones que se ejecutan antes, durante o después de un paso de SFC.

4.4.2 Lectura de señales

Para esta práctica empezaremos por conectar el pulsador del semáforo al Arduino, suministrando al semáforo con 5V y GND y conectando el pin del pulsador en uno de los 4 pines de entrada digital del Arduino.

Ahora seguimos los pasos descritos en la práctica anterior para crear un canal, pero esta vez elegiremos el código funcional 0x02 y el registro correspondiente al pin al que hayamos conectado el pulsador.

En el programa de la primera practica creamos una variable booleana, y hacemos con que las transiciones dependan de esta variable además de los temporizadores que ya teníamos uniendo ambas condiciones con un AND (ver ejemplo en la figura 4.20).

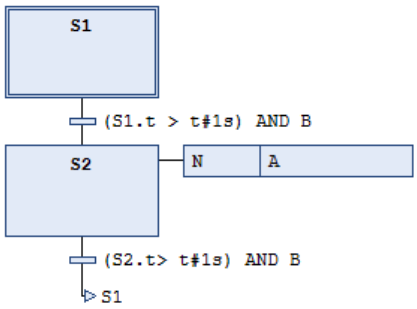


Figura 4.20 - Ejemplo de programa SFC con operaciones booleanas.

Finalmente asignamos la nueva variable al canal que configuramos con el pin del pulsador y cargamos el programa.

Una vez el programa está en marcha se debería de observar que el programa funciona como si el pulsador no existiera, pero si pulsamos el pulsador deberíamos de ver la nueva variable cambiar su valor a *FALSE* y el programa debería de detenerse mientras la variable este en este estado.

4.4.3 Acciones activas

Sin embargo, en aplicaciones industriales no se suele usar el valor de lectura como tal, sino que se aplican condiciones de flanco. Así podemos detectar cuando la variable pasa de *FALSE* a *TRUE* o viceversa.

Para ello hay que empezar por declarar una variable de tipo *F_TRIG* o *R_TRIG* según queramos un flanco de bajada o de subida.

Las variables de tipo flanco son bloques funcionales con 2 variables internas, *CLK* y *Q*. Siendo *CLK* la variable sobre la cual se va a detectar el flanco y *Q* la variable que se cambia a *TRUE* durante 1 ciclo de la PLC cuando está detecta el flanco y vuelve a *FALSE* en el siguiente ciclo.

En este ejercicio reemplazaremos las transiciones del ejercicio anterior por una transición de 5 segundos y otra con la variable *Q*, para ello, primero hay que mencionar la variable que se ha creado con tipo *F_TRIG* o *R_TRIG* seguido de un "punto" y el nombre de la variable interna, *Q* en este caso (ver figura 4.21).

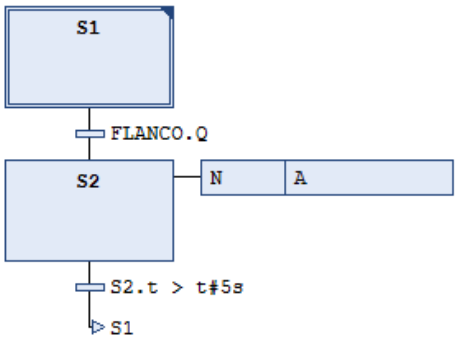


Figura 4.21 - Lectura de flanco en transición.

Para que la función detecte el flanco hay que asignarle el valor del pulsador, pero eso no se puede hacer en SFC. Entonces, creamos una acción en paso activo, para ello arrastramos una acción a la esquina superior derecha de un paso, siendo la esquina inferior izquierda para acciones que se ejecutan cuando se entra en el paso y la esquina inferior derecha para acciones que se ejecutan cuando se sale del paso.

Al soltar la acción se tiene que elegir como se programará. En este caso se usará *ST* (texto estructurado). Esto crea un nuevo programa dentro de PLC_PRG en el menú de la izquierda. Abrimos el programa y asignamos el valor del pulsador al CLK de la variable de flanco (ver figura 4.22).

Configuramos las señales de salida en los canales para que las luces verdes, ámbar y rojo coincidan con los LEDs para vehículos y cargamos el programa en el PLC virtual, observando que el semáforo se detiene en rojo hasta que se pulsa el pulsador, momento en el que se pasa a verde durante 5 segundos y ámbar durante 1 segundo.

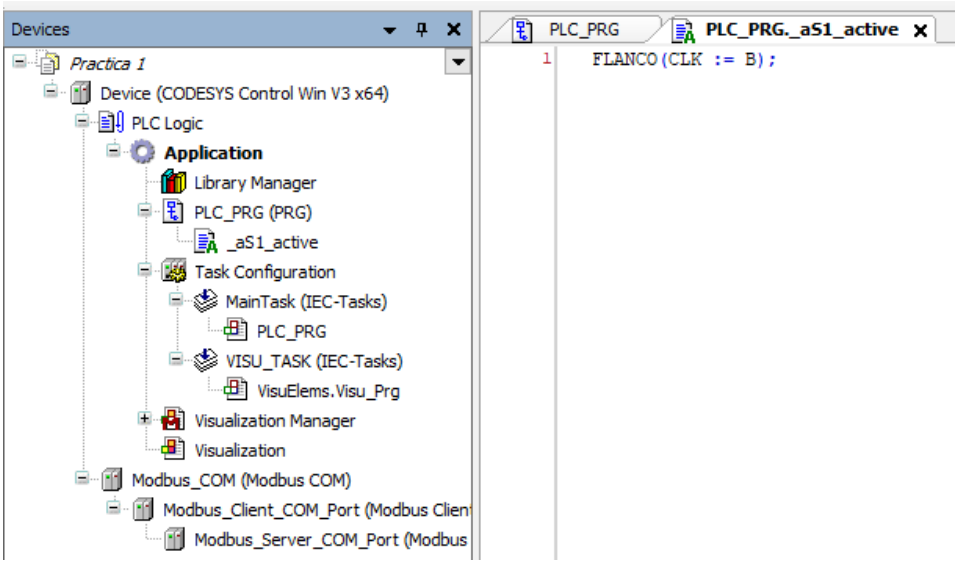


Figura 4.22 - Ejemplo asignando un valor a la variable CLK.

Finalmente cargamos el programa que debería de permanecer en la etapa inicial hasta que detecte que pulsamos el pulsador, esperar 5 segundos y volver al estado inicial.

Las acciones activas, de entrada, o de salida son muy útiles en un gran rango de aplicaciones, pudiendo también crearse del tipo *SFC*.

4.4.4 Ramificaciones

La última herramienta que tenemos a nuestra disposición son las ramificaciones, etapas que se ejecutan a la vez.

Para crear una ramificación empezamos por arrastrar la herramienta "*Branch*" al punto en el programa donde queremos que empiece la ramificación y después pulsar sobre el punto en el que queremos que termine.

Las ramificaciones tienen que cumplir ciertos requisitos:

- El principio y final de cada rama tienen que ser etapas.
- Habrá una única transición antes de entrar y otra para salir de la ramificación.
- No se puede inicializar el programa dentro de una ramificación

Para probar el uso de las ramificaciones se va a modificar el ejercicio anterior creando una ramificación paralela al estado que espera al flanco del pulsador. En la nueva rama creada se va a tener dos estados, el primero accionando una luz ámbar durante 1 segundo y el segundo dejando la luz en rojo hasta que se detecta el flanco y salimos de la ramificación.

Adicionalmente, si la ramificación es el inicio del programa, habrá que crear un estado adicional vacío con una transición con "*TRUE*" para usarla en el salto del final del programa. Para elegir este estado como el de inicio de programa hay que pulsar sobre el botón derecho encima de este y elegir "*Init Step*".

En la figura 4.23 se puede ver como quedaría el programa terminado.

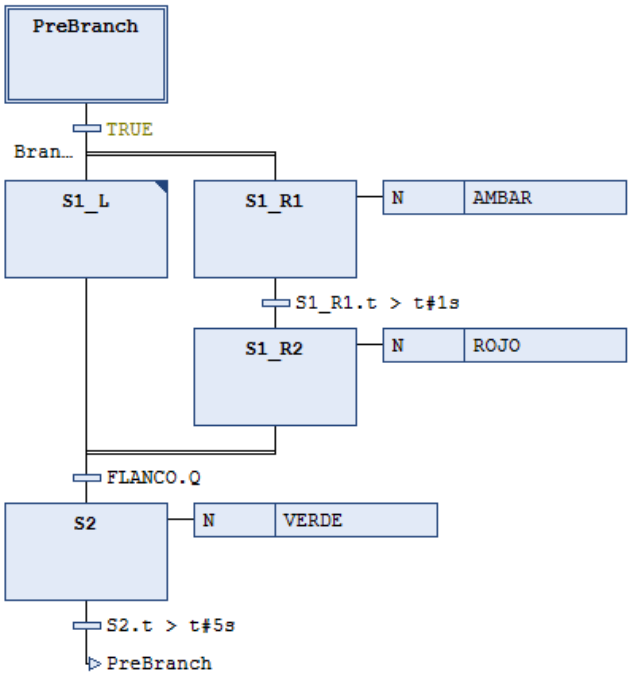


Figura 4.23 - Ejemplo de programa SFC con una ramificación.

4.5 Practica 4 - Programación de cruce semafórico

4.5.1 Introducción

Para esta práctica se entregarán cuatro semáforos y un Arduino Mega 2560 con el programa de control por ModBus ya cargado a cada grupo.

El objetivo de esta práctica es poner a prueba todos los conocimientos adquiridos por los alumnos, retándolos a programar un cruce semafórico completo.

4.5.2 Cruce semafórico

Con los conocimientos adquiridos en clase sobre GRAFCET y en estas prácticas sobre CoDeSys y ModBus, programa un cruce semafórico con el siguiente funcionamiento (resumen en la figura 4.23):

- El cruce estará compuesto de dos vías, ambas con dos direcciones y cada dirección a su vez con 1 semáforo, pudiendo así identificar cada semáforo según la dirección que controla (norte, sur, este, oeste).

- Al empezar todos los semáforos estarán con las luces rojas de peatones y vehículos encendidas durante 5 segundos.

- Pasado este tiempo, los semáforos norte y sur pasarán a estar verde para vehículos mientras que los semáforos este y oeste pasarán a estar verde para peatones durante 1 minuto.

- Al terminar el minuto volveremos a estar 5 segundos con todo en rojo, y repetimos el proceso, pero a la inversa.

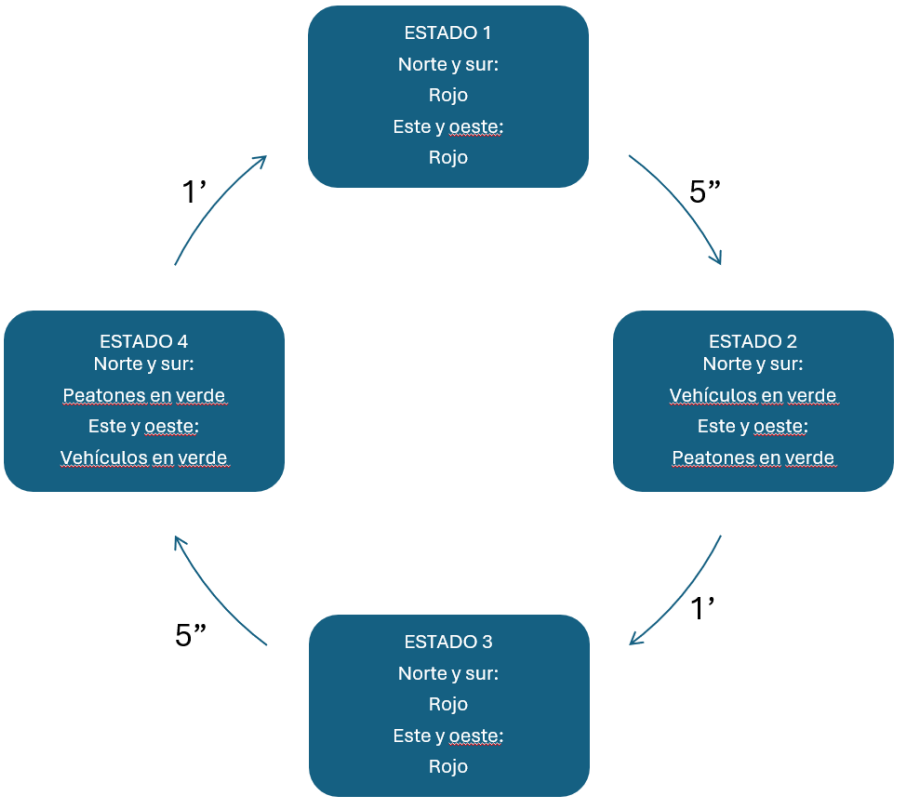


Figura 4.23 - Diagrama de funcionamiento de cruce.

- A 5 segundos de terminar el verde para vehículos, se apagará la señal verde y se encenderá la señal ámbar de vehículos. Estando verde para los vehículos, si se activa el pulsador, se comienza el proceso de pasar a rojo para vehículos, teniendo que durar el verde (y ámbar) para vehículos mínimo 30 segundos.

- En cuanto a los semáforos en verde para peatones:
  - Al ponerse en verde para peatones, la bocina debe de sonar durante 3 segundos.
  - Mientras el semáforo está en verde para peatones, la señalización de giro a derecha para vehículos se encenderá y se apagará intermitentemente durando cada estado 1 segundo.
  - Los últimos 10 segundos de verde para peatones, la señal verde para peatones se apagará y se encenderá cada durando cada estado 1 segundo.
  - Al terminar el tiempo y pasar el semáforo a rojo para peatones la bocina pitará 3 veces durante 1 segundo de cada vez y dejando 1 segundo de silencio entre medias de cada pitido.

Adicionalmente, en CoDeSys habrá que crear una pantalla de visualización representando un semáforo en cada vía y se tendrá que conectar los cuatro semáforos físicamente al CoDeSys por medio del Arduino pudiendo así comprobar que el cruce funciona correctamente.

5

Conclusiones y Mejoras

5.1 Conclusiones

En este proyecto se ha diseñado un semáforo a escala para su uso como material de prácticas en asignaturas de Grado. Se ha tenido que diseñar la electrónica atendiendo a unos criterios limitantes de potencia. Mecánicamente se diseñó el semáforo a escala 1:20 teniendo una altura de 20 centímetros y se fabricó mediante impresión 3D.

Finalmente se programó un Arduino Mega 2560 para que se comunicará con PLCs por ModBus haciendo uso de una librería de código abierto [18]. El código fuente de Arduino se encuentra en el apartado 7.2. Con estos pasos se han cumplido los objetivos principales del proyecto pudiendo simular un cruce semafórico por 132,26€ en materiales.

Adicionalmente, se han diseñado una serie de prácticas tutorizadas con el objetivo de enseñar a los alumnos a programar con el lenguaje SFC cuyos fundamentos son equivalentes a los diagramas GRAFCET que se imparten en la asignatura de Automática.

En estas prácticas se ha empezado por explicar cómo funciona el entorno de programación de PLCs CoDeSys, guiándolos en la creación de un programa básico con dos etapas y temporizadores. También se les enseña a crear una pantalla de visualización en la cual se puede plasmar la información más importante del programa para verla en tiempo real. A continuación, se explica a los alumnos el funcionamiento del prototipo desarrollado en este proyecto, guiándolos en la conexión ModBus entre CoDeSys y el mismo.

En este punto se ha diseñado un ejercicio no tutorizado en el que los alumnos tendrán que programar un único semáforo con funcionalidades reducidas. Una vez superado este ejercicio, se enseña a los alumnos a utilizar herramientas de SFC más avanzadas como la lectura de señales, los flancos de subida y/o bajada, las acciones de entrada, salida o activas y las ramificaciones de etapas.

Finalmente se valora el aprendizaje del alumno con un ejercicio final en el que tienen que combinar todos los conocimientos adquiridos para programar un cruce semafórico completo haciendo uso de todas las funcionalidades que se han incluido en el proyecto.

De este modo se han cumplido también los objetivos secundarios del proyecto, permitiendo así que los alumnos de la Escuela de Ingenierías Industriales de la Universidad de Málaga puedan experimentar en primera persona con la programación de PLCs en el transcurso del segundo curso.

El desarrollo de este proyecto ha supuesto una serie de retos con oportunidades de aprendizaje de nuevas habilidades aplicables a distintos campos de la ingeniería. A continuación, se enumerarán las habilidades adquiridas más relevantes:

- **Diseño 3D:** El proyecto ha requerido trabajar con programas de diseño 3D y buscar soluciones a distintos problemas de tamaño, Integración, fabricación, etc. Se ha aprovechado la oportunidad para aprender a trabajar con SolidWorks.

- **Depuración de fallos electrónicos:** En las distintas fases de diseño y fabricación han surgido fallos electrónicos que han requerido un trabajo de depuración al cual los alumnos no están acostumbrados.

- **Integración electromecánica:** Al Incluir tanto el diseño electrónico como el diseño mecánico del proyecto, se ha podido experimentar en primera persona los retos a la hora de Integrar una cantidad moderada de componentes en el espacio más reducido posible.

- **Docencia:** Diseñar las practicas ha permitido que el alumno se situará en la situación de un docente e intentará optimizar tanto los conocimientos impartidos como la dificultad de los ejercicios.

5.2 Mejoras

Aun estando satisfecho con el resultado final del proyecto, se considera que hay varias posibilidades de mejora:

- **Diseño de PCB:** Diseñar una PCB para integrar los componentes electrónicos no solo permitiría fabricar una base más compacta, sino que también facilitará el proceso de montaje pudiendo reducir el tiempo de montaje de la placa electrónica sustancialmente.

- **Desarrollo de *shield*:** Desarrollando un *shield* y una carcasa para Arduino se podría obtener un producto más similar a una tarjeta de E/S ModBus, con los puertos de E/S numerados según la referencia de acceso. Incluso se podría estudiar el desarrollo de una placa PCB con el microcontrolador ATmega2560 pudiéndolo programar desde el conector ICSP de un Arduino y así ahorrar en costes.

- **Conexiones eléctricas atornilladas:** Aunque las conexiones DuPont permiten un prototipado rápido, también pueden causar desconexiones inesperadas e incluso cortocircuitos si varios cables se desconectan y sus puntas entran en contacto. Adicionalmente, la conexión de los cables del semáforo a la placa que se encuentra en su base también podría ser realizada con el mismo método en vez de soldar los cables directamente a la PCB.

- **Rediseño mecánico:** El diseño del prototipo actual supone una serie de rectos a la hora de montaje y/o mantenimiento, siendo muy difícil reemplazar LEDs que dejen de funcionar entre otras cosas.

- **Semáforos adicionales:** Los semáforos desarrollados en este proyecto están diseñados para ser semáforos de entrada en un cruce, sin embargo, también debería de haber semáforos de salida de cruce, pudiendo controlar un cruce entero con los 8 semáforos con un Arduino Mega 2560.

6

Bibliografía

[1] Plan de Estudios. Grado en Ingeniería en Tecnologías Industriales. Escuela de Ingenierías Industriales, Universidad de Málaga. Consultado el 22/11/2024 en: https://www.uma.es/media/files/Plan_de_Estudios_Grado_en_Ingenieria_en_Tecnologias_Industriales.pdf

[2] Esquemas de los Títulos de Grado de la EII, Universidad de Málaga. Consultado el 22/11/2024 en: https://www.uma.es/escuela-de-ingenierias-industriales/info/107671/esquemas-de-los-titulos-de-grado-de-la-eii/

[3] *International Electrotechnical Commission. "Programmable controllers - Part 3: Programming languages." IEC 61131-3:2013. Geneva, Switzerland: International Electrotechnical Commission, 2013.*

[4] SETGA NX SEMÁFORO. Consultado el 22/11/2024 en https://setga.es/productos/semaforo-nx/.

[5] Página de descargas de *software* de la familia Arduino. Consultado el 22/11/2024 en: https://www.arduino.cc/en/software

[6] Página de descargas de *software* de CoDeSys. Consultado el 22/11/2024 en: https://www.codesys.com/download.html

[7] Página de descarga de *software* PrusaSlicer. Consultado el 22/11/2024 en: https://www.prusa3d.com/page/prusaslicer_424/

[8] *Arduino Mega 2560 Rev3 Product Reference Manual*, pag. 8. Consultado el 22/11/2024 en: https://docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf

[9] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, *Universal Serial Bus Specification Revision 2.0*, pag. 178.

[10] *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V [DATASHEET],* pag. 356. Disponible en: https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

[11] *Atmel ATmega8U2/16U2/32U2 [DATASHEET]*, pag. 265. Consultado el 22/11/2024 en: https://ww1.microchip.com/downloads/en/DeviceDoc/doc7799.pdf

[12] *HLMP-4700, HLMP-4719, HLMP-4740 HLMP-1700, HLMP-1719, HLMP-1790 T-13/4 (5 mm), T-1 (3 mm), Low Current LED Lamps*, pags. 1-5. Consultado el 22/11/2024 en: https://www.mouser.es/datasheet/2/678/AV02_1557EN_2021_08_09-3007319.pdf

[13] *Same sky CPT-2207-90T PIEZO BUZZER TRANSDUCER,* pag. 1. Consultado el 22/11/2024 en: https://www.mouser.es/datasheet/2/1628/cpt_2207_90t-3510064.pdf

[14] *TEXAS INSTRUMENTS LM555 SNAS5548D - FEBRUARY 2000 - REVISED JANUARY 2015,* pags. 5, 10-11. Consultado el 22/11/2024 en: https://www.ti.com/lit/ds/symlink/lm555.pdf

[15] CODESYS Essentials · Reference: Programming · Operands · Constant: TIME, LTIME. Consultado el 22/11/2024 en: https://content.helpme-codesys.com/en/CODESYS%20Development%20System/_cds_operands_constant_time.html

[16] CODESYS Essentials · Reference: Programming · Qualifiers for Actions in SFC Consultado el 22/11/2024 en: https://content.helpme-codesys.com/en/CODESYS%20SFC/_cds_sfc_action_qualifier.html

[17] ElectronicsHub, Arduino Mega Pinout (2560 Pin Diagram & Specifications). Consultado el 22/11/2024 en https://www.electronicshub.org/arduino-mega-pinout/

[18] epsilonrt, *Modbus Serial Library for Arduino*. Consultado el 22/11/2024 en: https://github.com/epsilonrt/modbus-serial

7

Código fuente

7.1 Código de Arduino del primer prototipo

El código mostrado en este anexo esta desarrollado en el entorno de programación de Arduino siendo un programa que simula el funcionamiento de un semáforo.

```
1   #define CAR_RED 22
2   #define CAR_AMBER 23
3   #define CAR_GREEN 24
4   #define CAR_RIGHT 25
5
6   #define PEDESTRIAN_RED 26
7   #define PEDESTRIAN_GREEN 27
8   #define PEDESTRIAN_BUTTON 28
9   #define PEDESTRIAN_BEEP 29
10
11  #define CAR_GREEN_TIME 5000
12  #define CAR_RED_TIME 5000
13  #define SAFE_TIME 1000
14
15  uint64_t stateTimer;
16  uint64_t currentTime;
17
18  uint8_t stateVar;
19
20  void setup() {
21    // put your setup code here, to run once:
22    pinMode(CAR_RED, OUTPUT);
23    pinMode(CAR_AMBER, OUTPUT);
```

```
24    pinMode(CAR_GREEN, OUTPUT);
25    pinMode(CAR_RIGHT, OUTPUT);
26    pinMode(PEDESTRIAN_RED, OUTPUT);
27    pinMode(PEDESTRIAN_GREEN, OUTPUT);
28    pinMode(PEDESTRIAN_BEEP, OUTPUT);
29    pinMode(PEDESTRIAN_BUTTON, INPUT_PULLUP);
30                  attachInterrupt(digitalPinToInterrupt(PEDESTRIAN_BUTTON),
31  Run_ButtonPressed, FALLING);
32
33    stateTimer = millis();
34    stateVar = 1;
35  }
36
37  void loop() {
38    // put your main code here, to run repeatedly:
39    switch (stateVar) {
40      case 1:
41        Run_State1();
42        break;
43      case 2:
44        Run_State2();
45        break;
46      case 3:
47        Run_State3();
48        break;
49      case 4:
50        Run_State4();
51        break;
52      case 5:
53        Run_State5();
54        break;
55    }
56  }
57
58  void Run_State1() {
59    if (millis() - stateTimer > CAR_GREEN_TIME - SAFE_TIME) {
60      stateVar = 2;
61      stateTimer = millis();
62    } else {
63      digitalWrite(CAR_RED, LOW);
64      digitalWrite(CAR_GREEN, HIGH);
65      digitalWrite(CAR_RIGHT, LOW);
66      digitalWrite(PEDESTRIAN_RED, HIGH);
67      digitalWrite(PEDESTRIAN_GREEN, LOW);
68      digitalWrite(PEDESTRIAN_BEEP, LOW);
69    }
70  }
71
72  void Run_State2() {
73    if (millis() - stateTimer > SAFE_TIME) {
```

```
74        stateVar = 3;
75        stateTimer = millis();
76      } else {
77        digitalWrite(CAR_AMBER, HIGH);
78        digitalWrite(CAR_GREEN, LOW);
79      }
80    }
81
82    void Run_State3() {
83      if (millis() - stateTimer > SAFE_TIME) {
84        stateVar = 4;
85        stateTimer = millis();
86      } else {
87        digitalWrite(CAR_RED, HIGH);
88        digitalWrite(CAR_AMBER, LOW);
89      }
90    }
91
92    void Run_State4() {
93      if (millis() - stateTimer > CAR_RED_TIME) {
94        stateVar = 5;
95        stateTimer = millis();
96      } else {
97        digitalWrite(PEDESTRIAN_RED, LOW);
98        digitalWrite(PEDESTRIAN_GREEN, HIGH);
99        if ((millis() - stateTimer) % 400 < 200) {
100         digitalWrite(CAR_RIGHT, HIGH);
101         tone(PEDESTRIAN_BEEP,4000);
102       } else {
103         digitalWrite(CAR_RIGHT, LOW);
104         noTone(PEDESTRIAN_BEEP);
105       }
106     }
107   }
108
109   void Run_State5() {
110     if (millis() - stateTimer > SAFE_TIME) {
111       stateVar = 1;
112       stateTimer = millis();
113     } else {
114       digitalWrite(PEDESTRIAN_RED, HIGH);
115       digitalWrite(PEDESTRIAN_GREEN, LOW);
116       digitalWrite(CAR_RIGHT, LOW);
117       noTone(PEDESTRIAN_BEEP);
118     }
119   }
120
121   void Run_ButtonPressed() {
122     if (stateVar == 1) {
123       stateVar = 2;
```

```
124       stateTimer = millis();
125     }
126   }
```

7.2 Código de comunicación ModBus

El código mostrado en este anexo esta desarrollado en el entorno de programación de Arduino siendo un programa que habilita la comunicación ModBus a través del puerto serial asignado a la Interfaz USB. Indicando al propio Arduino el tipo de cada pin (entrada o salida digital) y la dirección con la que se accede desde otro dispositivo. Para ello se ha usado una librería de código abierto que contiene la lógica para la comunicación ModBus.

```
1   /**
2       @file modbusKernel.ino
3       @version 4
4       @author Vicente Arévalo, Daniel Silva
5
6       This software enables arduino to work as a modbus slave with:
7         - 4 digital inputs and,
8         - 24 digital outputs
9       by using a RS232 connection for the communication with the master.
10
11      RS232 setup:
12      ---------------------------------
13      |      Parameter      |   Value   |
14      ---------------------------------
15      |  Slave ID           |       1   |
16      |  Bauds              |  115200   |
17      |  Parity             |    EVEN   |
18      |  Num of bits        |       8   |
19      |  Num of stop bits   |       1   |
20      ---------------------------------
21
22      ----------------------------------------
23      | Modbus FC |        Operation         |
24      ----------------------------------------
25      |   0x02    |   Read Discrete Inputs   |
26      |   0x05    |   Write Single Coil      |
27      ----------------------------------------
28  */
29
30  #include <ModbusSerial.h>
31
32  // IO card data
33  const byte SlaveId = 1;
34  const unsigned long BaudRate = 115200;
35
36  // IO map (arduino pin <-> modbus register)
```

```
37  struct ioMap
38  {
39      int pin;
40      int reg;
41  };
42
43  ioMap digitalInput[]  = { {22, 0}, {23, 1}, {24, 2}, {25, 3} };
44  ioMap digitalOutput[] = {
45    {26, 0}, {27, 1}, {28, 2}, {29, 3}, {30, 4}, {31, 5}, {32, 6}, {33, 7},
46    {34, 8}, {35, 9}, {36, 10}, {37, 11}, {38, 12}, {39, 13}, {40, 14}, {41,
47  15}, {42, 16}, {43, 17}, {44, 18}, {45, 19}, {46, 20}, {47, 21}, {48, 22},
48  {49, 23}, {50, 24}, {51, 25}, {52, 26}, {53, 27}
49  };
50
51  const int TxenPin = -1; // -1 disables the feature, change that if you are
52  using an RS485 driver, this pin would be connected to the DE and /RE pins
53  of the driver.
54
55  // modbus master communication
56  #define MySerial Serial // define serial port used, Serial most of the
57  time, or Serial1, Serial2 ... if it is available
58
59  // modbus serial object
60  ModbusSerial mb(MySerial, SlaveId, TxenPin);
61
62  //  setup the modbus serial object & I/O registers
63  void setup_modbus()
64  {
65    MySerial.begin(BaudRate, MB_PARITY_EVEN); // 8 bits, even parity, 1 stop
66  bit
67    while(!MySerial);
68
69    mb.config(BaudRate);
70    mb.setAdditionalServerData("Serial MODBUS IO card"); // for Report Server
71  ID function (0x11)
72
73    // setup registers for analog/digital inputs/outputs
74    for (int i = 0; i < 4; i++)
75    {
76      mb.addIsts(digitalInput[i].reg);
77    }
78
79    for (int i = 0; i < 28; i++)
80    {
81      mb.addCoil(digitalOutput[i].reg);
82    }
83  }
84
85  //  setup the arduino I/O pines
86  void setup_pines()
```

```
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(digitalInput[i].pin,  INPUT);
  }

  for (int i = 0; i < 28; i++)
  {
    pinMode(digitalOutput[i].pin, OUTPUT);
  };
}

void myDigitalWrite(int idx)
{
  bool val = mb.Coil(digitalOutput[idx].reg);
  digitalWrite(digitalOutput[idx].pin, val);
}

void myDigitalRead(int idx)
{
  bool val = digitalRead(digitalInput[idx].pin);
  mb.Ists(digitalInput[idx].reg, val);
}

void setup()
{
  setup_modbus();
  setup_pines();
}

void loop()
{
  // Call once inside loop() - all magic here
  mb.task();

  // read data from the arduino inputs
  for(int i = 0; i < 4; i++)
  {
    myDigitalRead(i);
  }

  // write data to the arduino outputs
  for(int i = 0; i < 28; i++)
  {
    myDigitalWrite(i);
  }
}
```

7.3 Código de CoDeSys para cruce semafórico completo

En este anexo se mostrará la solución de la ultima práctica en la que se propone programar un cruce semafórico completo. Se incluirán tanto las partes de ST (texto estructurado) como las partes de SFC.

PLC_PRG

Texto

```
1   PROGRAM PLC_PRG
2   VAR
3           /// Semaphore 1
4           S1_VR: BOOL; // Semaphore 1 Vehicles Red
5           S1_VA: BOOL; // Semaphore 1 Vehicles Amber
6           S1_VG: BOOL; // Semaphore 1 Vehicles Green
7           S1_VTR: BOOL; // Semaphore 1 Vehicles Turn Right
8           S1_PR: BOOL; // Semaphore 1 Pedestrians Red
9           S1_PG: BOOL; // Semaphore 1 Pedestrians Green
10          S1_PB: BOOL; // Semaphore 1 Pedestrians Beep
11          S1_PPB: BOOL; // Semaphore 1 Pedestrians Push Button
12          /// Semaphore 2
13          S2_VR: BOOL; // Semaphore 2 Vehicles Red
14          S2_VA: BOOL; // Semaphore 2 Vehicles Amber
15          S2_VG: BOOL; // Semaphore 2 Vehicles Green
16          S2_VTR: BOOL; // Semaphore 2 Vehicles Turn Right
17          S2_PR: BOOL; // Semaphore 2 Pedestrians Red
18          S2_PG: BOOL; // Semaphore 2 Pedestrians Green
19          S2_PB: BOOL; // Semaphore 2 Pedestrians Beep
20          S2_PPB: BOOL; // Semaphore 2 Pedestrians Push Button
21          /// Semaphore 3
22          S3_VR: BOOL; // Semaphore 3 Vehicles Red
23          S3_VA: BOOL; // Semaphore 3 Vehicles Amber
24          S3_VG: BOOL; // Semaphore 3 Vehicles Green
25          S3_VTR: BOOL; // Semaphore 3 Vehicles Turn Right
26          S3_PR: BOOL; // Semaphore 3 Pedestrians Red
27          S3_PG: BOOL; // Semaphore 3 Pedestrians Green
28          S3_PB: BOOL; // Semaphore 3 Pedestrians Beep
29          S3_PPB: BOOL; // Semaphore 3 Pedestrians Push Button
30          /// Semaphore 4
31          S4_VR: BOOL; // Semaphore 4 Vehicles Red
32          S4_VA: BOOL; // Semaphore 4 Vehicles Amber
33          S4_VG: BOOL; // Semaphore 4 Vehicles Green
34          S4_VTR: BOOL; // Semaphore 4 Vehicles Turn Right
35          S4_PR: BOOL; // Semaphore 4 Pedestrians Red
36          S4_PG: BOOL; // Semaphore 4 Pedestrians Green
37          S4_PB: BOOL; // Semaphore 4 Pedestrians Beep
38          S4_PPB: BOOL; // Semaphore 4 Pedestrians Push Button
```

```
39          /// Falling Triggers for Push Buttons
40          FTRIG1 : F_TRIG; // Semaphore 1 Falling Trigger
41          FTRIG2 : F_TRIG; // Semaphore 2 Falling Trigger
42          FTRIG3 : F_TRIG; // Semaphore 3 Falling Trigger
43          FTRIG4 : F_TRIG; // Semaphore 4 Falling Trigger
44          // Flags for signalizing a trigger was detected when the semaphore can't update
45  step yet
46          TRIG1 : BOOL; // Semaphore 1 & 3 Trigger Detected Flag
47          TRIG2 : BOOL; // Semaphore 2 & 4 Trigger Detected Flag
48          // Flag to restart active functions once per cycle
49          FLAG : BOOL; // Flag to restart active functions once per cycle
50  END_VAR
```

SFC (compuesto por figuras 7.1, 7.2 y 7.3)

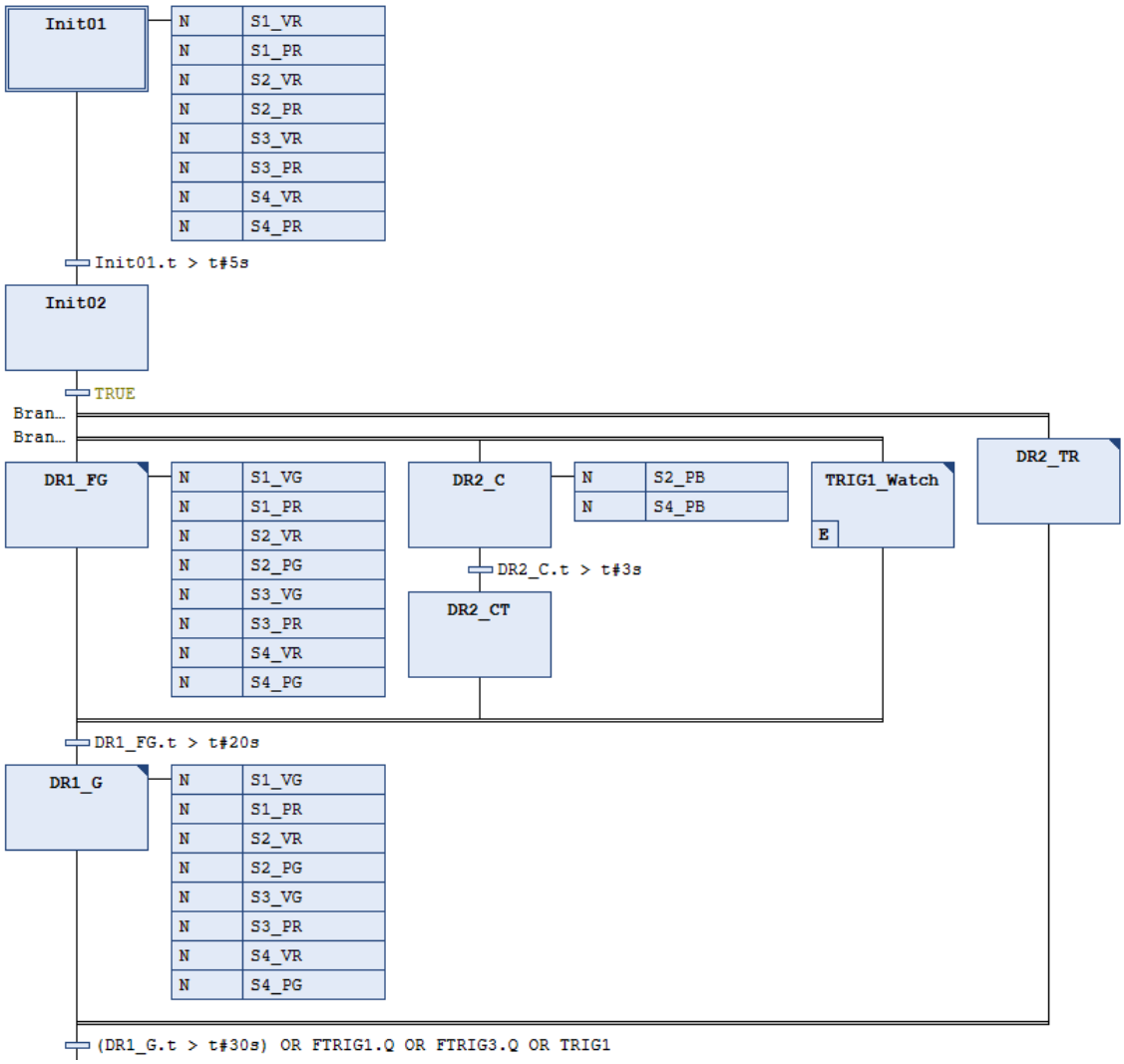


Figura 7.1 - SFC del programa principal (parte 1).

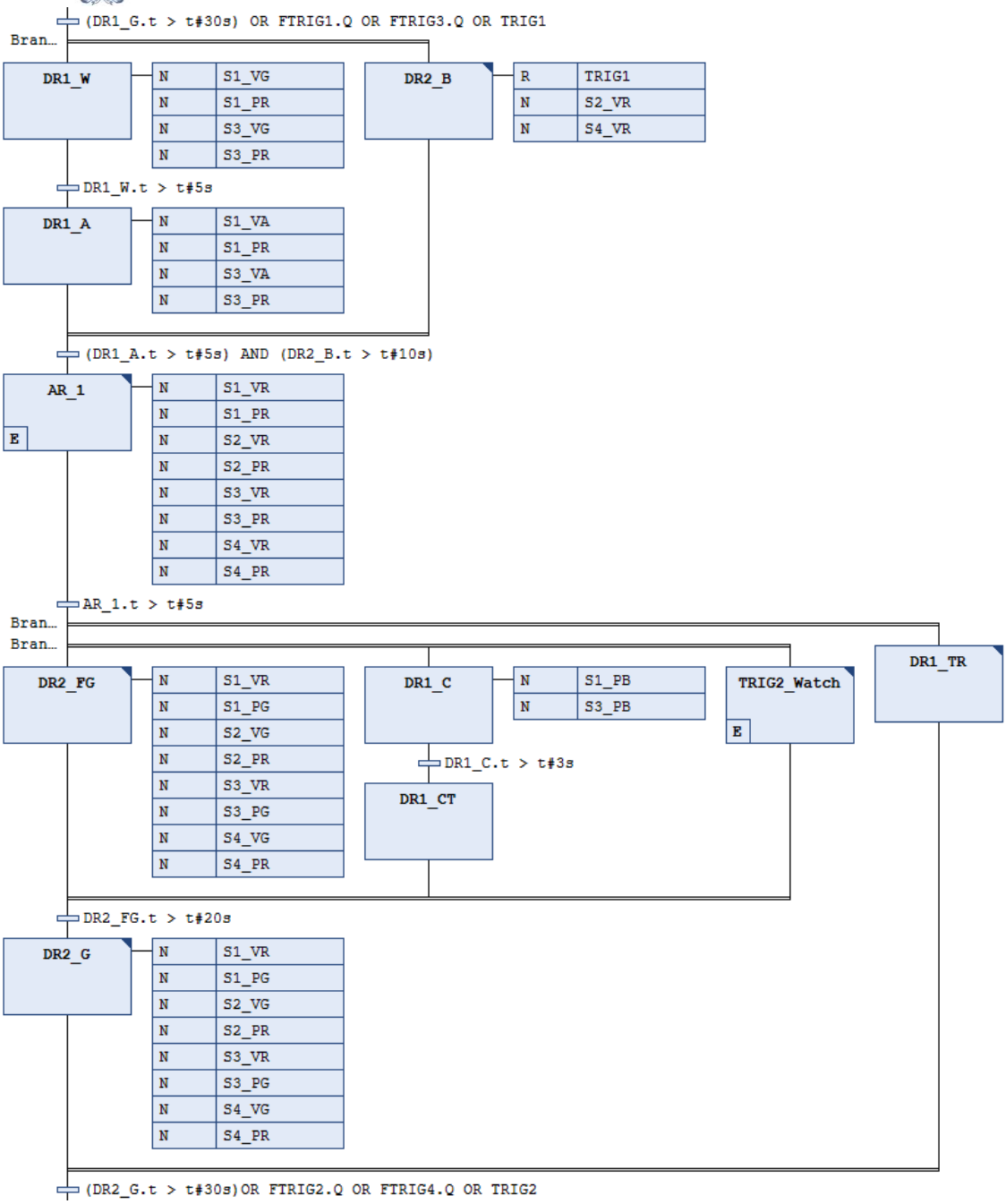


Figura 7.2 - SFC del programa principal (parte 2).

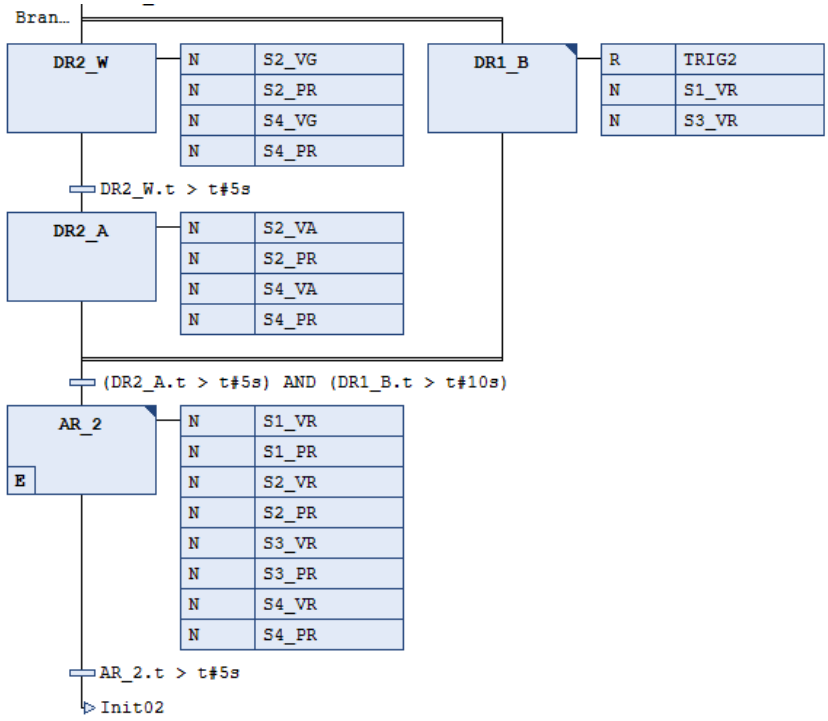


Figura 7.3 - SFC del programa principal (parte 3).

Los estados DR1_FG y DR1_G llaman a la función DR1_G_active:

DR1_G_active

```
1  FTRIG1(CLK := S1_PPB);
2  FTRIG3(CLK := S3_PPB);
```

Los estados DR2_FG y DR2_G llaman a la función DR2_G_active:

DR2_G_active

```
1  FTRIG2(CLK := S2_PPB);
2  FTRIG4(CLK := S4_PPB);
```

El estado DR1_TR llama a la función DR1_TR_active (ver figura 7.4):

DR1_TR_active

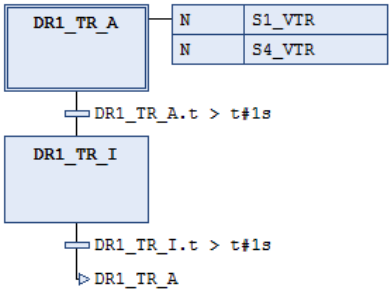


Figura 7.4 - SFC de la función AR_2_active.

El estado DR2_TR llama a la función DR2_TR_active (ver figura 7.5):

DR2_TR_active

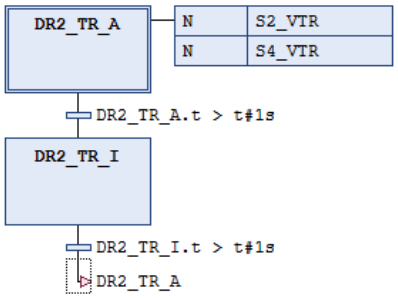


Figura 7.5 - SFC de la función AR_2_active.

El estado DR1_B llama a la función _aDR1_B_active (ver figura 7.6):

_aDR1_B_active

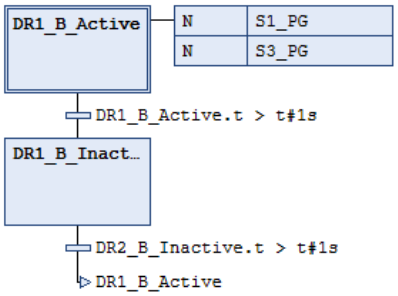


Figura 7.6 - SFC de la función aDR1_B_active.

El estado DR2_B llama a la función _aDR2_B_active (ver figura 7.7):

_aDR2_B_active

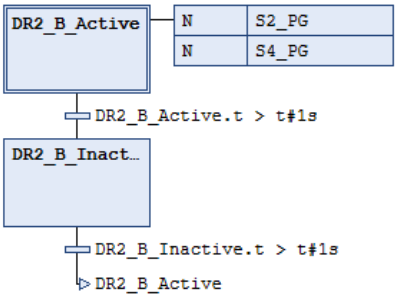


Figura 7.7 - SFC de la función aDR2_B_active.

El estado AR_1 llama a la función AR_1_active (ver figura 7.8):

AR_1_active

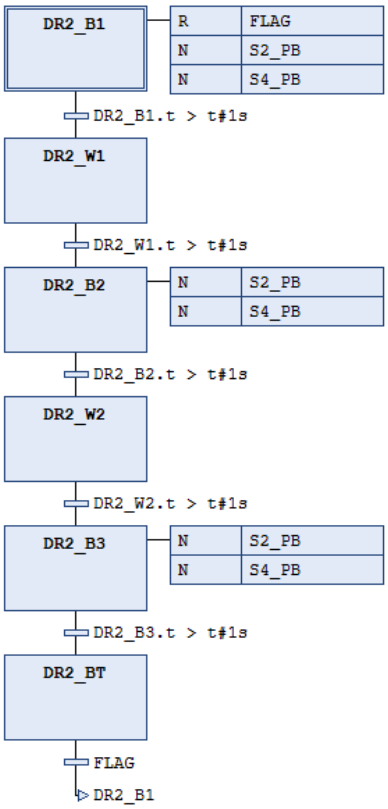


Figura 7.8 - SFC de la función AR_1_active.

El estado TRIG1_Watch llama a la función TRIG1_Watch_active (ver figura 7.9).

TRIG1_Watch_active

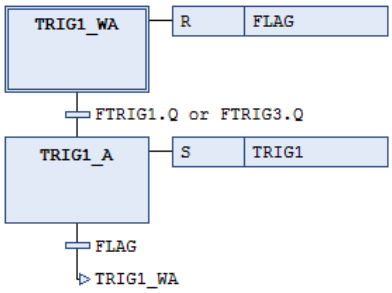


Figura 7.9 - SFC de la función TRIG1_Watch_active.

El estado AR_2 llama a la función AR_2_active (ver figura 7.10):

AR_2_active

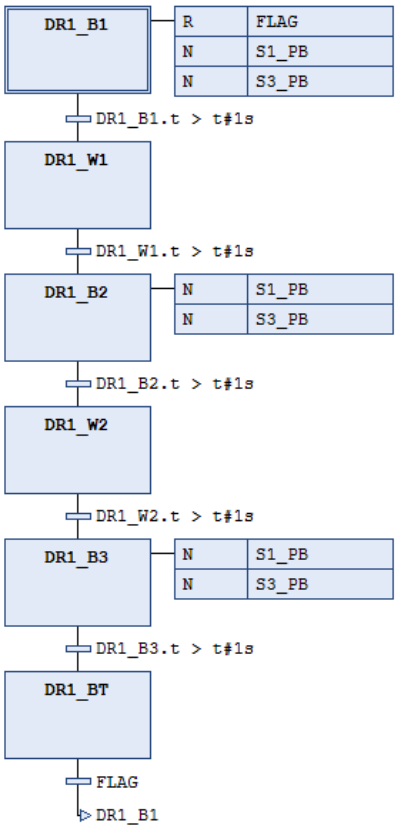


Figura 7.10 - SFC de la función AR_2_active.

El estado TRIG2_Watch llama a la función TRIG2_Watch_active (ver figura 7.11).

TRIG2_Watch_active

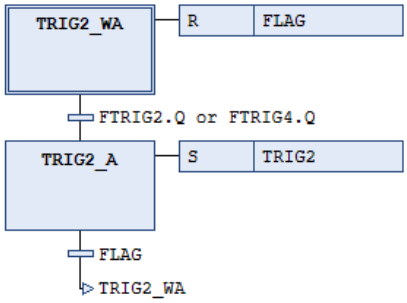


Figura 7.11 - SFC de la función TRIG2_Watch_active.

Al entrar en los estados AR_1, AR_2, TRIG1_Watch y TRIG2_Watch se llama a la función set_FLAG (ver figura 7.12).

set_FLAG

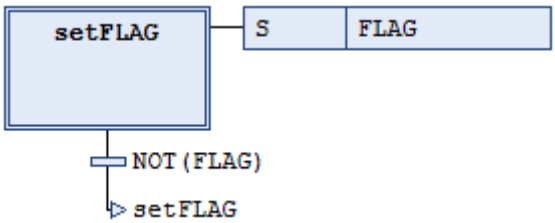


Figura 7.12 - SFC de la función set_FLAG.

8

Planos

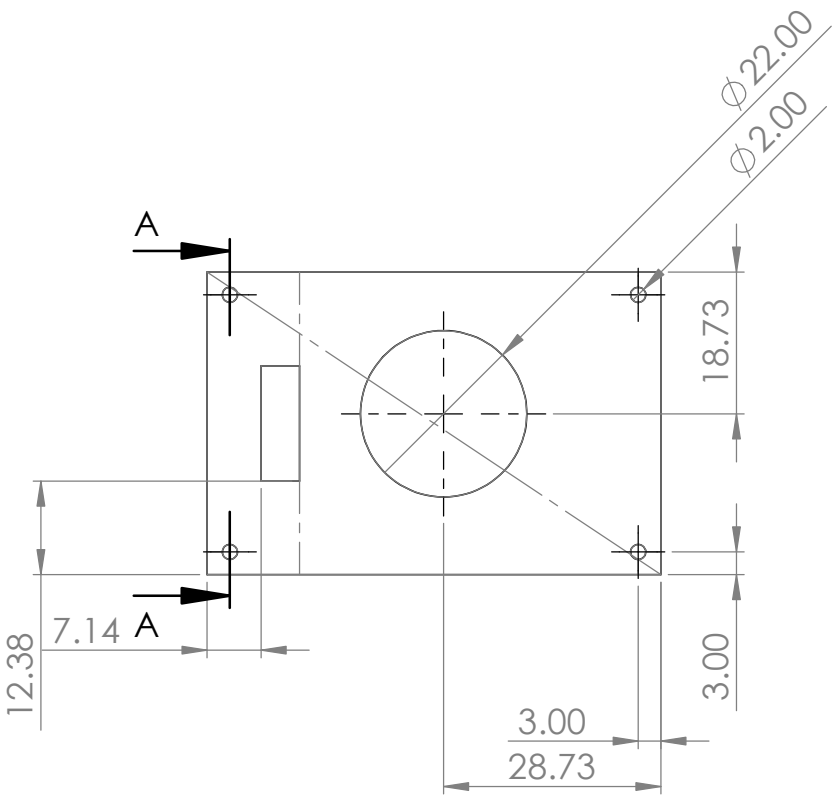
En este capítulo se encuentran los planos de todos los modelos 3D realizados Incluyendo:

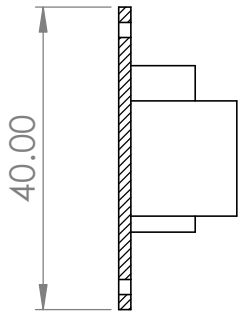
Planos auxiliares:

- Perfboard

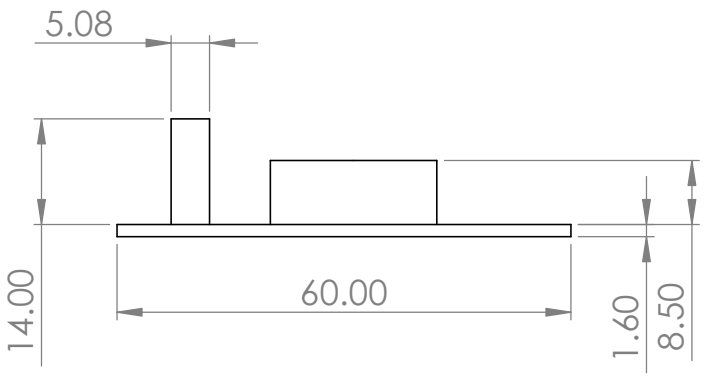
Planos de piezas para Impresión 3D:

- Placa de LEDs Coches
- Placa de LEDs Peatones
- Cuerpo Principal Semáforo
- Base de Semáforo
- Placa Trasera Semáforo



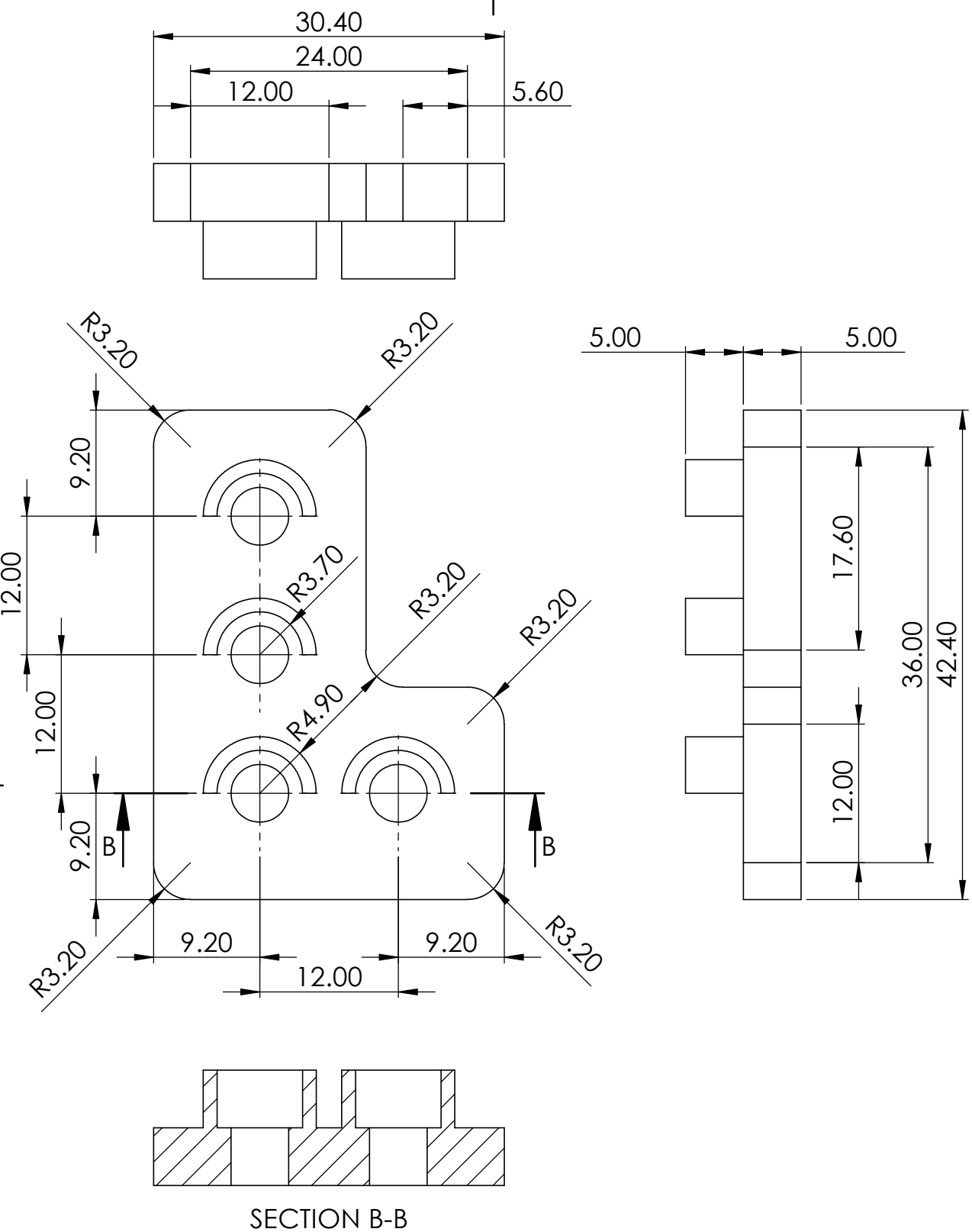


SECTION A-A



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN	Daniel Silva		14/11/24			Perfboard	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. Perfboard	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1

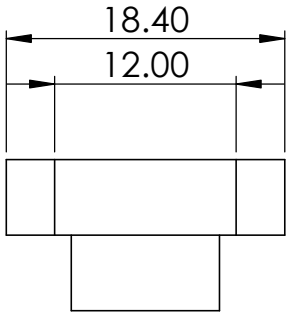
SOLIDWORKS Educational Product. For Instructional Use Only.

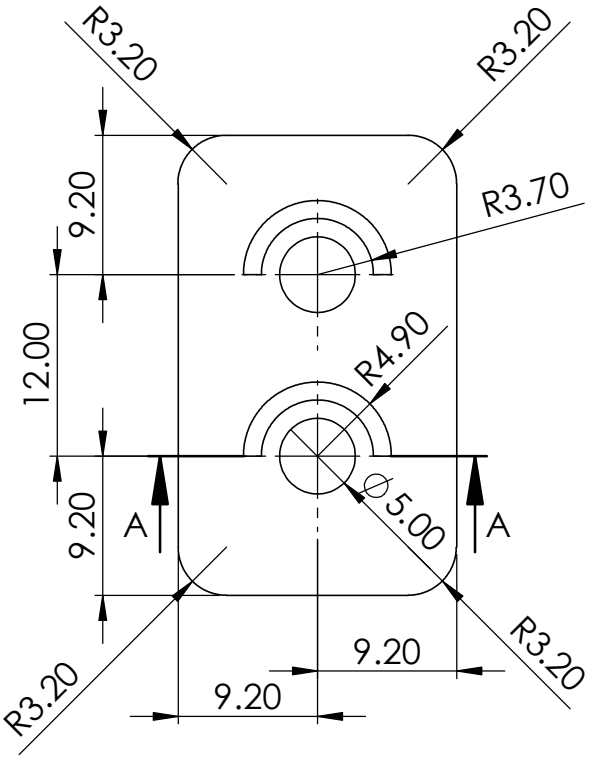


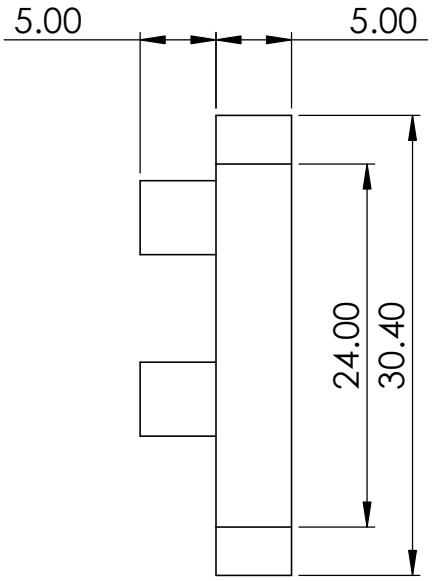
SECTION B-B

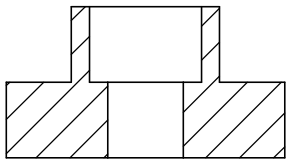
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:		
DRAWN	Daniel Silva		13/11/24	Placa de LEDs Coches		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. Front Leds Plate	A4
				WEIGHT:	SCALE:2:1	SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.





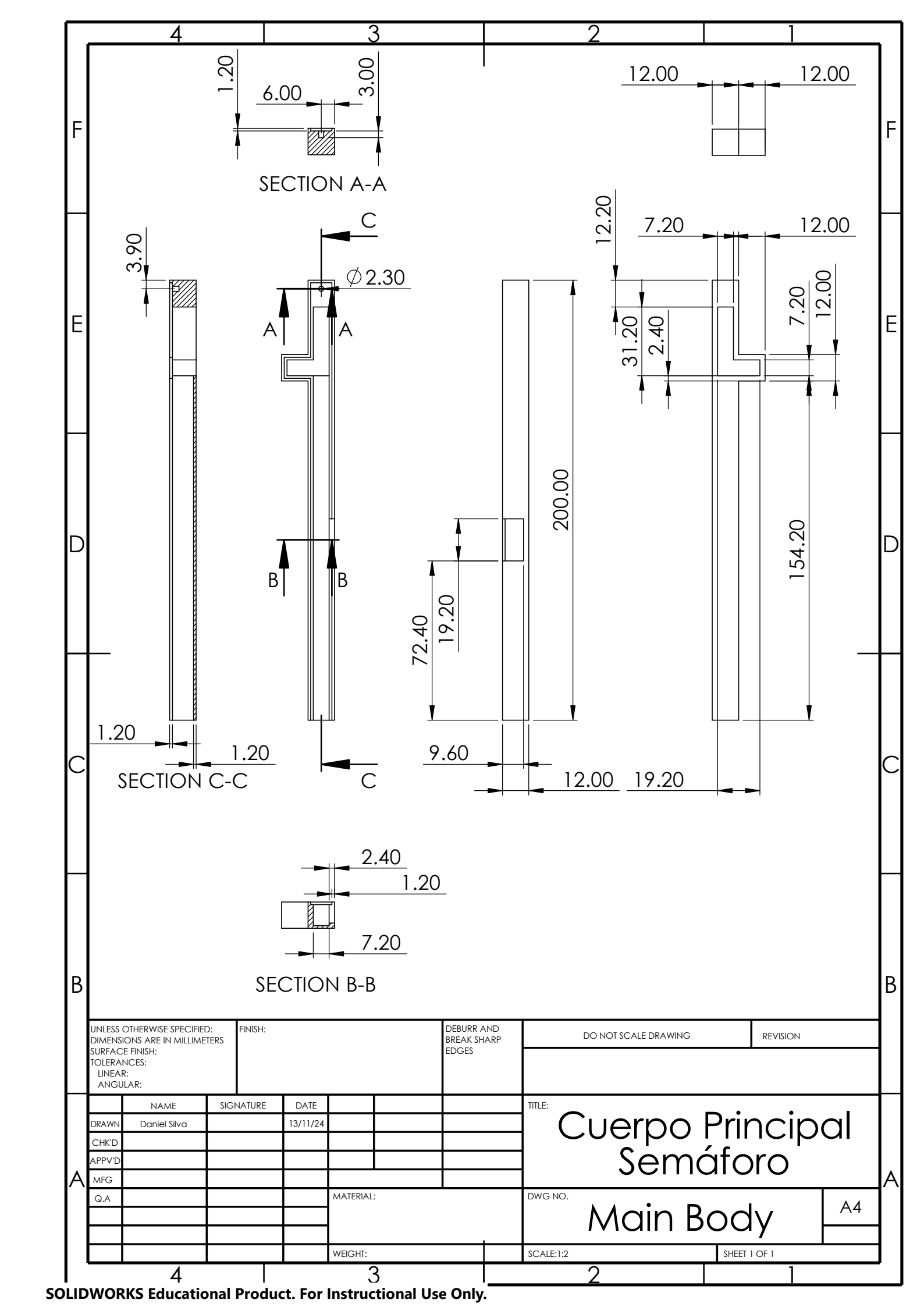




SECTION A-A

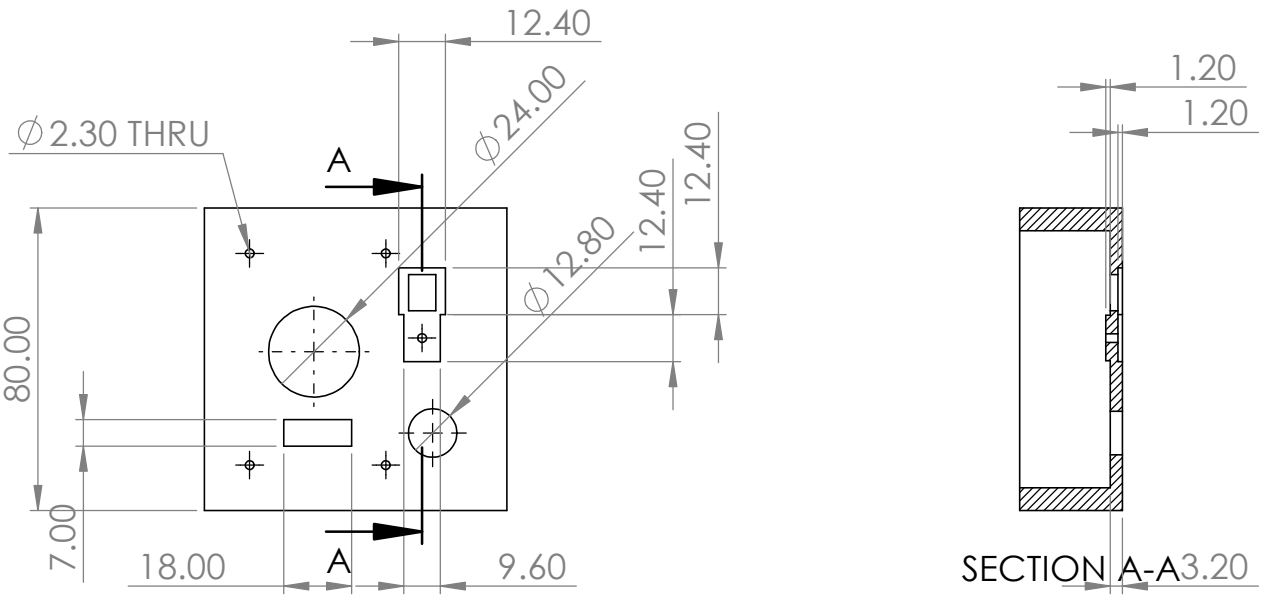
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE		TITLE: Placa de LEDs Peatones	
DRAWN	Daniel Silva		13/11/24			
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. Pedestrian Leds Plate	A4
				WEIGHT:	SCALE:2:1	SHEET 1 OF 1

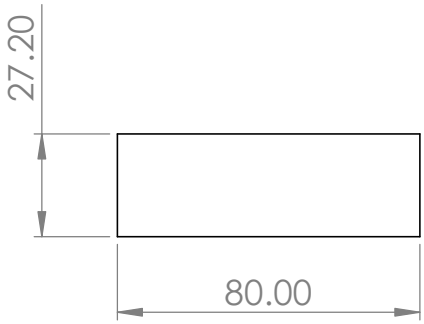
SOLIDWORKS Educational Product. For Instructional Use Only.

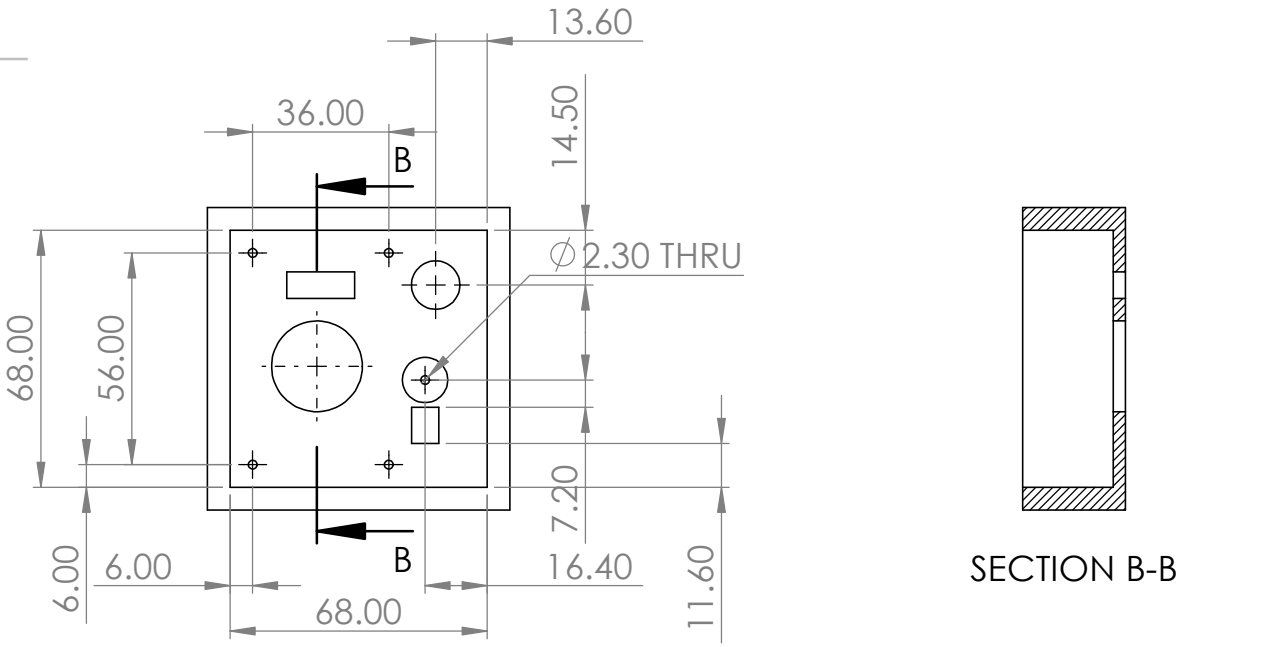


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:		
DRAWN	Daniel Silva		13/11/24	Cuerpo Principal Semáforo		
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. Main Body	A4
				WEIGHT:	SCALE:1:2	SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.





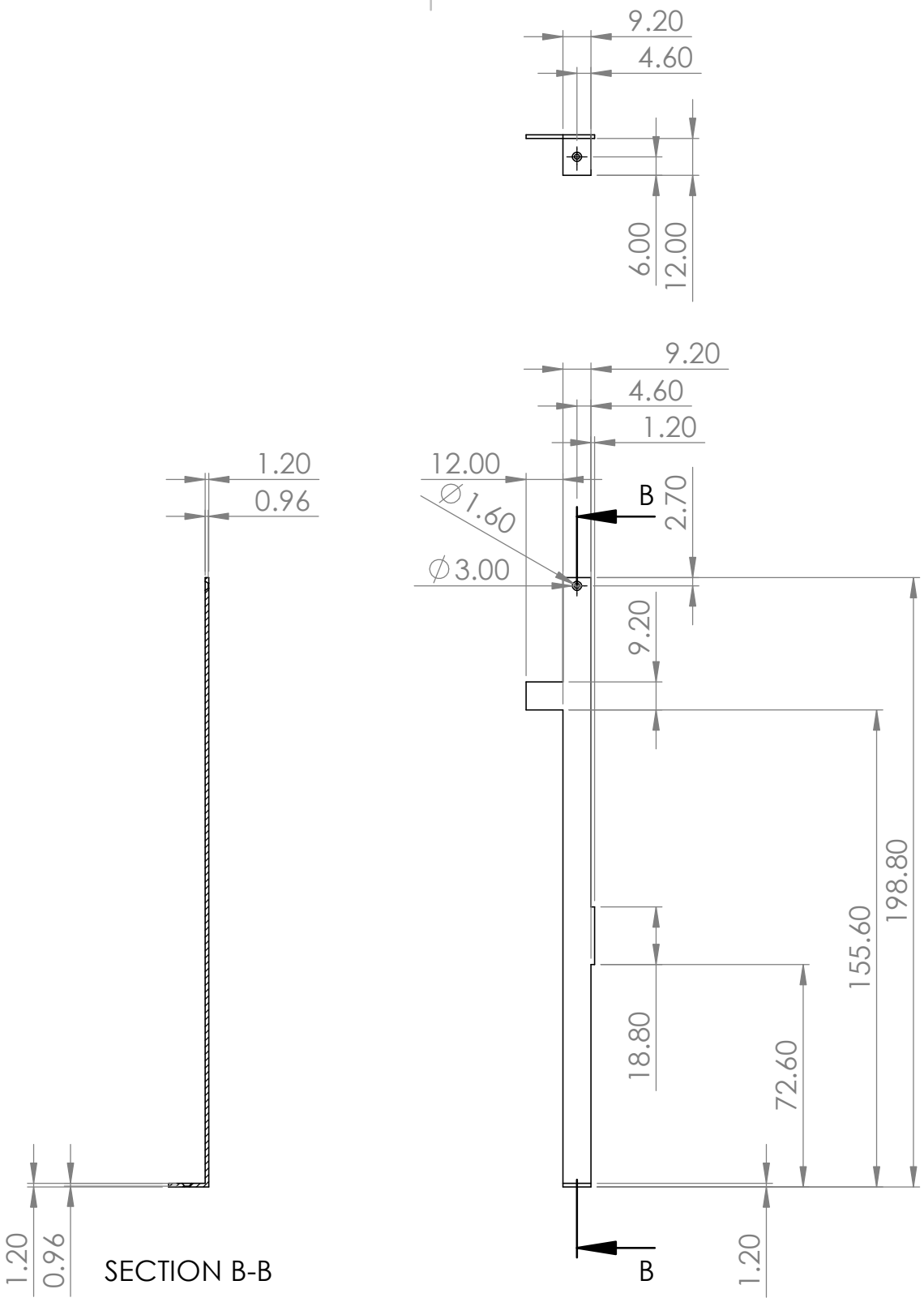


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	

	NAME	SIGNATURE	DATE			
DRAWN	Daniel Silva		14/11/24			
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO.	A4
					Base Plate	
				WEIGHT:	SCALE:1:2	SHEET 1 OF 1

TITLE: Base de Semáforo

SOLIDWORKS Educational Product. For Instructional Use Only.



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN	Daniel Silva		15/11/24			Placa Trasera Semáforo	
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO.	A4
						Back Plate	
				WEIGHT:		SCALE:1:2	SHEET 1 OF 1

SOLIDWORKS Educational Product. For Instructional Use Only.