

Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Electrónica y
Automática Industrial

Trabajo Fin de Grado

Interfaz y control de generación de señales para el
desarrollo de prácticas remotas

ESCUELA POLITECNICA
SUPERIOR

Autor: Rubén Gil Vera

Tutor: José Luis Lázaro Galilea

Cotutor: César Mátaix Gómez

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

**Grado en Ingeniería en Electrónica y Automática
Industrial**

Trabajo Fin de Grado

**Interfaz y control de generación de señales para el
desarrollo de prácticas remotas**

Autor: Rubén Gil Vera

Tutor: José Luis Lázaro Galilea

Cotutor: César Mátaix Gómez

Tribunal:

Presidente: Juan Manuel Miguel Jiménez

Vocal 1º: Pedro Alfonso Revenga de Toro

Vocal 2º: José Luis Lázaro Galilea

Fecha de depósito: 1 de enero de 2021

Agradecimientos

Stay hungry, stay foolish.

Steve Jobs.

A mis padres y hermanos por el apoyo incondicional durante este período de formación.

A mis compañeros y amigos que he conocido en el grado, Alonso, Jaime, Sandra y Rubén, y que tanto hemos compartido en estos cuatro años.

A mi tutor y compañeros del departamento por el apoyo técnico y las oportunidades que me han brindado para seguir formándome académicamente y como persona.

Mención especial a mis amigos de siempre, Manuel, Álvaro, Pablo, Mario y Lucas, por estar siempre ahí cuando más falta hace. A Álvaro por todo el apoyo, paciencia y comprensión ante cualquier situación.

Índice general

Índice general	vii
Índice de figuras	xi
Índice de tablas	xv
Lista de acrónimos	xv
Resumen	xvii
Abstract	xix
1 Introducción y objetivos	1
1.1 Introducción	1
1.2 Experiencias previas	2
1.3 Objetivos	3
2 Estudio teórico	5
2.1 Tarjeta de desarrollo LPC1768	5
2.1.1 DAC (Digital to Analog Converter)	6
2.1.2 Timers	6
2.1.3 Señales PWM	7
2.1.4 Puerto Ethernet	8
2.2 Protocolos de comunicación	9
2.2.1 Protocolo TCP/IP- Conceptos Clave	9
2.2.2 Protocolo HTTP	10
2.2.2.1 Peticiones HTTP	10

2.2.2.2	Respuestas HTTP	10
2.2.2.3	Métodos de petición	11
3	Desarrollo	13
3.1	Descripción general	13
3.2	Diagrama de bloques	14
3.3	Análisis de manejo web con microcontrolador	15
3.3.1	HTTP Server	17
3.3.2	Configuración de la capa de internet: Net_Config.c	18
3.3.3	Configuración de la capa de aplicación: HTTP_CGI.c y elementos HTML	20
3.3.3.1	HTTP_CGI.C	21
3.3.3.2	Páginas web dinámicas	22
3.3.3.3	Entrada de datos: uso de formularios	22
3.3.3.4	Agregar HTML como código C	26
3.4	Diseño front-end y back-end de la aplicación web	28
3.4.1	Front-end: desarrollo de la interfaz de usuario	29
3.4.1.1	Página principal: index.htm	29
3.4.1.2	Encabezados: pg_header1.inc y pg_header2.inc	31
3.4.1.3	Formato de las páginas index.cgi e index1.cgi	32
3.4.1.4	Diseño de la interfaz de la página de señales digitales: index.cgi	34
3.4.1.5	Diseño de la interfaz de la página de señales analógicas y PWM: index1.cgi	40
3.4.2	Back-end: Respuesta del servidor web	45
3.4.2.1	Funciones de configuración	46
3.4.2.2	Gestión de los datos introducidos por cuadros de texto	50
3.4.2.3	Gestión de los datos introducidos por checkboxes	51
3.4.2.4	Gestión de los datos introducidos por botones	53
3.4.2.5	Funciones de interrupción	58

4 Pruebas y resultados	67
4.1 Plataforma de prácticas	67
4.1.1 Protoboard	68
4.1.1.1 Sensor de temperatura	68
4.1.1.2 Display siete segmentos y LEDs de colores	69
4.1.2 Multiplexor analógico	69
4.1.3 Motor DC y puente en H	70
4.1.4 Servomotor	71
4.1.5 Medidor de distancia SRF04	72
4.1.6 Osciloscopio	73
4.2 Pruebas prácticas	73
4.2.1 Control y generación de una señal digital mediante puertos E/S	74
4.2.1.1 Verificación en tarjeta de prácticas de la plataforma	75
4.2.2 Control y generación de una señal digital mediante Timers	76
4.2.2.1 Verificación en tarjeta de prácticas de la plataforma	78
4.2.3 Manejo del convertor digital-analógico	79
4.2.3.1 Verificación en tarjeta de prácticas de la plataforma	82
4.2.4 Diseño de un sistema para un sonar ultrasónico	84
4.2.4.1 Verificación en tarjeta de prácticas de la plataforma	85
5 Conclusiones y líneas futuras	91
5.1 Conclusiones	91
5.2 Líneas futuras	92
6 Presupuesto	95
Bibliografía	97
Anexos	99
Apéndice A Manual de usuario	103

Índice de figuras

1.1	Equipamiento para enseñanza remota de sistemas de control. Universidad EPFL.	2
1.2	Herramientas para enseñanza remota de sistemas de robótica. Instituto Tecnológico de Karlsruhe.	2
2.1	Micronrolador LPC1768.	6
2.2	Descripción de la funcionalidad de los pines de los Timers. (Figura extraída de [1].)	7
2.3	Ejemplos de señal PWM. (Figura extraída de [1].)	8
2.4	Interacción entre capas del protocolo TCP/IP.	9
3.1	Diagrama de bloques del proyecto.	14
3.2	Componentes de la librería RL-ARM.	16
3.3	Estándares de servidor de la librería TCPNet.	16
3.4	Ficheros de HTTP-Server.	17
3.5	Obtención de la IP y la máscara de subred del servidor.	18
3.6	Funcionamiento protocolo CGI.	20
3.7	Interfaz para checkbox.	23
3.8	Interfaz para cuadro de texto.	25
3.9	Agregar archivos HTML al proyecto.	27
3.10	Configuración del archivo Web.inp.	27
3.11	Diagrama de la conversión de elementos HTML a código C.	28
3.12	Interfaz de la página principal de la aplicación web.	29
3.13	Encabezados de las páginas index.cgi e index1.cgi.	31
3.14	Icono página principal.	31

3.15	Iconos de navegación entre páginas.	32
3.16	Interfaz de la página para la generación de señales digitales.	34
3.17	Activación de las secuencias de las señales a y b.	37
3.18	Interfaz para el cambio de nivel de la señal "a".	40
3.19	Activación de la señal "a".	40
3.20	Interfaz de la página para la generación de señales analógicas y PWM. . .	41
3.21	Interfaz de configuración de las señales analógicas.	42
3.22	Interfaz de configuración de las señales PWM.	42
3.23	Interfaz de activación de las señales analógicas	43
3.24	Interfaz de activación de las señales PWM.	44
3.25	Interfaz de selección de los canales del multiplexor.	44
3.26	Interfaz de manejo del osciloscopio.	45
4.1	Plataforma de prácticas.	68
4.2	Conexión del sensor de temperatura BMP180.	69
4.3	Motor DC junto al puente en H.	71
4.6	Osciloscopio DS0138mini.	73
4.7	Terminales empleados para el programa desarrollado.	74
4.8	Simulación para un ciclo de trabajo del 40%.	75
4.9	Pruebas realizadas.	76
4.10	Terminales empleados en el programa.	77
4.11	Esquema temporal del programa.	77
4.12	Simulación para un ciclo de trabajo del 20%.	78
4.13	Pruebas realizadas.	79
4.14	Terminales empleados en el programa.	80
4.15	Simulación cuando los terminales P2.3...P2.0 = "0000"	81
4.16	Simulación cuando los terminales P2.3...P2.0 = "0001"	81
4.17	Simulación cuando los terminales P2.3...P2.0 = "1111"	82
4.18	Señal sinusoidal con frecuencia de 100 Hz (i...f = "0000").	82
4.19	Señal sinusoidal con frecuencia de 1 kHz (i...f = "0001").	83
4.20	Señal sinusoidal con frecuencia de 15 kHz (i...f = "1111").	83

4.21 Diagrama de bloques del s3nar ultras3nico.	84
4.22 Se3al sinusoidal generada al inicio del programa.	85
4.23 Servomotor posicionado a 3cm del obst3culo.	85
4.24 Medida de distancia para un obst3culo a 3 cm.	86
4.25 Se3al sinusoidal generada con una distancia al obst3culo de 3 cm.	86
4.26 Servomotor posicionado a 7,4cm del obst3culo.	87
4.27 Medida de distancia para un obst3culo a 7,4cm.	87
4.28 Se3al sinusoidal generada con una distancia al obst3culo de 7,4 cm.	88
4.29 Servomotor posicionado a una distancia superior a 15cm del obst3culo.	88
4.30 Medida de distancia para un obst3culo a 15 cm.	89
4.31 Se3al sinusoidal generada con una distancia al obst3culo superior a 15 cm.	89

Índice de tablas

3.1	Comandos del lenguaje CGI.	22
3.2	Atributos de la etiqueta <forms>.	33
3.3	Atributos de la etiqueta <input>para generar un cuadro de texto.	36
3.4	Atributos de la etiqueta <input>para generar checkboxes y botones.	37
3.5	Botones para las secuencias.	38
4.1	Tabla de verdad del multiplexor analógico.	70
4.2	Conexionado de los canales a la tarjeta de prácticas.	70
4.3	Código digital y ciclo de trabajo de la señal emitida en P3.25.	74
4.4	Código digital y ciclo de trabajo de la señal emitida en P1.29.	77
6.1	Conexionado de señales con terminales de entrada de la tarjeta de prácticas.	99
6.2	Conexionado de elementos con terminales de la tarjeta de prácticas.	100
6.3	Conexionado de los canales con la tarjeta de prácticas.	101

Resumen

El proyecto consiste en el diseño de una plataforma de prácticas donde se permita a los usuarios generar y controlar señales remotamente, tanto digitales, PWM y analógicas. Se va a desarrollar una aplicación web para la tarjeta de desarrollo mini-DK2 basada en microcontrolador, que constará de una interfaz web programada en HTML y un servidor web que será implementado en lenguaje C basándose en las herramientas aportadas por ARM en Keil uVision 5. La funcionalidad principal del proyecto es el manejo de distintas señales que trabajarán de manera simultánea que servirán de entradas a sistemas de prácticas y que, a partir de una interfaz simplificada, los estudiantes puedan realizar pruebas de forma remota disponiendo de realimentación visual y de información en su ordenador de sus prácticas de laboratorio, modificando a distancia las señales de entrada.

Palabras clave: Control de señales, laboratorio remoto, interfaz web, acceso remoto.

Abstract

The project consists on the design of a platform for practices where users generate and control signals remotely, both digital, PWMs and analog. It's going to be developed a web application for the mini-DK2 development board based on microcontroller, which will consist of a web interface programmed in HTML and a web server that will be implemented in C language based on the tools provided by ARM in Keil uVision 5. The main functionality of the project is the management of different signals that will work in simultaneously as inputs of the practices platform and through a simplified interface students would be able to remotely perform tests of their laboratory practices.

Keywords: Signals control, remote laboratory, web interface, remote Access.

Capítulo 1

Introducción y objetivos

1.1 Introducción

El promover una docencia universitaria de calidad está entre las prioridades de los responsables educativos, desde organismos transnacionales (Objetivo 4^o de la Agenda 2030) hasta locales.

Además, la situación de excepcionalidad sanitaria vino a acelerar procesos de enseñanza-aprendizaje que, a un ritmo más lento, se venían produciendo, teniendo que volcar los esfuerzos en el desarrollo de enseñanza no presencial e híbrida. En este sentido, el desarrollo de clases teóricas está, más o menos, resuelto, pero el desarrollo de docencia práctica en ocasiones se ve lastrado por la necesidad de equipos físicos de los que los estudiantes no pueden disponer en sus domicilios. En esta línea, y con el afán de poder mejorar la adquisición de las competencias prácticas sin necesidad de estar físicamente en los laboratorios, es una buena opción poder acceder remotamente a los equipos físicos y generar señales de entrada, recoger señales de salida y visualizar elementos luminosos de imagen o mecánicos que se estén manejando en dichas prácticas. Eso equivaldría a disponer de los mismos recursos instrumentales necesarios, pero, aumentando la disponibilidad de la instrumentación y equipos de laboratorio de forma remota.

Con la utilización de dicha infraestructura, además de poder entrar remotamente a desarrollar prácticas, serviría de apoyo complementario para que los estudiantes dispusiesen de mayor número de horas de laboratorio, e incluso para que estudiantes cuya distancia a las infraestructuras fuese incluso entre continentes pudieran realizar prácticas remotamente, trabajando con equipos reales y no simulados.

Se propone el desarrollo de herramientas y materiales para realizar prácticas y también poder impartir docencia en remoto y prácticas colaborativas entre profesores y alumnos de diferentes centros o alumnos en sus domicilios y en el centro. En este caso, inicialmente se centrará el trabajo Sistemas Electrónicos Digitales.

1.2 Experiencias previas

Si analizamos los posibles antecedentes en esta aplicación, en el ámbito europeo existen universidades con laboratorios remotos aplicados a diferentes disciplinas. Un ejemplo es el de la universidad suiza EPFL, donde se dispone de estaciones para experimentos online, con servo drivers, sistemas de control de temperatura y un sistema para el control de un péndulo invertido en materias de control.

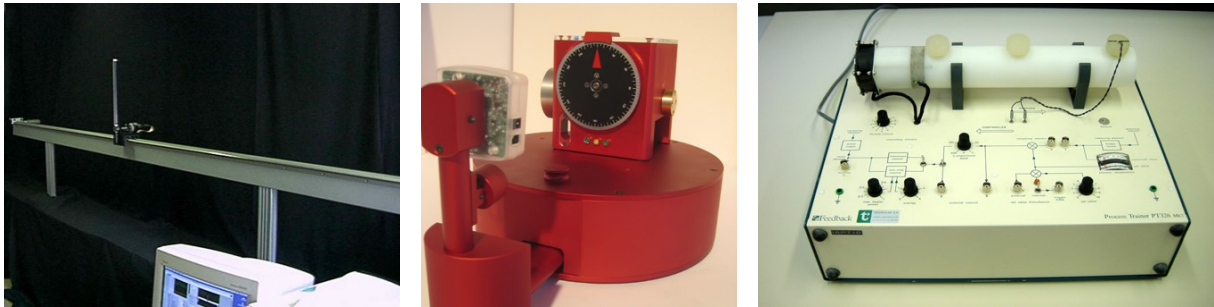


Figura 1.1: Equipamiento para enseñanza remota de sistemas de control. Universidad EPFL.

También en el Instituto Tecnológico de Karlsruhe disponen de un laboratorio de robótica accesible a distancia que permite a los estudiantes acceder a los robots del laboratorio de forma remota y ejecutar sus propios proyectos en los robots. Para los estudiantes, poder manejar el laboratorio más horas es una forma única de aprender los fundamentos y tener más experiencia con el hardware.



Figura 1.2: Herramientas para enseñanza remota de sistemas de robótica. Instituto Tecnológico de Karlsruhe.

1.3 Objetivos

El objetivo general de este proyecto es permitir el acceso a realizar prácticas remotas en maquetas en los laboratorios del Departamento de Electrónica, donde los alumnos tengan la posibilidad de generar y controlar distintos tipos de señales a través de una interfaz web con el fin de probar la respuesta de sus prácticas de laboratorio.

El campo de aplicación es el diseño de laboratorios remotos que contribuyan a una mejora de la calidad de la docencia y de la experiencia educativa de los estudiantes. Se propone reorientar tecnologías, recursos y herramientas tradicionalmente aplicadas de forma presencial para que el estudiante pueda desarrollar las mismas competencias de forma remota, personalizando y adaptando su formación a las condiciones espacio-temporales disponibles.

Entre los objetivos específicos se destacan los siguientes:

- Acceder y controlar remotamente las aplicaciones a desarrollar en este proyecto, instaladas en el ordenador que controla la práctica en el laboratorio (basada en una maqueta).
- Generar remotamente señales digitales (niveles o secuencias) en las entradas de los sistemas de las prácticas.
- Poder generar remotamente señales de frecuencia periódica (relojes) en pines de la tarjeta utilizada para la realización de la práctica.
- Generar remotamente señales analógicas en pines de la tarjeta utilizada para la realización de la práctica.
- Controlar las distintas señales a través de un multiplexor analógico empleando la interfaz web a desarrollar.
- Poder disponer de una realimentación visual, mediante cámaras, de la realización de la práctica.

Capítulo 2

Estudio teórico

2.1 Tarjeta de desarrollo LPC1768

Para el desarrollo del proyecto se ha empleado la tarjeta de desarrollo mini-DK2 cuyo microcontrolador es el LPC1768, con procesador Cortex-M3. Este es un procesador de 32 bits diseñado por ARM y basado en la arquitectura ARMv7, perteneciendo al perfil M de los procesadores con esta arquitectura. Dicho perfil es idóneo para aplicaciones de bajo costo, en donde la eficiencia de procesado es importante, con baja latencia de interrupción y facilidad de uso, como es el caso de nuestra aplicación.

La arquitectura del procesador es Harvard. Además, dispone de un reloj de 100MHz de frecuencia de CPU, que para la aplicación deseada es más que suficiente, siendo los accesos a memoria en un único ciclo.

Las principales razones por las que se ha decidido utilizar esta tarjeta son su facilidad para programarla y la disposición de los siguientes periféricos, que resultan indispensables para aplicaciones prácticas.

- Puerto Ethernet MAC.
- USB interface.
- ADC de 12 bits, con ocho canales de entrada.
- DAC de 10 bits.
- Cuatro temporizadores de propósito general.
- Seis salidas de PWM.
- Setenta pines de entrada/salida de propósito general.

Además, dispone de un botón de reset y un puerto JTAG para descarga del programa en placa. En la figura 2.1, se muestra la tarjeta de desarrollo LPC1768 empleada en el proyecto.

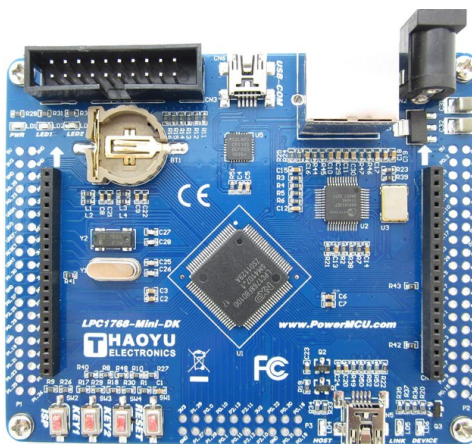


Figura 2.1: Microntrolador LPC1768.

A continuación, se va a realizar una descripción de los periféricos y elementos funcionales de la tarjeta empleados en este proyecto.

2.1.1 DAC (Digital to Analog Converter)

El conversor digital analógico de este microcontrolador dispone de una salida analógica situada en la patilla 26 del P0. Las características más destacables del DAC son:

- Máxima frecuencia de conversión de 1MHz.
- Conversor de 10 bits.
- Salida en buffer.
- Posibilidad de selección entre mayor velocidad o menor consumo.

2.1.2 Timers

Como se ha comentado en el apartado 2.1, se disponen de cuatro temporizadores de propósito general, los cuales resultan indispensables en nuestra aplicación, puesto que estos han sido usados en la emisión de las secuencias de las señales digitales (como disparador de evento periódico) y en la conversión del DAC.

Los cuatro timers tienen un contador de 32-bit con un prescaler programable de 32-bit también. Además, disponen de dos métodos de funcionamiento, en modo CAPTURE o MATCH; este último ha sido el empleado en el proyecto.

Cuando el modo de funcionamiento es CAPTURE, cada vez que el contador alcanza el valor del prescaler se procede a capturar el valor del pin del Timer empleado, siendo posible además generar una interrupción y resetear el contador del temporizador para realizar nuevas capturas.

Por otro lado, si se desea trabajar en modo MATCH, cada vez que el contador del programa alcance el valor del registro de match (registro MR3:0), algunas salidas pueden realizar un toggle, paso a nivel alto, a nivel bajo o no hacer nada. También, se tiene la posibilidad de generar una rutina de interrupción donde se implemente la funcionalidad específica requerida por la aplicación cada vez que se produce dicho evento, que es el caso que abordamos en el proyecto.

A continuación, se muestra la figura 2.2 obtenida del manual de usuario del fabricante [1] donde se ilustra lo descrito.

Table 424. Timer/Counter pin description

Pin	Type	Description
CAP0[1:0] CAP1[1:0] CAP2[1:0] CAP3[1:0]	Input	Capture Signals- A transition on a capture pin can be configured to load one of the Capture Registers with the value in the Timer Counter and optionally generate an interrupt. Capture functionality can be selected from a number of pins. When more than one pin is selected for a Capture input on a single TIMER0/1 channel, the pin with the lowest Port number is used Timer/Counter block can select a capture signal as a clock source instead of the PCLK derived clock. For more details see Section 21.6.3 .
MAT0[1:0] MAT1[1:0] MAT2[3:0] MAT3[1:0]	Output	External Match Output - When a match register (MR3:0) equals the timer counter (TC) this output can either toggle, go low, go high, or do nothing. The External Match Register (EMR) controls the functionality of this output. Match Output functionality can be selected on a number of pins in parallel.

Figura 2.2: Descripción de la funcionalidad de los pines de los Timers. (Figura extraída de [1].)

2.1.3 Señales PWM

La tarjeta de desarrollo dispone de seis salidas de señal PWM y seis registros de match, permitiendo, además, seleccionar el modo de funcionamiento como single edge o double edge.

Para el modo de funcionamiento como single edge son necesarios dos registros match, siendo PWMMR0 el encargado de controlar el período de la señal PWM, reseteando la cuenta en cada match. El otro registro controla el ciclo de trabajo de la señal, introduciéndose su valor en los registros PWMMR1:6.

Cuando se desea trabajar con una señal PWM double edge, son necesarios tres registros. Donde PWMMR0 se encarga de controlar el período de la señal y los otros registros, controlan la posición del flanco de subida y bajada de la señal PWM. Para esta configuración de señal deben emplearse dos registros consecutivos, recomendándose el uso del PWMMR1 junto al PWMMR2, PWMMR3 junto al PWMMR4 y el PWMMR5 junto al

PWMMR6. El primer registro controla cuando sucede el flanco de subida y el segundo el flanco de bajada de la señal PWM, además, la señal PWM se emite en la salida del segundo registro.

En la figura 2.3 extraída del manual de usuario [1], se muestran unos ejemplos del funcionamiento de las señales PWM.

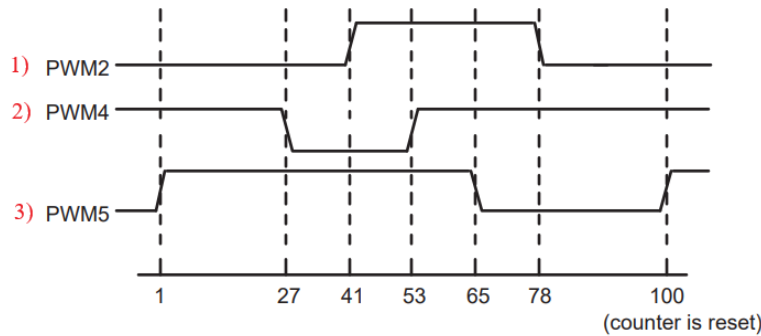


Figura 2.3: Ejemplos de señal PWM. (Figura extraída de [1].)

En el primer ejemplo, se emplea PWMMR1 junto al PWMMR2, como el valor de la cuenta de MR1 es equivalente a un ciclo de trabajo del 41 % y el de MR2 es del 78 %, se produce primero el flanco de subida y al llegar al 78 % de la PWM se da el flanco de bajada.

Para el segundo ejemplo, se emplea PWMMR3 junto al PWMMR4, pero como el ciclo de trabajo del segundo registro es del 27 % que es menor al del primero que es del 53 %, se produce primero el flanco de bajada y posteriormente, el de subida.

En el tercer ejemplo, la señal está a nivel alto desde el principio hasta el 65 %, cuando pasa a nivel bajo, indicado por el MR5 y funcionando como single edged.

2.1.4 Puerto Ethernet

El módulo Ethernet del LPC1768, permite trabajar a 10/100 Mbps y su configuración se gestiona por el archivo EMAC.c proporcionado por Keil. Este pertenece a la librería RL-TCPNet, que es descrita en el apartado 3.3 y debe ser agregada al proyecto para poder hacer uso del módulo Ethernet.

A partir de este puerto se realiza la comunicación entre la aplicación desarrollada situada en la tarjeta mini-DK2 y el ordenador del usuario.

2.2 Protocolos de comunicación

2.2.1 Protocolo TCP/IP- Conceptos Clave

En la actualidad el protocolo estándar de comunicación entre equipos, es TCP/IP. Este es integrado en nuestra aplicación, como se mencionará en el apartado 3.3, a través del componente TCPnet Networking Suite. Sin embargo, en este apartado se va a realizar una descripción genérica del funcionamiento y los aspectos más relevantes de dicho protocolo.

TCP/IP es un conjunto de reglas estandarizadas que permite a los equipos comunicarse en una red. Por un lado, IP es la parte que obtiene la dirección a la que se envían los datos y TCP se encarga de la entrega de los datos una vez se ha establecido dicha dirección IP.

Cuando se envía información, empleando dicho protocolo, esta se descompone en paquetes que una vez llegan al receptor se vuelven a ensamblar reconstruyendo los datos enviados por el emisor. Además, se dividen las distintas tareas de comunicación en capas, teniendo cada una de ellas una funcionalidad concreta. A continuación, se explica cada una de las capas que constituyen el paquete de protocolos:

- **Capa de enlace de datos.** Se encarga de controlar las partes físicas del envío y recepción de datos, realizado mediante el cable Ethernet que comunica el ordenador con la tarjeta de desarrollo en nuestra aplicación.
- **Capa de internet.** Controla la transmisión de los paquetes alrededor de la red utilizando el protocolo IP.
- **Capa de transporte.** Proporciona una conexión de datos fiables entre el receptor y el emisor, dividiendo los datos en paquetes y avisando cuándo se entregan los datos enviados. Para ello se emplea el protocolo TCP.
- **Capa de aplicaciones.** Es el conjunto de aplicaciones que requieren comunicación de red con la que el usuario interactúa. En nuestra aplicación sería la interfaz web desarrollada, implementada mediante el protocolo HTTP.

En la figura 2.4 se muestra cómo se realiza la comunicación entre las capas que conforman el protocolo.



Figura 2.4: Interacción entre capas del protocolo TCP/IP.

2.2.2 Protocolo HTTP

Este protocolo, tal y como se ha mencionado en el apartado anterior (2.2.1), forma parte de una de las capas del protocolo TCP/IP. En concreto se aplica en la capa a desarrollar en nuestra aplicación. A continuación, se va a realizar una breve explicación de su funcionamiento y características.

El protocolo de transferencia de hipertexto (HTTP) es el protocolo de comunicación que permite las transferencias de información a través de archivos (HTML, XHTML...) en internet. Además, está orientado a transacciones y trabaja bajo el sistema de petición-respuesta entre un cliente y servidor.

El cliente o usuario, realiza la petición enviando un mensaje y el servidor web, responde con un mensaje de repuesta devolviendo la información solicitada por el cliente.

Por otro lado, el formato de los mensajes que permite transmitir dicho protocolo es texto plano, codificado en ASCII. La mayor ventaja de este formato es que lo hace más legible y fácil de depurar; sin embargo, los mensajes son más largos. El formato de estos, sea de petición o respuesta, está formado por: **línea de estado, cabeceras y cuerpo**; pero su contenido en cada una de estas partes es distinto según el tipo de solicitud (de petición o respuesta). En los apartados 2.2.2.1 y 2.2.2.2 se realiza un análisis del formato de los mensajes.

2.2.2.1 Peticiones HTTP

Las peticiones HTTP son mensajes enviados por un cliente, que inician una acción en el servidor. Están formados por los siguientes elementos:

- **Línea de inicio.** Se especifica el método de petición (descritos en el 2.2.2.3), el objetivo de dicha petición y la versión de HTTP empleada.
- **Cabeceras.** Aportan información adicional acerca de la petición realizada.
- **Cuerpo.** Se trata de la parte final de una petición y lleva datos asociados con la petición solicitada. Sin embargo, existen peticiones en las que no aparece, en concreto, aquellas que reclaman datos como GET, HEAD, DELETE, OPTIONS.

2.2.2.2 Respuestas HTTP

Las respuestas HTTP son mensajes que realiza el servidor enviando los datos solicitados de vuelta al cliente. Están compuestos por:

- **Línea de inicio.** Contiene información acerca de la versión del protocolo, el código de estado y un texto que lo define. El código de estado es la parte más importante de la respuesta, después de su propio contenido, y se clasifican en cinco clases:
 1. Respuestas informativas. Representadas con los códigos 100 a 199.
 2. Respuestas satisfactorias. Representadas con los códigos 200 a 299.
 3. Redirecciones. Representadas con los códigos 300 a 399.
 4. Errores debidos al cliente. Representadas con los códigos 400 a 499.
 5. Errores debidos al servidor. Representadas con los códigos 500 a 599.
- **Cabeceras.** Aportan información adicional de la respuesta, al igual que en las peticiones.
- **Cuerpo.** No todas las respuestas tienen uno. Aporta información de los datos asociados a la respuesta dada.

2.2.2.3 Métodos de petición

HTTP define un conjunto de métodos de petición para indicar qué se desea realizar con un determinado recurso. Los más importantes se describen a continuación:

Método GET

Se emplea para enviar información que no cambia el contenido del servidor web, como petición de consultas. Además, no permite el envío de datos a excepción si dichos datos se envían como parámetro en la Url que realiza la petición, retornando tanto la cabecera como el contenido. Este método es muy utilizado en HTML cuando se quieren implementar formularios con cuadros de texto.

Como se ha comentado, dado que la información es consultable desde la Url de la página, permite utilizarla, modificarla o, incluso, eliminarla del registro sin necesidad de la interfaz visual.

Un ejemplo del empleo de este método es el mostrado en el código 2.1, donde se piden al usuario los datos de nombre y apellido.

Código 2.1: Ejemplo de método de envío GET.

```
1 <html>
2 <form action="www.GETmethod/index.html" method="get">
3 Nombre: <input type="text" name="nombre"><br>
4 Apellido: <input type="text" name="apellido"><br>
5 <input type="submit" value="Enviar">
6 </form>
7 </html>
```

En la Url del navegador se mostraría, antes de ser introducidos los datos por el usuario, lo siguiente: **www.GETmethod/index.html**; al introducir los datos y enviarlos pulsando el botón “Enviar”, se actualizaría a **www.GETmethod/index.html?nombre=ruben&apellido=gil**.

Método HEAD

Funciona de la misma manera que el método GET pero sin el cuerpo de la respuesta. Es decir, no retorna ningún contenido al servidor. Este método de envío se emplea cuando solo es de interés el código de estado (mostradas en el apartado 2.2.2.2) y el encabezado.

Por tanto, el método HEAD es de gran importancia si lo que quiere obtenerse es información acerca del estado de los enlaces que conforman el servidor web, con el fin de verificar si existen enlaces rotos en este.

Método POST

Este método de envío es uno de los más utilizados en los entornos web. A diferencia del método GET, en este, los datos son enviados a través del cuerpo en vez de hacerlo por la Url. Es decir, cuando se emplea este método, se envía un dato a un recurso específico causando un cambio en el estado o en el servidor. Un ejemplo del empleo de este método es el mostrado en el código 2.2, donde se pide al usuario los datos de nombre y apellido.

Código 2.2: Ejemplo de método de envío POST.

```
1 <form action="www.POSTmethod/index.html" method="post">
2   Nombre: <input type="text" name="nombre"><br>
3   Apellido: <input type="text" name="apellido"><br>
4   <input type="submit" value="Enviar">
5 </form>
```

Cuando el usuario pulsa el botón **Enviar** la Url no se actualiza, manteniéndose como **www.POSTmethod/index.html**. Sin embargo, el cuerpo de la petición contendrá los datos “**nombre**”: “**ruben**” y “**apellido**”: “**gil**” que se asignarán a variables de entorno al llegar al servidor.

Método PUT

Es similar al método POST con la diferencia de que el método PUT reemplaza todas las representaciones del recurso solicitado con la carga de la petición. Es decir, se emplea para la actualización de información ya existente, no asignando el valor a la variable nuevamente.

Capítulo 3

Desarrollo

3.1 Descripción general

En este capítulo se describirá cómo ha sido desarrollada la aplicación web para la plataforma de prácticas de nuestro proyecto. Se comenzará realizando un análisis de las herramientas y librerías necesarias para el desarrollo del proyecto. También se realizará una descripción de los conceptos clave de los protocolos de comunicación empleados en la aplicación. Dichos conceptos han sido extraídos del manual aportado por ARM para el desarrollo de aplicaciones web [2] y de bibliografía online acerca de protocolos como [3] y [4], entre otros. Tras esto, se describirá de forma detallada la implementación de la aplicación web en su totalidad; desde el front-end, que es la interfaz de usuario, hasta su back-end donde se encuentra toda la funcionalidad de la aplicación.

Se comenzará describiendo el diseño front-end en el apartado 3.4.1. Se explicará cómo se han generado los distintos elementos, como botones, checkboxes o cuadros de texto, con los que interactuará el usuario. Finalmente será descrito el back-end de la aplicación, en el apartado 3.4.2, donde se analizarán las funciones y los módulos ligados a la interfaz previamente diseñada, encargados de generar las señales PWM, conversión mediante DAC y temporizadores de emisión de secuencias, así como el cambio de nivel de señales.

Por tanto, la aplicación final generará diversas señales que posteriormente se usarán como entradas de la tarjeta de desarrollo situada en la plataforma de prácticas, donde el usuario realizará pruebas de sus prácticas de laboratorio.

3.2 Diagrama de bloques

El diagrama de bloques de la aplicación a desarrollar es el mostrado en la figura 3.1. El punto de partida es el ordenador remoto del usuario, que se conectará a la maqueta a través de la aplicación de control remoto **AnyDesk** u otra aplicación similar.

La maqueta, a su vez, está formada por el ordenador de laboratorio y la tarjeta de desarrollo descrita en el apartado 2.1, que en su conjunto forman el servidor web.

Finalmente nos encontramos con la tarjeta de prácticas que es el dispositivo donde el usuario realizará el volcado de sus prácticas, excitándola a través de las señales generadas en la aplicación web y observando su respuesta en los elementos de las prácticas.

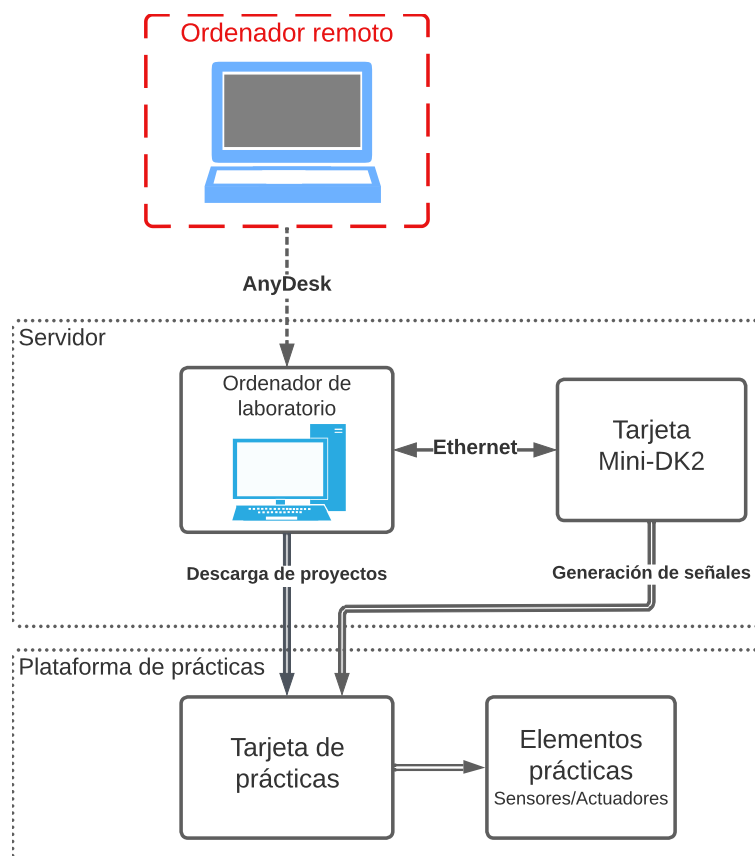


Figura 3.1: Diagrama de bloques del proyecto.

3.3 Análisis de manejo web con microcontrolador

Para el desarrollo de la aplicación web para la plataforma de prácticas del proyecto es necesario conocer una serie de conceptos previos que serán descritos a continuación. Uno de dichos conceptos es el de entorno de desarrollo integrado, que es una herramienta software que proporciona un entorno de programación completo para los desarrolladores de software. Dicho entorno es empleado para escribir, probar y depurar los programas desarrollados cuyo objetivo principal es maximizar la productividad reduciendo la configuración para reconstruir múltiples utilidades de desarrollo.

El entorno de desarrollo para la programación de nuestra aplicación es Keil uVision v5. Este entorno se basa en el concepto de proyecto, donde se incluyen los distintos ficheros (de tipo .s y .c) y las librerías a compilar, que en su conjunto forman la aplicación. Además, el programa se puede descargar directamente en la tarjeta de desarrollo a través de la opción Debug, pudiendo realizar una depuración paso a paso, consulta de variables y registros, entre otros.

Por otro lado, el lenguaje de programación del microcontrolador que será empleado para programar el back-end de nuestra aplicación web es C. Dicho lenguaje se encuentra dotado con una gran cantidad de funciones y librerías, previamente definidas, que facilitan la programación de la tarjeta, además de la gran cantidad de documentación que puede encontrarse en la web del fabricante.

Por último, para implementar la interfaz web se ha utilizado el lenguaje de marcado HTML, debido a la facilidad de integración en la herramienta de programación respecto a otros lenguajes como JavaScript. HTML es un estándar que define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes y los diferentes elementos para comunicarse con el servidor web.

La parte fundamental del proyecto es la comunicación entre el servidor web (formado por el ordenador de laboratorio y la tarjeta mini-DK2) con el ordenador remoto.

Para la creación y despliegue del servidor se ha empleado el paquete de librerías RL (Real-Time) aportado por ARM, el cual es un conjunto de librerías diseñadas con lenguaje de nivel medio para aplicaciones de comunicación en tiempo real en sistemas embebidos basados en la arquitectura ARM. Esto permite que los desarrolladores se centren exclusivamente en implementar su aplicación sin tener que invertir tiempo en desarrollar funciones de bajo nivel.

Dentro de este paquete nos encontramos con cuatro componentes, que son los mostrados en la figura 3.2.

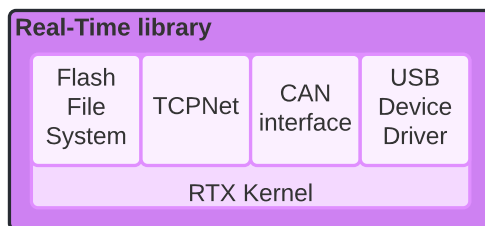


Figura 3.2: Componentes de la librería RL-ARM.

Además, como se puede observar en dicha figura, todos los elementos están diseñados para funcionar junto con el sistema operativo en tiempo real “*Keil RTX*”, aunque su uso es opcional dado que nuestra aplicación puede implementarse sin necesidad de sistema operativo como base.

En concreto, el elemento del que se van a obtener los recursos necesarios para la programación de nuestra aplicación es la suite de redes **TCPNet** sin uso del sistema operativo RTX, siendo el funcionamiento del protocolo en el que se basa dicha librería descrito en el apartado 2.2.1.

TCPNet se encarga de la implementación del protocolo TCP/IP para microcontroladores basados en ARMv7, permitiendo reducir el espacio de memoria usado y el tamaño del código en comparación a si se implementase desde un nivel más bajo, lo cual resulta muy ventajoso en sistemas con recursos limitados como los sistemas empujados. Además, incluye un gran número de estándares para el desarrollo de la capa de aplicación, que son los mostrados en la figura 3.3, pero, en concreto, para el desarrollo de la aplicación web de nuestro proyecto, se empleará la herramienta **HTTP Server** fundamentado en el protocolo HTTP que fue descrito en 2.2.2. Este servicio web nos permite la ejecución de páginas web escritas en HTML en cualquier navegador web para cualquier plataforma, ya sea PC, Mac, teléfono móvil o cualquier dispositivo con posibilidad de acceso a internet. A continuación, en el apartado 3.3.1, se realizará un análisis de los elementos que conforman este estándar.

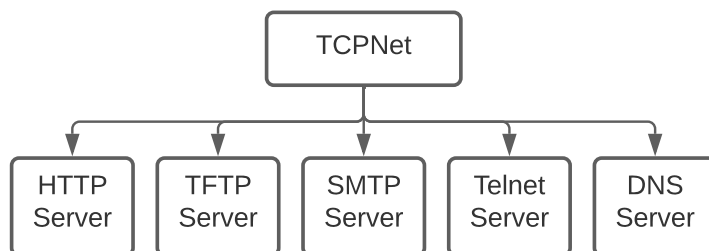
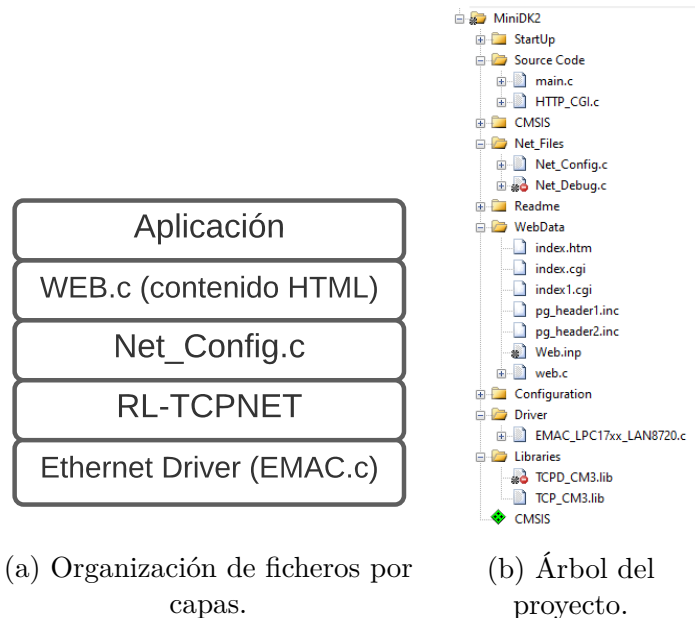


Figura 3.3: Estándares de servidor de la librería TCPNet.

3.3.1 HTTP Server

Dentro de este servicio nos encontramos los siguientes ficheros y su organización por capas, mostrada en la figura 3.4a y el árbol del proyecto resultante en la figura 3.4b.



(a) Organización de ficheros por capas.

(b) Árbol del proyecto.

Figura 3.4: Ficheros de HTTP-Server.

La capa de aplicación es donde se programa la aplicación web, tanto la interfaz de usuario como la lógica de esta. El back-end de la aplicación se programará en lenguaje C (como se verá en 3.4.2) en el archivo **HTTP_CGI.c** y por otro lado, la interfaz web o front-end (apartado 3.4.1) es programada en HTML y posteriormente convertida a lenguaje C como se verá más adelante.

La configuración del servidor Web se realiza desde el archivo **Net_Config.c**. Esto forma parte de la capa de internet dentro del protocolo TCP/IP de nuestro servidor, como se mencionó en el apartado 2.2.1.

La capa de transporte la conforman las librerías **TCP_CM3.lib** y **TCPD_CM3.lib** que vienen ya configuradas por ARM y no deben modificarse. Sin embargo, estos archivos son indispensables para el funcionamiento de la aplicación puesto que es donde se selecciona el uso de la librería TCPNet, estableciendo todos los protocolos para gestionar la comunicación del servidor web.

Por último, la comunicación entre el servidor web y cliente, que forma parte de la capa de enlace de datos, se realiza a través del puerto Ethernet de la tarjeta de desarrollo. Para habilitar este puerto debe agregarse al proyecto el driver **EMAC.c** perteneciente a la librería RL-TCPNet, que se encuentra siguiendo el *path* **C:\KEIL\ARM\RL\TCPNet\Drivers**.

A continuación, se va a realizar un análisis de los elementos que son necesarios modificar para la creación del servidor web; que son los pertenecientes a la capa de aplicación y la capa de internet.

3.3.2 Configuración de la capa de internet: Net_Config.c

El archivo `Net_Config.c`, es una plantilla que permite a los desarrolladores habilitar, de forma rápida y fácil, diversas características de la librería RL-TCPNet, permitiendo definir parámetros básicos de red como la dirección IP, la máscara de subred y la puerta de enlace de red. Este archivo se encuentra siguiendo el *path* `C:\KEIL\ARM\RL\TCPNet\Config` y debe ser agregado a nuestro proyecto.

Los parámetros a modificar son la IP del servidor y la máscara de subred; esta información se obtiene abriendo la ventana de comandos e introduciendo el comando `ipconfig`. Los valores de la dirección IP y de la máscara de subred (rodeados con un cuadro rojo en la figura 3.5) son los ligados al puerto Ethernet donde se conectará la tarjeta de desarrollo de la aplicación.

```
C:\Windows\system32>ipconfig

Configuración IP de Windows

Adaptador de Ethernet Ethernet:

    Sufijo DNS específico para la conexión. . . :
    Vínculo: dirección IPv6 local. . . : fe80::44a3:bb43:97c5:9885%7
    Dirección IPv4. . . . . : 192.168.72.147
    Máscara de subred . . . . . : 255.255.252.0
    Puerta de enlace predeterminada . . . . : 192.168.75.254
```

Figura 3.5: Obtención de la IP y la máscara de subred del servidor.

Para nuestro proyecto contamos con tres ordenadores de laboratorio y tres tarjetas de desarrollo, una por ordenador para que coexistan tres plataformas, y por tanto, deberán configurarse un total de tres servidores web. Las direcciones IP de estos ordenadores son **192.168.72.146**, **192.168.72.147**, **192.168.72.148**, respectivamente. El ordenador tomado como ejemplo en la figura 3.5 es del puesto dos.

Una vez obtenida la IP de nuestro servidor y su máscara de subred se debe abrir el archivo `Net_Config` y modificar los valores de *Local Static IP Address*, donde el valor a introducir para la IP debe variar en una cifra respecto a la IP obtenida del servidor. También, se debe modificar la máscara de subred por los valores obtenidos de la ventana de comandos de la figura 3.5. En el fragmento de código 3.1 se muestran las modificaciones realizadas.

Código 3.1: Configuración de la IP y la máscara de subred del servidor web

```
1 // <h>IP Address
2 // =====
3 // <i> Local Static IP Address
4 // <i> Value 255.255.255.255 is not allowed.
5 // <i> It is a Broadcast IP address.
6 // <o>Address byte 1 <0-255>
7 // <i> Default: 192
8 #define _IP1          192
9
10 // <o>Address byte 2 <0-255>
11 // <i> Default: 168
12 #define _IP2          168
13
14 // <o>Address byte 3 <0-255>
15 // <i> Default: 0
16 #define _IP3          71
17
18 // <o>Address byte 4 <0-255>
19 // <i> Default: 100
20 #define _IP4          147
21
22 // </h>
23 // <h>Subnet mask
24 // =====
25 // <i> Local Subnet mask
26 // <o>Mask byte 1 <0-255>
27 // <i> Default: 255
28 #define _MSK1         255
29
30 // <o>Mask byte 2 <0-255>
31 // <i> Default: 255
32 #define _MSK2         255
33
34 // <o>Mask byte 3 <0-255>
35 // <i> Default: 255
36 #define _MSK3         252
37
38 // <o>Mask byte 4 <0-255>
39 // <i> Default: 0
40 #define _MSK4         0
41
42 // </h>
43
```

3.3.3 Configuración de la capa de aplicación: HTTP_CGI.c y elementos HTML

La capa de aplicación de nuestro proyecto se fundamenta en el protocolo CGI. Este es un protocolo que permite intercambiar datos entre los servidores y las aplicaciones externas de manera estandarizada. Se encuentra entre las tecnologías de interfaz más antiguas de internet, aunque su uso sigue muy extendido en la actualidad.

Empleando dicho protocolo no es necesario que todo el contenido de la página HTML esté disponible en el servidor, sino que este se genera de forma dinámica cuando el usuario realiza la solicitud correspondiente a través de la propia interfaz web.

El protocolo CGI toma la información introducida por el cliente del servidor TCP/IP y la transmite a la aplicación software en forma de una variable de entorno. Además, también permite el envío de información de manera inversa desde la aplicación al cliente. En la imagen 3.6 se muestra el funcionamiento del protocolo.

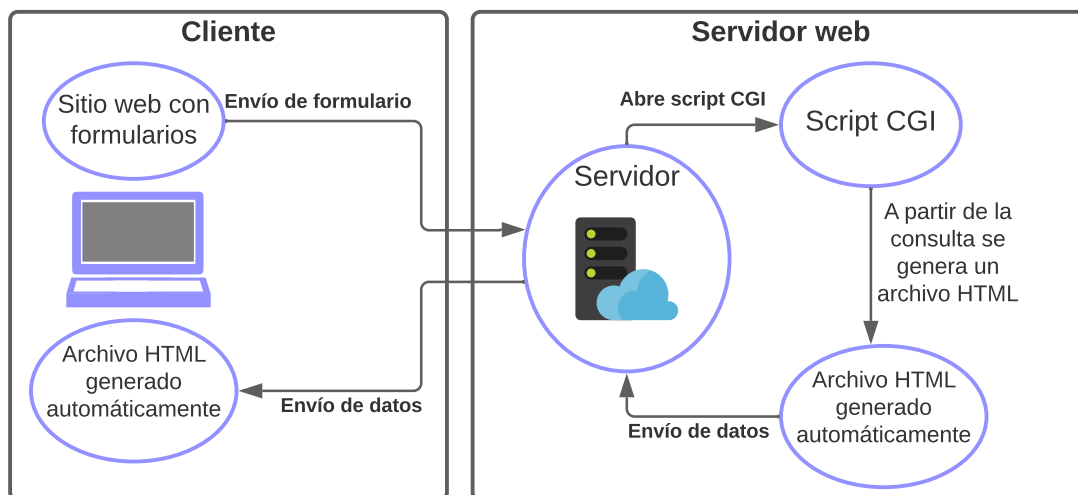


Figura 3.6: Funcionamiento protocolo CGI.

El *script* CGI puede ser escrito en diversos lenguajes de programación, garantizando que, al margen del lenguaje utilizado, el servidor web y el script puedan comunicarse entre sí. En los siguientes apartados (3.3.3.1, 3.3.3.2, 3.3.3.3 y 3.3.3.4) se realiza una descripción de los pasos a seguir y elementos necesarios para generar un servidor web y su interfaz a partir de la librería TCPNet de ARM-RL.

3.3.3.1 HTTP_CGI.C

Para habilitar la interfaz de comunicación CGI es necesario agregar al proyecto el archivo escrito en C `HTTP_CGI.c` perteneciente a la librería RL-TCPNet. Este se encuentra siguiendo el siguiente *path*: `C:\KEIL\ARM\RL\TCPNet\User`. El funcionamiento de este archivo es el siguiente:

Cada vez que el cliente introduzca datos al servidor el protocolo CGI se encarga de tomar dicha información y transmitírsela a la aplicación software como una variable de entorno. En definitiva, el archivo `HTTP_CGI.c` se encarga de relacionar los eventos que suceden en la interfaz web con la aplicación C empleando el protocolo de comunicación CGI descrito en el apartado 3.3.3. Para ello, existen varias funciones y dependiendo del método de solicitud empleado se ejecuta una u otra. Se procede, a continuación, a describir y analizar las funciones pertenecientes a este archivo.

- `cgi_process_data()`: Función encargada del tratamiento de los datos introducidos por el cliente al servidor web cuando se realiza por el método de petición POST. Esta función es nativa del archivo `HTTP_CGI.c` y debe modificarse para agregar tantas expresiones condicionales como elementos con formulario POST existan en nuestra página. Un ejemplo de su uso se puede observar en el código 3.3.
- `cgi_process_var()`: Función encargada de la gestión de los datos enviados por el cliente al servidor web cuando se realiza bajo petición GET. Esta función, al igual que en la anterior, debe modificarse agregando tantas expresiones condicionales como elementos con formulario GET existan en la página. Un ejemplo de su uso se puede observar en el código 3.6.
- `cgi_func()`: Dicha función se encarga de retornar desde el servidor web a la interfaz de usuario los cambios realizados por el cliente tras ser procesados por las funciones `cgi_process_data()` o `cgi_process_var()`. Es decir, permite que la página web sea visualmente dinámica. Ejemplos de su uso se dan en los códigos 3.4 y 3.7, para formulario POST y GET, respectivamente.

3.3.3.2 Páginas web dinámicas

El primer paso a seguir para el desarrollo de la interfaz web es implementar el protocolo CGI a los archivos programados en HTML de nuestra aplicación. Esto es fundamental puesto que la principal funcionalidad de un servidor web basado en un sistema embebido es su capacidad para mostrar y actualizar datos que se encuentran dentro de la aplicación C. Para ello, debe modificarse la extensión a “*.cgi*” de los archivos HTML, en vez de usar “*.htm*” o “*.inc*”. Además, debe implementarse el lenguaje de marcado CGI a nuestros códigos HTML. Este lenguaje contiene cinco comandos básicos que son los mostrados en la tabla 3.1 y deben situarse al comienzo de cada línea en el elemento HTML.

Tabla 3.1: Comandos del lenguaje CGI.

i	Se emplea para incluir un archivo con formato HTML (con extensión <i>.htm</i> o <i>.inc</i>) como cabeceras.
c	Se emplea para aquellos elementos cuyos valores se actualizan en la interfaz web, siendo analizados en la función <i>cgi_func()</i> del archivo <i>HTTP_CGI.c</i> .
.	En todo archivo CGI debe agregarse al final, marcando el final del script.
#	Se emplea para agregar comentarios en el código.
t	Se emplea cuando no son utilizados ninguno de los comandos anteriores. Su funcionalidad es tratar los datos de esta línea como HTML genérico.

3.3.3.3 Entrada de datos: uso de formularios

Una vez se ha definido cómo crear páginas web dinámicas en el apartado 3.3.3.2, ahora vamos a abordar cómo enviar datos desde el navegador web a la aplicación C. Para ello, se emplea el uso de formularios web. Estos son uno de los principales puntos de interacción entre un usuario y un sitio web, o aplicación, cuando los mensajes son ya de cierta complejidad.

Los formularios permiten a los usuarios la introducción de datos, que generalmente se envían a un servidor web para su procesamiento y almacenamiento o se usan en el lado del cliente para provocar, de alguna manera, una actualización inmediata de la interfaz. En nuestro caso, existen dos métodos de entrada de datos compatibles con el módulo CGI de la librería RL-TCPNet. Estos dos métodos son GET y POST, cuya funcionalidad fue descrita en el apartado 2.2.2.3. Ambos se utilizan para introducir datos a través de un formulario de la página HTML.

El método GET debe usarse si los datos de entrada no tienen ningún efecto observable físicamente. Por ejemplo, ingresar un dato a través de un cuadro de texto para modificar una variable interna de un proceso de la aplicación web.

El método POST debe usarse si los datos de entrada se van a usar para cambiar el estado de elementos físicos. Por ejemplo, si se quiere realizar el encendido/apagado de un LED o la generación de una señal PWM.

Para nuestro propósito, la gran parte de nuestra aplicación ha sido implementada con el método POST, exceptuando la entrada de texto donde se ha usado el método GET. A continuación, se va a explicar cómo integrar ambos métodos en nuestra aplicación web.

Integración del método POST

Para permitir que un usuario remoto introduzca datos a través de un navegador web debemos agregar la etiqueta de formulario especificando el método a emplear (línea 3 del código 3.2), el elemento de comunicación (línea 7), ya sea un checkbox o un botón, y un botón de envío de petición (línea 9). Para el ejemplo mostrado en el código 3.2 se toma como elemento de entrada de datos un checkbox.

Código 3.2: Formulario con método POST.

```

1 t <HTML><head><title> POST method example </title></head>
2 t <body>
3 t   <form action="index.cgi" method="POST" name="CGI">
4 t     <table>
5 t       <tr>
6 t         <td>
7 c a   <input name="checkbox" value="ON" type="checkbox" %s></td>
8 t         <td>
9 t       <button name="send" value="ON" type="submit">Send</button>
10 t     </td>
11 t   </tr>
12 t </table>
13 t </form>
14 t </body>
15 t </HTML>
16 . # End of the file

```

Si visualizamos la página desde el navegador podemos observar la generación del checkbox y el botón “Send” tal y como se aprecia en la figura 3.7. Al pulsar el botón de enviar se llama al método POST del formulario enviando la información del checkbox al back-end de la aplicación; es decir, invocando a la función `cgi_process_data()` (código 3.3).

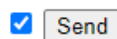


Figura 3.7: Interfaz para checkbox.

Código 3.3: Función de procesamiento de petición POST: `cgi_process_data()`.

```

1 void cgi_process_data (U8 *dat, U16 len) {

```

```

2     bool checkbox;
3     var = (U8 *) alloc_mem (40);
4     do {
5         dat = http_get_var (dat, var, 40);
6         if (var [0] != 0 ) {
7             if (str_scomp (var, "checkbox=ON") == __TRUE) {
8                 checkbox=1;
9             }
10        }
11    }
12 }
13

```

Finalmente, una vez ha sido modificada la variable de entorno “checkbox” ligada a la casilla, se ejecuta la función *cgi_func()* con el fin actualizar el valor de la casilla de manera dinámica. Si se suprime el uso de esta función, al clickar sobre la casilla y enviarlo, el check no se mantendría mostrado en la casilla, quedándose como si no se hubiese cambiado su estado. El código de dicha función se muestra a continuación, en el fragmento 3.4 donde se analiza el caso asignado desde HTML en el código 3.2 al checkbox siendo este el caso *a*.

Código 3.4: Función de emisión de datos caso POST: *cgi_func()*.

```

1 U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
2     switch (env [0]) {
3         case 'a':
4             len = sprintf ((S8 *) buf, (const S8 *) &env [2], checkbox);
5             break;
6     }
7     return ((U16) len);
8 }
9

```

Integración del método GET

La integración del método GET para el envío de datos servidor-cliente de nuestra aplicación se programa de la misma manera que para el método POST. La diferencia es que al definir el tipo de formulario debe cambiarse por “GET” en vez de “POST” como puede apreciarse en la línea 3 del código 3.5. Asimismo, también se modifica el elemento de comunicación siendo, en este caso, un cuadro de texto. Un ejemplo de esto se observa en el fragmento de código 3.5.

Código 3.5: Formulario con método GET

```

1 t <HTML><head><title> GET method example </title></head>
2 t <body>
3 t <form action="index.cgi" method="GET" name="CGI">
4 t <table>
5 t <tr>

```

```

6 t      <td>
7 c a    <input name="textbox" size="16" type="text" value="%"></td>
8 t      <td>
9 t      <button name="send" value="ON" type="submit">Send</button>
10 t     </td>
11 t     </tr>
12 t     <table>
13 t   </form>
14 t </body>
15 t </HTML>
16 . # End of the file

```

Desde el navegador se puede observar cómo se ha generado el cuadro de texto junto al botón de “Send” (Figura 3.8). Tras escribir en el cuadro y pulsar el botón de envío se hace una llamada a la función `cgi_process_var()` de `HTTP_CGI.c`. Esta función se encarga de procesar las peticiones cuando son debidas a un formulario de tipo GET, como se explicó en el apartado 3.3.3.1.



Figura 3.8: Interfaz para cuadro de texto.

En el código 3.6 se muestra el análisis de la petición en el servidor. Primero, debe extraerse el dato del buffer `var` a partir de su octava posición, puesto que el contenido del buffer para este caso es “`textbox= %`”, donde “`%`” es el dato de tipo entero introducido en el cuadro de texto. Tras esto, la función `scanf` lo almacena en la variable de entorno “`buffer`”.

Código 3.6: Función de procesamiento de petición GET: `cgi_process_var()`.

```

1 void cgi_process_var (U8 *qs) {
2     U8 *var;
3     unsigned char buffer [16];
4     var = (U8 *) alloc_mem (40);
5     do {
6         qs = http_get_var (qs, var, 40);
7         if (var [0] != 0) {
8             if (str_scomp (var, "textbox=") == __TRUE) {
9                 sscanf((const char *)&var[8], "%d.", &buffer[0]);
10            }
11        }
12    }while (qs);
13    free_mem ((OS_FRAME *) var);
14 }
15
16

```

Finalmente, una vez almacenado dicho contenido se ejecuta, tal y como sucedía con el método POST, la función `cgi_func()` actualizando en el cuadro de texto el valor introducido por el usuario. Si se suprimiese el uso de dicha función, al introducir un dato en el

cuadro de texto y enviarlo el valor introducido no se mantendría mostrado en el cuadro quedándose vacío. El código de dicha función se muestra a continuación, en el fragmento 3.7.

Código 3.7: Función de emisión de datos caso GET: `cgi_func()`.

```

1  U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
2      switch (env [0]) {
3          case 'a':
4              len = sprintf ((S8 *) buf, (const S8 *) &env [2], buffer);
5              break;
6          }
7      return ((U16) len);
8  }
9  
```

3.3.3.4 Agregar HTML como código C

El contenido que se muestra en el servidor web puede ser de cualquier formato siempre que pueda mostrarlo un navegador. Para la aplicación en cuestión se emplea HTML, que, además, puede incluir imágenes (en formatos comunes como PNG,JPG,...) o enlaces de direccionamiento a múltiples páginas.

Una vez se ha implementado tanto el protocolo CGI y los métodos de comunicación descritos en los apartados 3.3.3.2 y 3.3.3.3, respectivamente, se deben agregar las páginas escritas en HTML a nuestro servidor. Esto se puede hacer por dos métodos:

1. **Conversión de HTML a arrays C:** una vez convertidos, pasan a formar parte del código de la aplicación ejecutándose en el microcontrolador. Este método es el conveniente si lo que se busca es un servidor de tamaño pequeño alojado en un único microcontrolador.
2. **Empleo de la librería RL-Flash:** en este caso, se debe emplear otra de las librerías pertenecientes al paquete RL-ARM usando para la comunicación el protocolo TFTP.

Para nuestra aplicación, será empleado el primer método basado en la librería TCPNet y el protocolo de comunicación HTTP, descritos en el apartado 3.3.1.

Una vez elegido el método, para agregar páginas en formato HTML a nuestro servidor web deben incluirse los archivos con extensión `.cgi`, para aquellas páginas que tienen funcionalidad de comunicación entre servidor-cliente (como `index.cgi` e `index1.cgi`) y con extensión `“.html”` o `“.inc”` para la página de inicio o cabeceras (como `index.htm`, `pg_header1.inc` y `pg_header2.inc`).

Para agregarlos simplemente debe hacerse click derecho sobre la carpeta del proyecto en cuestión y seleccionar el formato de archivo Text Document File, como se muestra en la imagen 3.9.

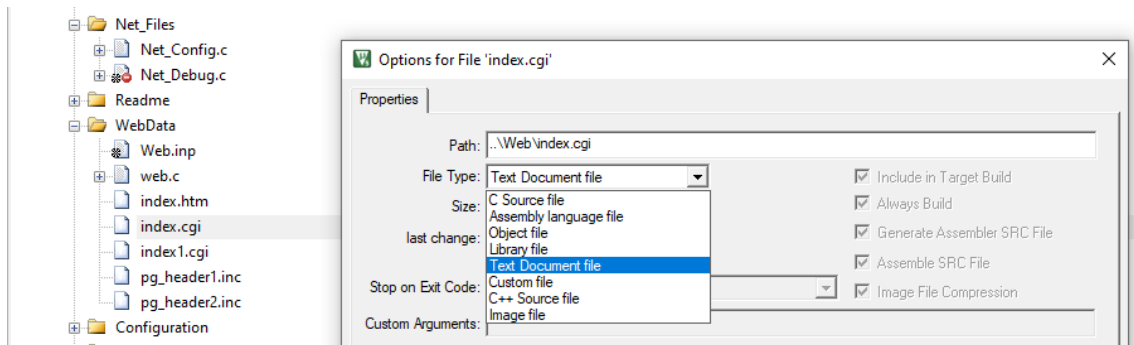


Figura 3.9: Agregar archivos HTML al proyecto.

Cabe resaltar que la diferencia entre el uso de la extensión “.html” e “.inc” es que la primera es una página más dentro de la interfaz web y que las de extensión “.inc” son archivos que se muestran en otras páginas, usualmente empleados para generar cabeceras y pies de página, como es el caso de nuestro proyecto.

Una vez que han sido agregados los elementos HTML necesarios deben convertirse a código C. Para ello, se emplea la funcionalidad integrada en Keil uVision, **FCARM.EXE**. Para implementar el conversor debe agregarse al proyecto un archivo con formato texto plano y llamarlo *Web.inp*, donde se escribirán los archivos a convertir a C ya sean elementos en código HTML, imágenes o iconos a incluir en la interfaz. De igual manera que en el caso previo, este archivo se agrega a la carpeta *WebData* seleccionando su tipo como Custom file. Además, en el apartado “*Custom Arguments:*” se solicita que al compilar se realice la conversión, mediante **FCARM.EXE**, de los archivos contenidos; esto se hace incluyendo el *path* mostrado en la figura 3.10.

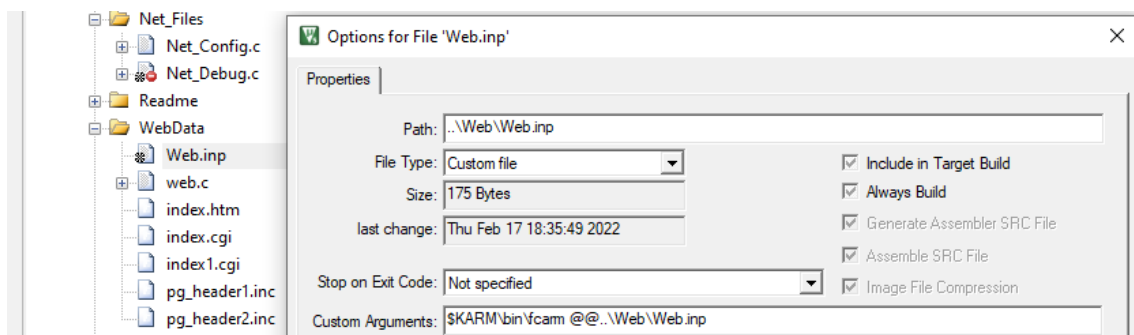


Figura 3.10: Configuración del archivo Web.inp.

Una vez agregado el archivo *Web.inp* se deben indicar los elementos que conforman nuestra interfaz web para convertirlos, tal y como se muestra en el código a continuación.

Código 3.8: Contenido del archivo Web.inp.

```

1 index.htm, index.cgi, index1.cgi, pg_header1.inc, pg_header2.inc, home1.png, depeca.gif,
2 unilogo.gif, analog.png, digital.png, escudo.ico to web.c nopr root(..\Web)

```

Finalmente, los archivos contenidos son analizados y su información es convertida y almacenada como arrays C en el archivo WEB.c, auto-generado por **FCARM.EXE** tras la compilación. En la figura 3.11 se muestra el diagrama del proceso de conversión descrito a lo largo de este apartado.

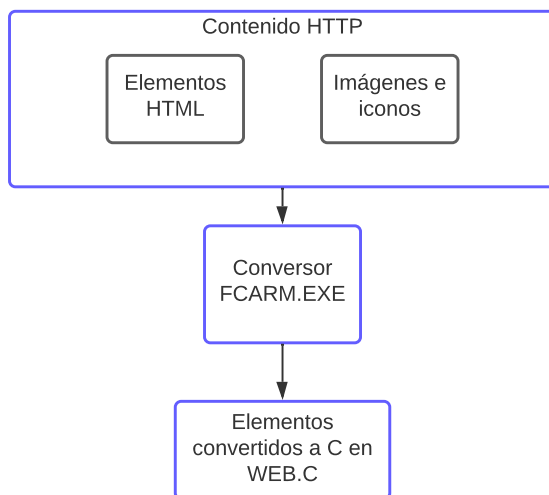


Figura 3.11: Diagrama de la conversión de elementos HTML a código C.

3.4 Diseño front-end y back-end de la aplicación web

Toda aplicación web consta de una interfaz de usuario, denominada **front-end** que es la parte que interactúa con los usuarios, también conocido como lado del cliente. El desarrollo front-end se programa en lenguajes de marcado o no tipados como HTML5/CSS, JavaScript y Ajax.

Por otro lado, se encuentra el diseño **back-end** que es la parte del desarrollo web que se encarga de que toda la lógica de una página web funcione, conocida como el lado del servidor. Se trata del conjunto de acciones que suceden en una web pero que no vemos como, por ejemplo, la comunicación con el servidor. Para el desarrollo de esta parte de la aplicación se emplean lenguajes de programación como C/C++, PHP y Python, entre otros muchos.

En los apartados 3.4.1 y 3.4.2, se realiza una descripción del desarrollo de ambas partes para nuestra aplicación web.

3.4.1 Front-end: desarrollo de la interfaz de usuario

En nuestra aplicación, la interfaz web ha sido desarrollada utilizando lenguaje HTML y, posteriormente, integrada en el proyecto siguiendo las pautas descritas en los apartados 3.3.3.2, 3.3.3.3 y 3.3.3.4.

Por tanto, en este apartado nos centraremos exclusivamente en describir los archivos que conforman el diseño front-end de la aplicación y cómo han sido programados. La carpeta donde se almacenan los elementos es la denominada *WebData* en el árbol del proyecto de la figura 3.4b, donde se pueden destacar dos tipos de archivos:

- **Página principal y cabeceras:** index.htm, pg_header1.inc y pg_header2.inc.
- **Páginas con funcionalidad de comunicación:** index.cgi e index1.cgi.

Comenzamos describiendo aquellas que son de interés visual para la aplicación (cabeceras y página de inicio) y posteriormente, se analizarán las de interés comunicativo.

3.4.1.1 Página principal: index.htm

Tras conectarse al servidor web a través del ordenador remoto, el usuario visualizará una página de bienvenida donde se muestran dos botones que redireccionan a las dos páginas de comunicación, una encargada de generar señales digitales y otra para la generación de señales PWM y señales analógicas, que son index.cgi e index1.cgi, respectivamente. En la figura 3.12 se observa la interfaz desarrollada para esta página.

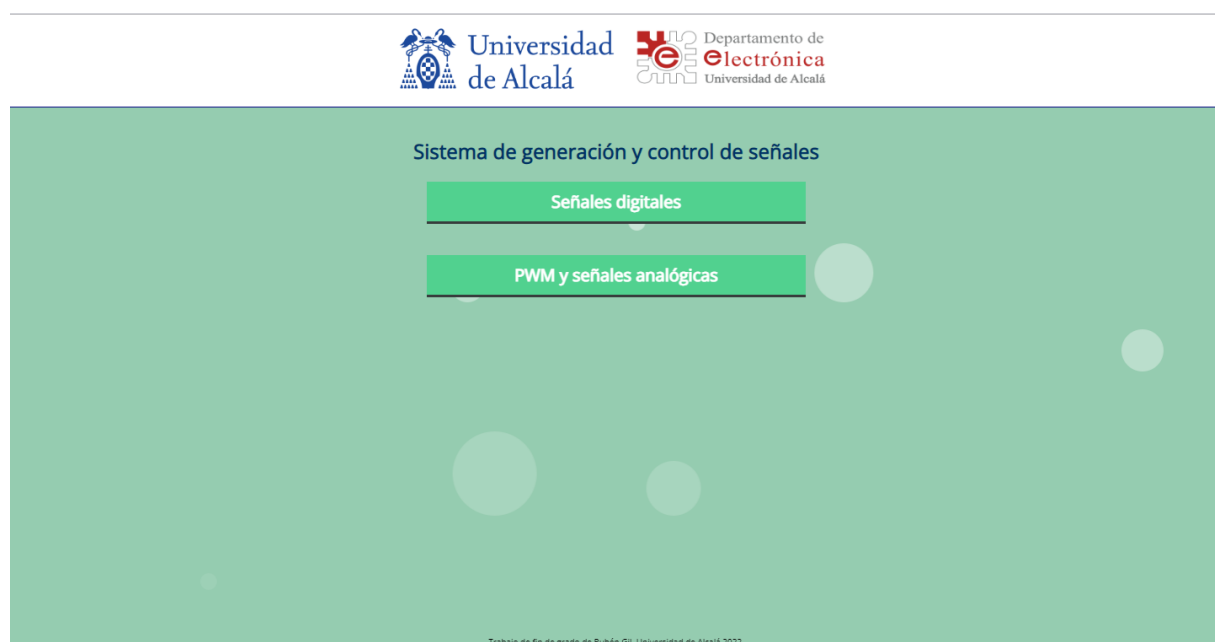


Figura 3.12: Interfaz de la página principal de la aplicación web.

Cabe resaltar que para que el conversor **FCARM.exe** (apartado 3.3.3.4), encargado de convertir los archivos HTML a C, tenga en cuenta qué elemento será la página de bienvenida de nuestra aplicación web este debe aparecer el primero tanto en la carpeta del proyecto *WebData* como en el archivo *Web.inp*, tal y como se muestra en la figura 3.4b y el código 3.8, respectivamente.

Tras esto, se procede a analizar las partes fundamentales del código HTML empleado para realizar la interfaz de esta página. La página esta generada a partir de una tabla de dimensiones igual al ancho y alto del navegador y dividida en distintas celdas, cada una de ellas con una de las partes mostradas en la interfaz.

La primera de estas celdas es la encargada de generar el encabezado de la página. Para ello, se genera una fila y se le da color blanco tal y como se muestra en la línea 3 del fragmento de código 3.9. Tras esto, se divide dicha fila en tres columnas, las mostradas en las líneas 4, 7 y 10. La columna de la línea 4 se emplea para escribir el título de la aplicación web Para ello, se utiliza la etiqueta de título `<h2></h2>` junto con la etiqueta `` encargada de dar formato al texto de su interior. Las líneas 7 y 10 son empleadas para insertar las imágenes de los logos de la Universidad de Alcalá y el Departamento de Electrónica, respectivamente. Para ello, se emplea la etiqueta `` indicando en el campo *src* la imagen a insertar que debe encontrarse alojada en la misma carpeta donde esté el código HTML de la página y en el archivo *Web.inp* mostrado en 3.8 para poder convertir las imágenes a lenguaje C.

Código 3.9: Implementación del encabezado de la página principal.

```

1 <table height=100% cellspacing=0 cellpadding=0 width="100%">
2 <tbody>
3 <tr class="header">
4 <td align=center style="border-bottom: 1px solid #000080" valign=center nowrap height=110
5 
6 
7 </td></tr>

```

Después de generar el encabezado se diseñan los botones de enlace a las dos páginas de nuestra web. Esto se realiza, tal y como se muestra en el código 3.10, empleando las etiqueta de hipervínculo `<a>`, donde en el campo *href* se indica la página web de direccionamiento al clicar el botón (en nuestro caso, *index.cgi* e *index1.cgi*). La etiqueta `` se emplea para dar formato con CSS, creando distintas clases por elemento. Para observar dichas clases generadas, se adjunta el código de la página completo en el anexo.

Código 3.10: Implementación de los botones de navegación de la página principal.

```

1 <a href="/index.cgi" class="button large">

```

```

2 <span class="icon-code"></span>Senales digitales</a>
3
4 <a href="/index1.cgi" class="button large">
5 <span class="icon-code"></span>PWMs y senales analogicas</a></b>

```

3.4.1.2 Encabezados: pg_header1.inc y pg_header2.inc

Los encabezados de las páginas index.cgi e index1.cgi tienen como función principal la navegación rápida entre ambas y volver a la página de bienvenida descrita en 3.4.1.1. En las figuras 3.13a y 3.13b, se muestran los encabezados pg_header1.inc y pg_header2.inc, respectivamente.



Figura 3.13: Encabezados de las páginas index.cgi e index1.cgi.

El código desarrollado para implementar los encabezados de ambas páginas es el mismo que el empleado en el encabezado de la página principal. Es decir, a partir de una tabla subdividida en celdas. Pero, además, se agregan dos botones de navegación a dichas cabeceras.

El primero de ellos permite navegar hasta la página principal y para ello debe realizarse un hiper-vínculo al archivo index.htm empleando la etiqueta `<a>` tal y como se muestra en el fragmento de código 3.11. Además, dicho hipervínculo es mostrado de manera gráfica en la figura 3.14.



Figura 3.14: Icono página principal.

Para generar dicho icono se ha empleado la etiqueta `` donde, como *src*, se selecciona el icono a mostrar que debe estar alojado en la misma carpeta que el código de la página y, también, ha de ser llamado en el archivo Web.inp como se hace en el código 3.8.

Código 3.11: Implementación del botón de redireccionamiento a la página principal.

```

1 <td style="border-bottom: 1px solid #000080" align=center vAlign=center noWrap width="70">
2 <a href="index.htm"></a>
3 </td>

```

Por otro lado, nos encontramos con el segundo icono que difiere dependiendo de la página en la que nos encontremos. Cuando estamos navegando en la página de señales digitales (`index.cgi`) y se desea ir a la de PWM y analógicas (`index1.cgi`), el icono utilizado en ese caso es el mostrado en la figura 3.15a. Para el caso contrario, se emplea el icono 3.15b.



(a) (b)

Figura 3.15: Iconos de navegación entre páginas.

La implementación en código de ambos iconos es análoga a la mostrada en el código 3.11, donde únicamente debe modificarse el atributo `href` de la etiqueta `<a>`, cambiándolo por el archivo de la página a vincular y la fuente de imagen a utilizar en el atributo `src` de la etiqueta ``. Los códigos completos de los encabezados se adjuntan en el anexo.

3.4.1.3 Formato de las páginas `index.cgi` e `index1.cgi`

Una vez que se han diseñado la página de inicio y los encabezados, nos centraremos en la distribución y diseño de la interfaz de las páginas encargadas de generar las diversas señales de nuestra aplicación web, siendo estas `index.cgi` e `index1.cgi`.

Dado que ambas páginas están divididas en dos secciones marcadas, una para la configuración de los parámetros de las señales y otra para su activación/desactivación, se emplean dos formularios por página. Para la sección de configuración de las páginas, se va a emplear el formulario GET, puesto que el elemento de comunicación entre usuario e interfaz son cuadros de texto. Se van a implementar tal y como fue descrito en el apartado 3.3.3.3.

Por otro lado, para las secciones dedicadas a interactuar con las diversas señales, activarlas, pararlas o resetearlas se hará uso del formulario POST; dichas interacciones se harán a través de checkboxes o botones.

Para su integración se hará tal y como fue descrito en el apartado 3.3.3.3. Finalmente, para crear los formularios se va a emplear la etiqueta `<form></form>`. Los atributos de dicha etiqueta son los mostrados en la tabla 3.2.

Tabla 3.2: Atributos de la etiqueta `<forms>`.

id	Define un identificador único (ID) que diferencia a los diversos formularios que se vayan a implementar.
action	Define la ubicación donde se envían los datos que el formulario recopila cuando se validan.
method	Especifica el método de envío de datos a emplear por el formulario.

Por último, destacar el formato que tendrán dichos formularios. Dado que cada formulario debe tener un identificador único, el ID de los formularios GET será *formularioNget* y para formularios POST será *formularioNpost*, siendo el valor de N la página donde se encuentra: 1 para la página de señales digitales y 2 para la página de señales analógicas y PWM.

En el fragmento de código 3.12 se muestra la implementación del formulario GET para la página de señales digitales. De igual manera, en el código 3.13 se observa la implementación del formulario POST para la misma página.

3.4.1.4 Diseño de la interfaz de la página de señales digitales: index.cgi

El archivo index.cgi es un archivo escrito en HTML e integrado en el entorno de Keil empleando los pasos citados en los apartados 3.3.3.2, 3.3.3.3, 3.3.3.4. La funcionalidad de esta página es generar diversas señales digitales, secuencias, el paso a nivel alto/bajo de una señal y controlarlas a partir de los diversos botones de la interfaz. En concreto, la distribución elegida para esta página es la mostrada en la figura 3.16.

Figura 3.16: Interfaz de la página para la generación de señales digitales.

Con el fin de lograr una aplicación web con la mayor versatilidad posible para adecuarse a todos los escenarios de prácticas, se ha optado por el uso de todos los periféricos descritos en el apartado 2.1 y que serán configurados en el apartado 3.4.2.1. Para esta página solo es necesario el uso de temporizadores. Se van a utilizar los cuatro existentes y todas las salidas de las que se disponen, contando en total con diez señales para la generación de secuencias, comenzando en la señal **a** hasta la señal **j**. Estas señales también pueden usarse como señales de cambio de nivel, aunque para dicho propósito han sido agregadas las señales **k** hasta la **n**. A continuación, se va a realizar un análisis de la distribución empleada e implementación en código de cada una de las partes de dicha interfaz.

Sección de la interfaz para configurar las secuencias

Como se puede observar en la figura 3.16, lo primero que visualizamos es el apartado de configuración de secuencias, titulado **Configuración del tiempo por símbolo**, en el cual el usuario modificará el tiempo que permanece activo o inactivo cada símbolo de

las secuencias. Este debe ser un entero entre 0 y 9999999 microsegundos y corresponde con el tiempo de interrupción de cada uno de los timers de la tarjeta (configurados en 3.4.2.1). Por defecto, el valor del tiempo por símbolo de una secuencia es de 1000000 microsegundos, es decir, un segundo por símbolo.

Tal y como se comentó en el apartado 3.4.1.3, los tiempos por símbolo de las secuencias deben introducirse mediante el uso del formulario GET pues los elementos empleados son cuadros de texto. Como todos los cuadros de texto son implementados de la misma manera se va a describir el código para configurar el tiempo por símbolo para las secuencias **a** y **b**, realizándose el resto de forma análoga. Dicho código es el mostrado en el fragmento 3.12.

Código 3.12: Implementación de los cuadros de texto para el tiempo por símbolo.

```

1 t <form id="formulariolget" action="/index.cgi" method="get">
2 ...
3 t <td style="width: 14%" bgcolor="C9EEE8">Tiempo por simbolo senales a y b:</td>
4 c c 1 <td style="width: 10%" bgcolor="C9EEE8"><input type="text" name="tim0" size=4 maxlength=7
   value="%d"> us</td>
5 ...
6 t <tr width="99%" valign="top" align="center">
7 t <td colspan="8"><input type="submit" name="set" value="Send data" id="sbm"></td></tr>
8 t </form>

```

Cada cuadro de texto viene precedido por un nombre que indica el dato a introducir en dicho cuadro. Por ejemplo, para el caso mencionado (secuencias de las señales a y b) se escribe la línea 3 del código 3.12, donde se indica al usuario qué dato debe introducirse en ese cuadro, siendo en este caso *Tiempo por símbolo señales a y b* y en la línea 4 se genera el cuadro de texto para introducirlo. Como se puede observar, el cuadro de texto emplea el comando “c” que se definió en la tabla 3.1 para permitir la actualización del valor de manera dinámica.

Para generar la interfaz de los cuadros de texto se emplea la etiqueta `<input>` cuyos atributos son los mostrados en la tabla 3.3.

Tabla 3.3: Atributos de la etiqueta `<input>` para generar un cuadro de texto.

type	Se emplea para definir el tipo de elemento de entrada de datos a utilizar. En este caso es <code>text</code> , pues se requiere un cuadro de texto.
name	Identificador con el que se manda la información de dicho cuadro de texto al backend de la aplicación.
size	Se emplea para indicar cuántos caracteres de ancho debe tener el campo de entrada.
maxlength	Se emplea para delimitar el tamaño del dato de entrada.
value	Valor por defecto que aparece mostrado en el cuadro de texto. Se emplea <code>%d</code> para que sea dinámico y se actualice con el dato introducido por el usuario.

El identificador asignado a los cuadros de texto para configurar el tiempo por símbolo de las secuencias es *timN*, siendo N un valor de 0 a 3 (puesto que es el tiempo de interrupción de cada timer).

Por último, se debe incluir un botón de envío de datos en la interfaz. Este se muestra en la línea 7 del código 3.12. Para ello se emplea la etiqueta *input*, pero cuyo atributo *type* es definido como “*submit*” que es un tipo predefinido de HTML encargado de generar un botón de envío de datos del formulario al servidor web. El atributo *value* se refiere al texto que aparece en el botón de envío, siendo en este caso *Send data*.

Una vez que el usuario ha modificado los valores de los cuadros de texto y realiza una petición al servidor pulsando el botón de envío de datos, estos serán analizados tal y como se verá en el apartado 3.4.2.2.

Sección de la interfaz para implementación de las secuencias

Tras configurar los parámetros de las secuencias se procede a implementarlas en la interfaz de la página. Lo primero que debe escribirse es el método de comunicación a emplear, tal y como se ha hecho en la sección anterior, donde se usó un formulario con método GET (apartado 3.4.1.4). En este caso, dado que se van a implementar checkboxes para introducir los valores de las secuencias, el método a emplear es POST tal como se describió en el apartado 3.4.1.3,

En el fragmento de código 3.13 se observa la creación de dicho formulario, el cual es el encargado de establecer el método de envío de información tanto para esta sección de la página como en la siguiente (3.4.1.4), encargada del cambio de nivel de las señales. Todo el código HTML realizado para implementar ambas secciones debe hallarse en la línea 2 del código, mostrado como puntos suspensivos por dicha razón.

Código 3.13: Implementación del formulario para método POST en index.cgi.

```

1 t <form id="formulario1post" action="/index.cgi" method="post">
2   ...
3 t </form>

```

Las secuencias han sido agrupadas en secciones identificadas por distintos colores, como se aprecia en la figura 3.16, según el TIMER que empleen, juntando aquellas que usan el mismo. Dado que la implementación de la interfaz para las diez secuencias es la misma, en este apartado se va a describir cómo se ha hecho para las secuencias de las señales **a** y **b** cuya interfaz se muestra en la figura 3.17.

Figura 3.17: Activación de las secuencias de las señales a y b.

Cada una de las secuencias cuenta con dieciséis símbolos por período, cuyo nivel es seleccionado por el usuario a través de los checkboxes que aparecen tras el texto “*Secuencia señal X*”, donde X es la señal de la secuencia (Desde **a** hasta **j**).

La implementación de estos checkboxes se realiza mediante el uso de la etiqueta `<input>` cuyos atributos son los mostrados en la tabla 3.4.

Tabla 3.4: Atributos de la etiqueta `<input>` para generar checkboxes y botones.

name	Asigna un identificador único a cada símbolo.
value	Establece el estado que se fija al identificador cuando se interacciona con el elemento.
type	Define el tipo de elemento de entrada de datos a utilizar.

En el fragmento de código 3.14 se muestra la implementación para la secuencia de la señal **a**. El formato utilizado para los identificadores de cada checkbox es *sig.X_seqY*, donde X es la señal utilizada en la secuencia (desde la **a** hasta la **j**) e Y es la numeración del símbolo de dicha secuencia (desde cero a quince). El estado asignado al atributo *value* al verificar el checkbox es *ON* y el tipo de elemento indicado en el atributo *type* debe ser checkbox. Además, cada checkbox lleva asignado un caso para realizar la actualización dinámica de la interfaz empleado la función *cgi_func()*, que se describió en el apartado 3.3.3.3.

Código 3.14: Implementación de los símbolos de una secuencia.

```

1 t Secuencia señal a:
2 c d 1 <input name="sig.a_seq0" value="ON" type="checkbox" %s >

```

```

3 c d 2 <input name="sig.a_seq1" value="ON" type="checkbox" %s >
4 c d 3 <input name="sig.a_seq2" value="ON" type="checkbox" %s >
5 c d 4 <input name="sig.a_seq3" value="ON" type="checkbox" %s >
6 c d 5 <input name="sig.a_seq4" value="ON" type="checkbox" %s >
7 c d 6 <input name="sig.a_seq5" value="ON" type="checkbox" %s >
8 c d 7 <input name="sig.a_seq6" value="ON" type="checkbox" %s >
9 c d 8 <input name="sig.a_seq7" value="ON" type="checkbox" %s >
10 c d 9 <input name="sig.a_seq8" value="ON" type="checkbox" %s >
11 c d A <input name="sig.a_seq9" value="ON" type="checkbox" %s >
12 c d B <input name="sig.a_seq10" value="ON" type="checkbox" %s >
13 c d C <input name="sig.a_seq11" value="ON" type="checkbox" %s >
14 c d D <input name="sig.a_seq12" value="ON" type="checkbox" %s >
15 c d E <input name="sig.a_seq13" value="ON" type="checkbox" %s >
16 c d F <input name="sig.a_seq14" value="ON" type="checkbox" %s >
17 c d G <input name="sig.a_seq15" value="ON" type="checkbox" %s >

```

Una vez seleccionado el nivel de cada símbolo de la secuencia se ha de elegir el método de funcionamiento; emisión una única vez o de manera cíclica. Esto es posible seleccionarlo desde la casilla que se encuentra consecutiva al texto “*Cíclica X:*”, donde X es la señal empleada en la secuencia (desde la **a** hasta la **j**). Cuando se quieren emitir ambas secuencias de la sección de manera simultánea la implementación es la misma, siendo el texto mostrado en este caso “ X_1 y X_2 cíclicas:” y X_1 y X_2 las señales a emitir.

Por último, se generan tres botones encargados de enviar los datos al servidor y cuyas funcionalidades son las mostradas en la tabla 3.5. Cada vez que el usuario pulse cualquiera de dichos botones se realizará una llamada al servidor gestionándose primero los valores introducidos en los checkboxes, como se describe en el apartado 3.4.2.3 y tras esto se iniciará una acción determinada dependiendo del botón pulsado, cuyo análisis se verá en el apartado 3.4.2.4.

Tabla 3.5: Botones para las secuencias.

Start	Envía el estado de todos los checkboxes relacionados con dicha secuencia y da comienzo a su emisión.
Start all	Envía el estado de todos los checkboxes relacionados con las secuencias y da comienzo a la emisión de estas de manera simultánea.
Stop	Para la emisión de la secuencia dada, pudiendo reanudarse de nuevo al pulsar de nuevo Start.
Stop all	Para la emisión de todas las secuencias pertenecientes a la misma sección, pudiendo reanudarse tras volver a pulsar Start all.
Clear	Para la emisión de la secuencia dada y, además, pone a cero los valores de cada símbolo de la determinada secuencia.
Clear all	Para la emisión de todas las secuencias pertenecientes a la misma sección y pone a cero el valor de cada símbolo de estas.

En el fragmento de código 3.15 se muestra la implementación de los botones de comienzo, pausa y clear para la secuencia **a**, así como el checkbox para seleccionar su periodicidad.

Código 3.15: Implementación de los botones y selección de periodicidad de una secuencia.

```

1 t Ciclica a:
2 c b 1 <input name="signal_cyclic[0]" value="ON" type="checkbox" %s >
3 t <button name="start[0]" value="ON" type="submit">Start</button>
4 t <button name="stop[0]" value="ON" type="submit">Stop</button>
5 t <button name="clear[0]" value="ON" type="submit">Clear</button>

```

Para generar el checkbox encargado de seleccionar la periodicidad de una secuencia se emplea el mismo método usado para establecer el nivel de cada símbolo de las secuencias, visto en el código 3.14. En este caso, el formato del identificador es *signal_cyclic[N]*, comenzando N en cero hasta trece, puesto que diez se emplean para seleccionar el modo de emisión cuando es una secuencia única y cuatro para la emisión en conjunto.

Por otro lado, para generar los botones se emplea la etiqueta `<button></button>` seleccionando el estado del atributo *type* como “*submit*”, lo que hace que al pulsarlos se envíen los datos desde la página al servidor web para ser gestionados en el back-end de la aplicación. Al igual que en los checkboxes, se debe asignar un identificador único a cada botón. Para ello, el formato establecido es *start[N]* para los botones de Start y Start all, *stop[N]* para los de Stop y Stop all y *clear[N]* para los de Clear y Clear all, donde N va desde cero hasta trece.

Sección de cambio de nivel de señales digitales

La última funcionalidad con la que cuenta esta página de la aplicación es la de modificar el nivel de las señales digitales, alto o bajo. En total, han sido implementadas catorce señales con esta funcionalidad donde diez de ellas también cuentan con la función de emitir secuencias. Cabe destacar que, si se está empleando la funcionalidad de secuencia para una señal, no puede ser modificado su nivel hasta que la emisión de la secuencia finalice o sea parada manualmente. Las señales desde la **a** hasta la **j** son las que comparten ambas funcionalidades, mientras que desde la **k** a la **n** solo son señales de paso de nivel alto a bajo. Puesto que la implementación en código es la misma en todas ellas se procede a explicar cómo ha sido implementado para la señal **a**. Para ello, se analiza el fragmento de código 3.16 mostrado a continuación, siendo la figura 3.18 la interfaz resultante de la implementación de este.

Código 3.16: Implementación de los botones de selección del nivel.

```

1 t <td>
2 t Senal a
3 t <br>
4 t <input type="submit" name="sig.a" value="ON">
5 t <input type="submit" name="sig.a" value="OFF">
6 t </td>

```

```

7 c a l <td bgcolor="%s" style="width:10%">
8 t     <div style="color:white;font-weight:bold"><small>Señal a</small></div>
9 t     </td>

```

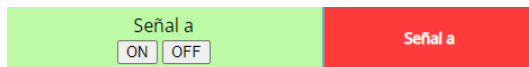


Figura 3.18: Interfaz para el cambio de nivel de la señal “a”.

Para seleccionar el nivel de la señal se han empleado dos botones; uno para el paso a nivel alto, identificado como ON y otro para el paso a nivel bajo, identificado como OFF, tal y como se observa en las líneas 4 y 5 del fragmento 3.16. Sin embargo, a diferencia de los botones vistos en el fragmento de código 3.15 del apartado 3.4.1.4, donde se empleó la etiqueta `<button></button>` para los botones, se ha usado la etiqueta `<input>` cuyos atributos son descritos en la tabla 3.4. Esto es debido a que la etiqueta `<button></button>` no admite la asignación de dos estados en el atributo *value* para un mismo identificador. El formato empleado en los identificadores es *sig.X*, siendo X la señal a la que se desea modificar su nivel (desde a hasta n). Por tanto, cada vez que el usuario pulse uno de los botones se realiza una petición al servidor que se gestiona como se verá en el apartado 3.4.2.4.

Además, se implementa un cuadro dinámico cuyo color varía en función del nivel de la señal. Esto es, cuando está en nivel alto es de color verde y cuando está a nivel bajo cambia a rojo. Dicha funcionalidad es programada en la línea 8 del fragmento de código 3.16, empleando la función *cgi_func()* descrita en el apartado 3.3.3.3. En la figura 3.19 se muestra la activación de la señal “a” tras pulsar el botón de ON.

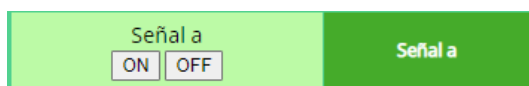


Figura 3.19: Activación de la señal “a”.

3.4.1.5 Diseño de la interfaz de la página de señales analógicas y PWM: index1.cgi

Esta página es la encargada de la generación de señales PWM y analógicas y es codificada en HTML, en el archivo index1.cgi. Para su implementación, se han seguido las instrucciones descritas en los apartados 3.3.3.2, 3.3.3.3 y 3.3.3.4. Al igual que en la página para generar señales digitales (apartado 3.4.1.4), se ha dividido la interfaz en distintas secciones. En total, esta página cuenta con la posibilidad de generar hasta cuatro señales PWM, dos de tipo single edged y dos de tipo double edged, y una señal analógica cuya forma de onda puede ser sinusoidal, triangular o diente de sierra.

La primera sección se emplea para la configuración de los parámetros de las señales analógicas, como su frecuencia y amplitud, y para la configuración de las PWM, seleccionando su frecuencia y los ciclos de trabajo. El método de envío de datos seleccionado es GET, tal y como se mencionó en el apartado dedicado al formato de estas páginas 3.4.1.3, debido a las ventajas que fueron descritas en el apartado 3.3.3.3.

La segunda sección de la interfaz es la encargada de la activación/desactivación de dichas señales, de la selección de las salidas del multiplexor analógico con el que contará la plataforma de prácticas y, además, con unos botones para el manejo del osciloscopio. Esta sección ha sido implementada mediante el uso de método POST, tal y como se mencionó en el apartado 3.4.1.3. La interfaz completa de esta página es la mostrada en la figura 3.20.

Control de señales analógicas

Universidad de Alcalá

Departamento de Electrónica

AVISO: Si se quiere utilizar la funcionalidad de secuencia de las señales i y j, no debe modificarse la frecuencia de las señales analógicas, pues emplean el mismo TIMER.

Configuración señales analógicas

Frecuencia de la señal analógica: 1 Hz

Amplitud de la señal analógica: 0 mV

Configuración señales PWM

Frecuencia de las señales PWM: 100 Hz

Ciclo de trabajo PWM1.1 (single edged): 0 %

Ciclo de trabajo PWM1.2 (single edged): 0 %

Ciclo de trabajo PWM1.3 (doubled edged): 0 %

Ciclo de trabajo PWM1.4 (doubled edged): 0 %

Ciclo de trabajo PWM1.5 (doubled edged): 0 %

Ciclo de trabajo PWM1.6 (doubled edged): 0 %

Send data

Activación señales analógicas

Start Sinusoidal

Start Sierra

Start Triangular

Stop

Activación señales PWM

Start

Stop

Selección multiplexor de la plataforma

Manejo osciloscopio

0 P1.23

2 P3.26

4 P0.11

6 P1.29

P3.25 1

P0.26 3

P0.4 5

P1.28 7

Send

+

-

sel

Figura 3.20: Interfaz de la página para la generación de señales analógicas y PWM.

A continuación, se procede a describir cómo han sido implementadas cada una de las secciones de esta interfaz.

Sección de la interfaz para configurar las señales analógicas y PWM

La sección de configuración de esta página cuenta con dos apartados. Uno denominado **Configuración señal analógica**, donde se permite al usuario configurar parámetros de la señal analógicas, como su frecuencia y su amplitud. El otro apartado, llamado **Configuración de señales PWM**, es dedicado a seleccionar la frecuencia de las PWM, siendo la misma en todas ellas y el ciclo de trabajo de cada señal PWM. Para la introducción de dichos valores se han empleado cuadros de texto y un botón de envío de datos que debe

ser pulsado cada vez que se modifica un parámetro, ya sea de configuración de PWM o señal analógica y cuyo tratamiento en el back-end de nuestra aplicación se verá en el apartado 3.4.2.2.

Dado que la metodología para implementar tanto los cuadros de texto como el botón es análoga a la utilizada en el fragmento de código 3.12 del apartado 3.4.1.4, se va a explicar únicamente el formato de los valores a introducir en dichos elementos.

La frecuencia a introducir para la señal analógica está expresada en Hz y su valor debe estar comprendido entre 0 y 1MHz. Por otro lado, el valor de amplitud de la señal debe ser introducido en mV y estar comprendido entre 0 y 3300mV. La interfaz generada para configurar las señales analógicas puede observarse en la figura 3.21.

Figura 3.21: Interfaz de configuración de las señales analógicas.

Por último, el valor de la frecuencia de las señales PWM está expresado en Hz y comprendido entre 0 y 1MHz. Además, se ha asignado un color a cada una de las señales PWM, existiendo dos señales de tipo single edged y dos de tipo doubled edged. El funcionamiento de las distintas señales PWM, ya sea single edged o doubled edged, fue descrito en el apartado 2.1.3. La interfaz implementada para la configuración de las señales PWM junto con el botón de envío es la mostrada en la figura 3.22.

Configuración PWMs	
Frecuencia de las señales PWM:	100 Hz
Ciclo de trabajo PWM1.1 (single edged):	0 %
Ciclo de trabajo PWM1.2 (single edged):	0 %
Ciclo de trabajo PWM1.3 (doubled edged):	0 %
Ciclo de trabajo PWM1.4 (doubled edged):	0 %
Ciclo de trabajo PWM1.5 (doubled edged):	0 %
Ciclo de trabajo PWM1.6 (doubled edged):	0 %

Send data

Figura 3.22: Interfaz de configuración de las señales PWM.

Sección para la activación de las señales analógicas y PWM

Una vez configurados todos los parámetros necesarios se procede a implementar la interfaz para la sección encargada de generar las señales analógicas y PWM. Para la activación y desactivación de las señales se emplean botones del tipo “submit” indicado en el atributo *type*, que hace que al pulsar sobre estos se produzca el envío inmediato de

los datos al servidor. Los atributos que se han empleado para la implementación de dichos botones son los descritos previamente en la tabla 3.4.

Por un lado, los botones dedicados a la selección de forma de onda y activación de las señales analógicas llevan como identificador *analog*, cuyo valor asignado al pulsar puede ser *Sinusoidal*, *Triangular* y *Sierra* (como se observa en las líneas 3, 4 y 5 del código 3.17). También se cuenta con un botón que detiene la emisión de las señales analógicas cuyo estado asignado al identificador al pulsarlo es *Stop* (línea 6 del código 3.17). Al pulsar cualquiera de dichos botones, se inicia el proceso de petición al servidor debido a las señales analógicas que se describirá en el apartado 3.4.2.4 perteneciente al back-end de la aplicación.

Por ejemplo, al pulsar el botón *Sinusoidal* se selecciona y activa dicho tipo de señal y si se desea cambiar la forma de onda no es necesario detener su emisión pulsando el botón *Stop*, si no que puede pulsarse el de señal *Triangular* o *Sierra* cambiando la forma de onda de manera inmediata. En la figura 3.23 se puede observar la sección de la interfaz dedicada a activar las señales analógicas.



Figura 3.23: Interfaz de activación de las señales analógicas

Para las señales PWM se ha elegido el identificador *PWM* cuyos estados son *Start* y *Stop*, dando comienzo y parando las señales PWM, respectivamente. Estos botones son implementados en las líneas 8 y 9 del código 3.17.

Código 3.17: Implementación de los botones para las señales analógicas y PWM.

```

1 t <form id="formulario2post" action="/index1.cgi" method="post">
2 ...
3 t <td>Start<input type="submit" name="analog" value="Sinusoidal"></td>
4 t <td>Start<input type="submit" name="analog" value="Sierra"></td>
5 t <td>Start<input type="submit" name="analog" value="Triangular"></td>
6 t <td><input type="submit" name="analog" value="Stop"></td>
7 ...
8 t <td><input type="submit" name="PWM" value="Start">
9 t <input type="submit" name="PWM" value="Stop"></td>
10 ...
11 t </form>

```

Por tanto, para las señales PWM son necesarios dos botones, tal y como se ha mencionado previamente, uno para activarlas y otro para detener la emisión de dichas señales. Al pulsar uno de los botones se activan/desactivan todas las PWM de manera simultánea. Además, si en mitad de una emisión se desea cambiar algún parámetro, como el ciclo de trabajo o frecuencia de las señales, se puede hacer sin tener que pararlas. El tratamiento

de la petición debida a estos botones es la indicada en el apartado 3.4.2.4. La interfaz resultante para la activación de las señales PWM es la mostrada en la figura 3.24.

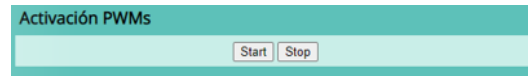


Figura 3.24: Interfaz de activación de las señales PWM.

Sección de la interfaz para el selector del multiplexor analógico

Esta sección está dedicada al selector del multiplexor analógico de ocho salidas cuya finalidad es la selección entre las diversas señales a mostrar en el osciloscopio integrado en la plataforma de prácticas.

La implementación de los bits del selector se ha realizado mediante tres checkboxes empleando la etiqueta `<input>`, cuyos atributos han sido descritos en la tabla 3.4. El código para generar la interfaz de estos checkboxes es el mostrado en el fragmento 3.18.

Código 3.18: Implementación de los bits del selector del multiplexor.

```

1 c i 2 <input name="mux2" value="ON" type="checkbox">
2 c i 2 <input name="mux1" value="ON" type="checkbox">
3 c i 3 <input name="mux0" value="ON" type="checkbox">
4 t <input type="submit" name="mux" value="Send">

```

Cada bit del multiplexor lleva asignado un identificador cuyo formato es $muxN$, siendo N un número de 0 a 2. Por último, el valor que se asigna a cada identificador al verificar el checkbox es *ON*. Además, se ha de implementar un botón de envío de petición como se muestra en la línea 4 del código. Tras pulsar dicho botón, se inicia el tratamiento en el servidor que es gestionado como se verá en el apartado 3.4.2.3 y se encenderá en color verde el canal seleccionado en cada caso. La interfaz resultante generada para esta sección es la mostrada en la figura 3.25.

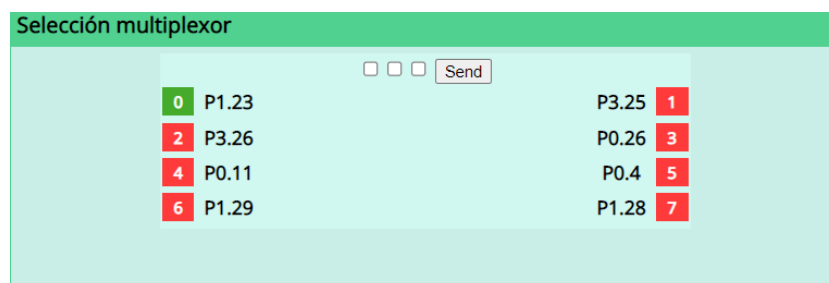


Figura 3.25: Interfaz de selección de los canales del multiplexor.

Sección de la interfaz para el manejo del osciloscopio

Para el manejo del osciloscopio se han generado los botones mostrados en la figura 3.26.

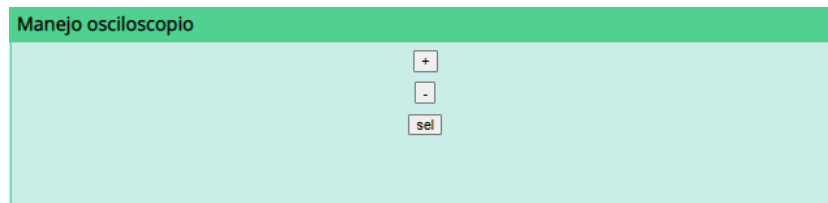


Figura 3.26: Interfaz de manejo del osciloscopio.

Cada vez que se realice la pulsación de uno de los botones el pin cambiará de nivel y modificará la escala (aumentando al pulsar “+” y disminuyendo al pulsar “-”) o seleccionando otro elemento (al pulsar “sel”). Para ello se empleó la etiqueta `<input>` al igual que en el apartado anterior, donde el atributo `value` varía en función del botón pulsado. A continuación, se muestra el código necesario para implementar dichos botones.

Código 3.19: Implementación de los botones para el manejo del osciloscopio.

```

1 t <tr><td><div align="center"><input name="Osciloscopio" value="+" type="submit" id="plus">
   </td></tr>
2 t <tr><td height=5px><div align="center"></td></tr>
3 t <tr><td><div align="center"><input name="Osciloscopio" value="-" type="submit" id="minus">
   </td></tr>
4 t <tr><td height=5px><div align="center"></td></tr>
5 t <tr><td><div align="center"><input name="Osciloscopio" value="sel" type="submit" id="sel">

```

3.4.2 Back-end: Respuesta del servidor web

El diseño back-end de nuestra aplicación ha sido programado en C basándose en las funciones y métodos predefinidos por el entorno de desarrollo Keil uVision, que fueron descritos en el apartado 3.3.3.1.

En este apartado se va a describir las funciones de configuración de los periféricos empleados, además de la implementación y desarrollo de las funciones encargadas de gestionar las peticiones del usuario al servidor. Es decir, todo aquello implementado en el archivo `HTTP_CGI.c` (como se mencionó en el apartado 3.3.3.1).

De cara a los algoritmos de gestión de las peticiones que se describirán en este apartado, se debe mencionar que la comunicación cliente-servidor es de tipo string; el dato que el usuario introduce se almacena en un string de caracteres denominado `var`, cuya información es trasladada a la variable de entorno correspondiente al atender cada petición, ya sea tanto por método GET como POST.

3.4.2.1 Funciones de configuración

Lo primero que debe programarse dentro del back-end de nuestra aplicación son las distintas funciones encargadas de configurar los periféricos de la tarjeta de desarrollo que se utilizarán para la generación y emisión de las diversas señales de la plataforma. Las funcionalidades y características de dichos periféricos fueron descritas en los apartados 2.1.1, 2.1.2 y 2.1.3, para el DAC, los timers y las señales PWM, respectivamente.

Función de configuración del timer 0

La función `configtimer0()` se encarga de configurar el *timer 0* para que interrumpa según el tiempo introducido por el usuario desde la sección de la interfaz descrita en 3.4.1.4 con el fin de generar las secuencias para las señales a y b. Para ello, son empleados MAT0.0 y MAT0.1, junto con sus respectivos pines por donde serán emitidas dichas secuencias. En el fragmento de código 3.20 se muestra la función `configtimer0()`.

Código 3.20: Función de configuración del timer 0.

```

1  #include <stdio.h>
2  #include <LPC17xx.H>
3  #define Fpclk 25e6      // Fcpu/4 (defecto despues del reset)
4
5  uint32_t frecuencia0=1; //valor por defecto del tiempo por simbolo
6
7  /*----- configuration -----*/
8  void configtimer0(){ //configuracion como match para usar para el trigger
9      LPC_SC->PCONP|=0x00000002; //Power up el TIMER 0
10     LPC_PINCON->PINSEL3|= 0x0F000000; // selecciono modo MAT0.0 (P1.28) y MAT0.1(P1.29)
11     LPC_TIMER->MR0=Fpclk/frecuencia0 ; //interrumpe segun la frecuencia definida
12     LPC_TIMER->MR1=Fpclk/frecuencia0 ; //interrumpe segun la frecuencia definida
13     LPC_TIMER->MCR|=0x0000001B; //MR0 y MR1 resetean el TC al hacer match e interrumpen
14     NVIC_SetPriority(TIMERO_IRQn,4); //prioridad del TIMERO
15     NVIC_EnableIRQ(TIMERO_IRQn); //habilito el timer 0
16 }
17

```

Función de configuración del timer 1

El *timer 1* se utilizará para la generación de las secuencias de las señales c y d. Su configuración es realizada desde la función `configtimer1()` donde de igual manera que para el *timer 0* se configura cada cuánto tiempo se interrumpe. Para ello, son empleados MAT1.0 y MAT1.1, junto con sus respectivos pines por donde serán emitidas dichas secuencias. En el fragmento de código 3.21 se muestra la función `configtimer1()`.

Código 3.21: Función de configuración del timer 1.

```

1  #include <LPC17xx.H>

```

```

2  #define Fpclk 25e6
3
4  uint32_t frecuencial=1; //valor por defecto del tiempo por simbolo
5
6  /*----- configuration -----*/
7  void configtimer1(){
8
9      LPC_PINCON->PINSEL3|= 0x000C3000; // selecciono modo MAT1.0 (P1.22) y MAT1.1 (P1.25)
10     LPC_TIM1->MR0=Fpclk/frecuencial ; //interrumpe segun la frecuencia definida
11     LPC_TIM1->MR1=Fpclk/frecuencial ; //interrumpe segun la frecuencia definida
12     LPC_TIM1->MCR|=0x0000001B; //MR0 y MR1 resetean el TC al hacer match e interrumpe
13     NVIC_SetPriority(TIMER1_IRQn,4); //prioridad del TIMER1
14     NVIC_EnableIRQ(TIMER1_IRQn); //habilito el timer 1
15 }
16

```

Función de configuración del timer 2

La función de configuración del *timer 2* es *configtimer2()* y es la encargada de establecer el tiempo de interrupción para la emisión de cada símbolo en las secuencias e,f,g y h. Para ello, han sido empleados MAT2.0, MAT2.1, MAT2.2 y MAT2.3, con sus respectivos pines por los cuales serán emitidas dichas secuencias. La función *configtimer2()* se muestra en el código 3.22.

Código 3.22: Función de configuración del timer 2.

```

1  #include <LPC17xx.H>
2  #define Fpclk 25e6
3
4  uint32_t frecuencia2=1; //valor por defecto del tiempo por simbolo
5
6  /*----- configuration -----*/
7
8  void configtimer2(){
9
10     LPC_SC->PCONP|=(1<<22); //Power up timer 2
11     LPC_PINCON->PINSEL0 |=0x000FF000; //selecciono modo MAT2.0 MAT2.1 MAT2.2 MAT2.3 (del P0.6 al
12     P0.9)
13     LPC_TIM2->PR = 0x0000; //F_tick= 25e6/(0+1) = 25e6 Hz (40ns/cuenta)
14     LPC_TIM2->MR0=Fpclk/frecuencia2 ; //interrumpe segun la frecuencia definida
15     LPC_TIM2->MR1=Fpclk/frecuencia2 ; //interrumpe segun la frecuencia definida
16     LPC_TIM2->MR2=Fpclk/frecuencia2 ; //interrumpe segun la frecuencia definida
17     LPC_TIM2->MR3=Fpclk/frecuencia2 ; //interrumpe segun la frecuencia definida
18     LPC_TIM2->MCR|=0x000006DB; //MR0,MR1,MR2 y MR3 resetean el TC al hacer match e interrumpe
19     NVIC_SetPriority(TIMER2_IRQn,4); //prioridad del TIMER2
20     NVIC_EnableIRQ(TIMER2_IRQn); //habilito el TIMER2
21 }

```

Función de configuración del timer 3

El último temporizador a configurar es el *timer 3* que se encarga de la emisión para las secuencias de las señales *i* y *j*; y además, de realizar la conversión y sacar las muestras de las señales analógicas generadas en el DAC. Su configuración es realizada desde la función *configtimer3()* donde se configura cada cuánto tiempo interrumpe. Para ello, son empleados MAT3.0 y MAT3.1, junto con sus respectivos pines por donde serán emitidas las secuencias. En el fragmento de código 3.23 se muestra la función *configtimer3()*.

Código 3.23: Función de configuración del timer 3.

```

1  #include <LPC17xx.H>
2  #define Fpclk 25e6
3
4  uint32_t frecuencia3=1;//valor por defecto del tiempo por simbolo
5
6  /*----- configuration -----*/
7  void configtimer3(){
8
9      LPC_SC->PCONP|=(1<<23); //POWER TIMER 3
10     LPC_PINCON->PINSEL0 |=0x00F00000; //selecciono modo MAT3.0 (P0.10) y MAT3.1 (P0.11)
11     LPC_TIM3->MR0=Fpclk/frecuencia3; //interrumpe segun la frecuencia definida
12     LPC_TIM3->MR1=Fpclk/frecuencia3; //interrumpe segun la frecuencia definida
13     LPC_TIM3->MCR|=0x0000001B; //MR0 y MR1 resetean el TC al hacer match e interrumpe
14     NVIC_SetPriority(TIMER3_IRQn,4); //prioridad del TIMER1
15     NVIC_EnableIRQ(TIMER3_IRQn); //habilito el timer
16
17 }
18

```

Función de configuración de puertos I/O

Por otro lado, contamos con la funcionalidad de cambio de nivel en señales digitales, el selector del multiplexor y los terminales para el manejo del osciloscopio cuyas interfaces fueron desarrolladas en los apartados 3.4.1.4, 3.4.1.5 y 3.4.1.5, respectivamente. Por tanto, se deben configurar un total de diez terminales de la tarjeta de desarrollo como salidas. Su configuración se realiza desde la función *configPORTS()* y se muestra en el código 3.24.

Código 3.24: Función de configuración de los puertos de salidas.

```

1  #include <LPC17xx.h>
2
3  void configPORTS(void)
4  {
5      LPC_PINCON->PINSEL0 &=0x3FFFF0F0; //configured as gpios pins P0.0,P0.1,P0.4,P0.5 y P0.15
6      LPC_PINCON->PINSEL1 &=0xFFFFFFFF0; //configured as gpios pins P0.16 y P0.17
7      LPC_GPIO0->FIODIR |=0x00038033; //configured as OUTPUTS pins all pins
8  }
9

```

Función de configuración del DAC

El DAC se configura mediante la función *configDAC()*, desde la cual se establece el pin P0.26 como salida desde donde se sacarán las distintas señales analógicas generadas, empleando para ello el *timer 3*. La función de configuración se muestra en el código 3.25.

Código 3.25: Función de configuración del DAC.

```

1  #include <LPC17xx.H>
2  #define Fpclk 25e6
3
4  //configuracion del DAC
5
6  void configDAC(){
7
8      LPC_PINCON->PINSEL1|=(2<<20); //DAC output
9      LPC_PINCON->PINMODE1|=(2<<20); //Deshabilitar pullup-pulldown
10     LPC_SC->PCLKSEL0|= (0x00<<22); // CCLK/4 (Fpclk despues del reset)
11     LPC_DAC->DACCTRL=0;
12 }
13

```

Función de configuración de las señales PWM

Por último, se configuran los periféricos encargados de generar las cuatro señales PWM, dos de tipo single edged y dos de tipo double edged. Para ello, se emplea MR0 marcando el período de las señales PWM, MR1 y MR2 como señales PWM single edged y finalmente, MR3 junto a MR4 y MR5 junto a MR6 como señales PWM double edged. La función encargada de configurar esto es *config_pwm()* y se muestra en el fragmento de código 3.26.

Código 3.26: Función de configuración de las PWM.

```

1  #include <LPC17xx.H>
2  #define Fpclk 25e6 // Fcpu/4
3  uint32_t Freq_PWM=100; // Frecuencia de las PWMs (100Hz) por defecto
4
5  int Tiempo_pwm; //Periodo de las PWMs
6  uint16_t ciclo1,ciclo2,ciclo3,ciclo4,ciclo5,ciclo6=0; //ciclo de trabajo del 0% por defecto
   para todas las PWMs
7
8  void config_pwm(void) {
9
10     LPC_PINCON->PINSEL3|=(2<<4); // P1.18 salida PWM (PWM1.1)
11     LPC_PINCON->PINSEL3&= ~(1<<4); // P1.18 salida PWM (PWM1.1)
12     LPC_PINCON->PINSEL3|=(2<<8); // P1.20 salida PWM (PWM1.2)
13     LPC_PINCON->PINSEL3&= ~(1<<8); // P1.20 salida PWM (PWM1.2)
14     LPC_PINCON->PINSEL3|=(2<<14); // P1.23 salida PWM (PWM1.4)
15     LPC_PINCON->PINSEL3&= ~(1<<14); // P1.23 salida PWM (PWM1.4)
16     LPC_PINCON->PINSEL3|=(2<<20); // P1.26 salida PWM (PWM1.6)
17     LPC_PINCON->PINSEL3&= ~(1<<20); // P1.26 salida PWM (PWM1.6)
18

```

```

19  LPC_SC->PCONP|= (1<<6); // Enable conexion Periferico
20  Tiempo_pwm= (Fpclk/Freq_PWM)-1;
21
22  LPC_PWM1->MR0= Tiempo_pwm;
23  LPC_PWM1->MR1= (Tiempo_pwm/100)*ciclo1; // ciclo de trabajo PWM 1
24  LPC_PWM1->MR2= (Tiempo_pwm/100)*ciclo2; // ciclo de trabajo PWM 2
25  LPC_PWM1->MR3= (Tiempo_pwm/100)*ciclo3; // Paso a nivel alto PWM 3
26  LPC_PWM1->MR4= (Tiempo_pwm/100)*ciclo4; // Paso a nivel bajo PWM 3
27  LPC_PWM1->MR5= (Tiempo_pwm/100)*ciclo5; // Paso a nivel alto PWM 4
28  LPC_PWM1->MR6= (Tiempo_pwm/100)*ciclo6; // Paso a nivel bajo PWM 4
29
30  LPC_PWM1->PCR|= (1<<3|1<<4|1<<5|1<<6); //configurado MR3 y MR4, MR5 y MR6 como double edge
31
32  LPC_PWM1->PCR|= (1<<9|1<<10|1<<12|1<<14); //configurado el ENAs para activar salidas 1,2,4 y 6
33
34  LPC_PWM1->MCR|= (1<<1); //Reset counter al final del periodo
35 }
36

```

3.4.2.2 Gestión de los datos introducidos por cuadros de texto

En nuestro proyecto, la llamada al back-end de la aplicación debido a cuadros de texto se realiza desde las secciones de configuración de las dos páginas de generación de señales, tal y como se describió en los apartados 3.4.1.4 y 3.4.1.5, para señales digitales y analógicas/PWM, respectivamente.

Para la explicación del tratamiento de los datos se retomará el ejemplo descrito en el código 3.12, realizándose el tratamiento de la información de manera análoga para el resto de cuadros de texto.

Una vez que el usuario ha introducido el dato en el cuadro de texto y pulsado el botón de envío de datos se realiza una llamada al servidor web de la aplicación cuya respuesta se encuentra programada en el archivo HTTP_CGI.c. Dado que el tipo de formulario empleado es GET el tratamiento del dato se realiza mediante la función `cgi_process_var()`, que es la encargada de gestionar los envíos al servidor en estos casos, como se indicó en el apartado 3.3.3.1.

En el fragmento de código 3.27 se puede observar cómo se gestiona el dato que el usuario introduce. En este caso el ejemplo es para el cuadro de texto encargado de configurar el tiempo por símbolo de las secuencias **a** y **b**.

Código 3.27: Tratamiento de petición debido a un cuadro de texto.

```

1  void cgi_process_var (U8 *qs) {
2      /* This function is called by HTTP server to process the Query_String */
3      /* for the CGI Form GET method. It is called on SUBMIT from the browser. */
4      /*.The Query_String.is SPACE terminated. */
5      U8 *var;
6          int s[4]; //buffer que almacena el tiempo por simbolo de las secuencias en a y b
7      ...

```

```

8   do {
9       /* Loop through all the parameters. */
10      qs = http_get_var (qs, var, 40);
11      /* Check the returned string, 'qs' now points to the next. */
12      if (var[0] != 0) {
13          /* Returned string is non 0-length. */
14      //-----LECTURA DE ENTRADA DE TEXTO-----
15
16      if (str_scomp (var, "tim0=") == __TRUE) {
17          sscanf((const char *)&var[5], "%d.", &s[0]);
18          tiempo0=s[0];
19      }
20      ...
21      }while (qs);
22      free_mem ((OS_FRAME *)var);
23  }

```

Lo primero que ha de declararse es la variable de entorno donde se va a almacenar el dato leído del string *var*. Para nuestro ejemplo se declara una cadena de tipo entero denominada *s[]* de tamaño cuatro caracteres, tal y como se observa en la línea 6 del código 3.27.

Una vez realizadas las declaraciones pertinentes el programa entra en el bucle *do-while* que comienza en la línea 8 y termina en la 20, el cual se ejecuta mientras existan datos de entrada que aún no hayan sido gestionados. Para analizar si aún quedan datos por tratar se realiza una llamada a la función *http_get_var()*, tal y como sucede en la línea 10. Dicha función se describe a continuación.

- *http_get_var()*: Función perteneciente a la librería RL-TCPnet encargada de procesar los datos provenientes de peticiones GET y POST, identificando dónde termina el primer dato enviado. Devuelve un puntero a la posición del siguiente dato a procesar, o NULL en caso de no existir más datos.

Tras esto, en la línea 12 se comprueba que el string *var* no está vacío y se procede a la lectura del dato. Para realizar dichas lecturas se utiliza la expresión condicional **if** donde se compara, empleando la función *str_comp*, la cadena *var* con el identificador que le fue asignado al cuadro de texto, como se vio en el apartado 3.4.1.4.

Si dicha condición es verdadera el dato se copia mediante la función *sscanf()* en la variable de entorno previamente declarada *s[]*. Finalmente, ese dato se almacena en una variable global para poder acceder a su contenido desde cualquier parte del programa.

3.4.2.3 Gestión de los datos introducidos por checkboxes

En nuestra aplicación, los checkboxes son empleados tanto para establecer el nivel por símbolo de las secuencias, el bit que selecciona la periodicidad de estas y el selector

del multiplexor. Las interfaces diseñadas para estos elementos han sido descritos en los apartados 3.4.1.4 y 3.4.1.5, respectivamente.

Dado que el método de petición para checkboxes es POST, la función empleada para la gestión de los datos es `cgi_process_data()`, tal y como se indicó en el apartado 3.3.3.1. En el código 3.28 se muestra un ejemplo de cómo se gestiona una petición en el servidor debido a un checkbox. Dicho código es la continuación del ejemplo mostrado en el fragmento 3.14.

Código 3.28: Tratamiento de petición debido a un checkbox.

```

1 void cgi_process_data (U8 code, U8 *dat, U16 len) {
2 /* This function is called by HTTP server to process the returned Data */
3 /* for the CGI Form POST method. It is called on SUBMIT from the browser. */
4 /* Parameters: */
5 /* code - callback context code */
6 /* 0 = www-url-encoded form data */
7 /* 1 = filename for file upload (0-terminated string) */
8 /* 2 = file upload raw data */
9 /* 3 = end of file upload (file close requested) */
10 /* 4 = any xml encoded POST data (single or last stream) */
11 /* 5 = the same as 4, but with more xml data to follow */
12 /* Use http_get_content_type() to check the content type */
13 /* dat - pointer to POST received data */
14 /* len - received data length */
15 /*****
16
17 U8 *var;
18 do {
19     dat = http_get_var (dat, var, 40);
20     if (var[0] != 0) {
21         if (str_scomp (var, "sig.a_seq0=ON") == __TRUE) {
22             signal_seq[0][0]=1;
23         }
24         ...
25     }
26 }while (dat);
27 free_mem ((OS_FRAME *)var);
28 }
29

```

Tal y como puede observarse, la gestión del dato es análoga a la realizada en el apartado 3.4.2.2, pese a usar funciones distintas. Sin embargo, al tratarse de checkboxes, cuando el usuario realiza un cambio en estos y los envía al servidor solo son enviados aquellos que han sido verificados. Por tanto, cuando el contenido del string `var` es igual al identificador del checkbox modificado se activa una variable global que indica que determinado símbolo está a nivel alto. Esto se puede observar en la línea 22 del código 3.28.

3.4.2.4 Gestión de los datos introducidos por botones

Los botones son una parte fundamental de la aplicación web puesto que, además de la activación/desactivación de señales, son los encargados del control de las secuencias, señales analógicas y PWM.

Dado que el tipo de formulario empleado para botones es el método POST, la gestión de los datos se realiza a través de la función *cgi_process_data()* al igual que en el apartado anterior (3.4.2.3).

Sin embargo, a diferencia del apartado anterior donde todos los checkboxes seguían el mismo patrón de tratamiento de la petición, en este caso la gestión de los datos es diferente según la funcionalidad del botón. Por ello, se va a analizar cada caso por separado a continuación.

Botones de cambio de nivel en señales

Los botones de cambio de nivel de una señal son los empleados en la página de señales digitales cuya interfaz fue descrita en el apartado 3.4.1.4. Una vez el usuario selecciona el nivel de dicha señal, alto o bajo, se realiza una petición al servidor. La petición será gestionada como se muestra en el fragmento de código 3.29, que pertenece a la función *cgi_process_data()*.

Código 3.29: Tratamiento de petición debido a los botones de cambio de nivel.

```

1 //-----CAMBIO DE NIVEL DE SENALES-----
2 else if (str_scomp (var, "sig.a=ON") == __TRUE) { //Cambio a nivel alto
3     if((start[0]==0)&&(start[2]==0)){ //Cambio de nivel si la secuencia no esta activa
4         signal[0] = 1;
5         LPC_TIM0->EMR|=(1<<0);
6     }
7 }
8 else if (str_scomp (var, "sig.a=OFF") == __TRUE) { //Cambio a nivel bajo
9     if((start[0]==0)&&(start[2]==0)){ //Cambio de nivel si la secuencia no esta activa
10        signal[0] = 0;
11        LPC_TIM0->EMR&=0xFE;
12    }
13 }
14

```

El código 3.29 se trata de la continuación del ejemplo mostrado en 3.16. Para la gestión de la petición se emplean dos expresiones condicionales: una cuando se ha pulsado el botón que activa la señal y otra que la desactiva.

Por tanto, según el botón que haya sido pulsado se analizará el estado del identificador activando/desactivando el pin, previamente configurado en 3.4.2.1, ligado a dicha señal mediante las funciones aportadas por el fabricante de la tarjeta (apartado 2.1).

Botones de funcionalidad en secuencias

Tal y como fue mencionado en el apartado 3.4.1.4 donde fue implementada la interfaz de estos botones, existen tres botones a partir de los cuales se permite al usuario controlar las secuencias de la página web dedicada a las señales digitales (las distintas funcionalidades se describieron en la tabla 3.5).

Cada vez que el usuario pulsa cualquiera de dichos botones, se produce una llamada al servidor sucediendo lo mostrado en el código 3.30, el cual es continuación del ejemplo 3.15 donde se implementó la interfaz de estos.

El código 3.5 contiene tres expresiones condicionales donde se comprueba qué botón ha sido pulsado: el de Start (línea 2), Stop (línea 13) o Clear (línea 24), respectivamente. Para ello, se analiza los identificadores de cada botón y se comprueba si su estado es ON, es decir, si ha sido pulsado por el usuario. Dentro de cada condición se gestionan las distintas variables globales necesarias en cada caso, y a partir de una serie de funciones aportadas por el fabricante se actuará sobre los timers encargados de la emisión de las secuencias.

Código 3.30: Tratamiento de petición debido a los botones de las secuencias.

```

1  //-----CONTROL DE SECUENCIAS-----
2  else if (str_scomp (var, "start[0]=ON") == __TRUE) {
3  //Se solicita el comienzo de la secuencia de la señal "a"
4  if((start[2]==0)&&(start[1]==0)){ //El inicio solo ocurre si no se ha solicitado previamente
      que ambas secuencias del timer salgan a la vez o si la otra secuencia con la que comparte
      timer no se esta ejecutando
5      start[0]=1;
6      stop_only[0]=0;
7      signal[0]=0; //Se desactiva la variable que emite en la web ON/OFF de la señal
8      frecuencia0=1e6/tiempo0;
9      LPC_TIM0->MR0=Fpclk/frecuencia0; //interrumpe segun la frecuencia definida por el
      usuario
10     LPC_TIM0->TCR|=0x01; //start timer
11 }
12 }
13 else if (str_scomp (var, "stop[0]=ON") == __TRUE) {
14 //Se solicita parar la secuencia de la señal a
15 if(start[2]==1){ //Caso en que se solicita parar la secuencia "a" cuando todas las
      secuencias del timer se emiten de forman simultanea
16     stop_only[0]=1;
17 }
18 else if(start[0]==1){ //Caso en que se solicita parar la secuencia "a" cuando se ejecuta
      unicamente esta del timer correspondiente
19     LPC_TIM0->TCR |= 0x02; //resetea el timer
20     LPC_TIM0->TCR&=0xFC; //stop timer
21     start[0]=0;
22 }
23 }
24 else if (str_scomp (var, "clear[0]=ON") == __TRUE) {

```

```

25 //Se solicita el fin de la secuencia de la senal a
26 if(start[2]==1){ //Caso en que se solicita clear de la secuencia "a" cuando todas las
    secuencias del timer se emiten de forman simultanea
27     uint8_t k;
28     LPC_TIM0->EMR&=~(1<<0); //Pongo a cero el P1.28
29
    for(k=0;k<16;k++){ //Reseteo de los bits de la secuencia
30         signal_seq[0][k]=0;
31     }
32 }
33 else if(start[0]==1){ //Caso en que se solicita clear de la secuencia "a" cuando se ejecuta
    unicamente esta del timer correspondiente
34     uint8_t k;
35     LPC_TIM0->TCR |= 0x02; //resetea el timer
36     LPC_TIM0->TCR&=0xFC; //stop timer
37     start[0]=0;
38     LPC_TIM0->EMR&=~(1<<0); //Pongo a cero el P1.28
39     cnt_a=0;
40     ia=0;
41     for(k=0;k<16;k++){ //Reseteo de los bits de la secuencia y bit de periodicidad
42         signal_seq[0][k]=0;
43         signal_cyclic[0]=0;
44     }
45 }
46 }
47

```

Botones para el control de las señales PWM

Tras describir en el apartado 3.4.1.5 cómo se implementa la interfaz de los botones encargados de activar y desactivar las señales PWM, ahora debe analizarse qué sucede cuando el usuario interactúa con ellos, generando una petición al servidor. Para ello, se parte del código 3.17 donde se asignó a los botones de activación/desactivación de los señales PWM el identificador *PWM* y sus dos posibles estados: *Start* y *Stop*.

Cuando el servidor atiende la petición se analiza cuál de los dos estados está activo a partir de las dos expresiones condicionales mostradas en las líneas 2 y 16 del fragmento de código 3.31. Tras esto, empleando las funciones proporcionadas por el fabricante de la tarjeta se actúa sobre los diversos registros encargados de controlar las señales PWM.

Código 3.31: Tratamiento de petición debido a los botones de las señales PWM.

```

1 //-----CONTROL DE SENALES PWM-----
2 else if (str_scomp (var, "PWM=Start") == __TRUE) {
3     LPC_PWM1->TCR&=0xF6; //seleccion de modo PWM counter mode y count input apagado
4     Tiempo_pwm= (Fpclk/Freq_PWM)-1;
5     LPC_PWM1->MR0= Tiempo_pwm; //Periodo de las senales PWM
6     LPC_PWM1->MR1= (Tiempo_pwm/100)*ciclo1;
7     LPC_PWM1->MR2= (Tiempo_pwm/100)*ciclo2;
8     LPC_PWM1->MR3= (Tiempo_pwm/100)*ciclo3;
9     LPC_PWM1->MR4= (Tiempo_pwm/100)*ciclo4;
10    LPC_PWM1->MR5= (Tiempo_pwm/100)*ciclo5;

```

```

11     LPC_PWM1->MR6= (Tiempo_pwm/100)*ciclo6;
12     LPC_PWM1->LER|=(1<<0)|(1<<1)|(1<<2)|(1<<4)|(1<<6);
13     LPC_PWM1->TCR|=(1<<0)|(1<<3); //Activacion modo PWM counter mode y count input
14 }
15
16 else if (str_scomp (var, "PWM=Stop") == __TRUE) {
17     LPC_PWM1->TCR&=0xF6; //Desactivacion modo PWM counter mode y count input
18 }
19

```

Botones para el control de las señales analógicas

Para la selección de las formas de onda y control de las señales analógicas se han diseñado cuatro botones tal y como fue descrito en el apartado 3.4.1.5 donde se implementó la interfaz de estos.

Una vez que el usuario ha seleccionado la forma de onda de la señal analógica a emitir o si desea parar su emisión, se realiza una llamada al servidor web. La respuesta a dicha petición es gestionada como se muestra en el fragmento de código 3.32, donde existe una expresión condicional analizando el estado del identificador *analog* (líneas 2, 14, 26 y 38) y actuando sobre los registros encargados de generar las señales analógicas.

Código 3.32: Tratamiento de petición debido a los botones de las señales analógicas.

```

1 //-----CONTROL DE SEÑALES ANALOGICAS-----
2 else if (str_scomp (var, "analog=Sinusoidal") == __TRUE) {
3     if ((start[11]==0)&&(start[12]==0)&&(start[13]==0)) { //Activo las senales analogicas si no se
4         emiten las secuencias i y j (pues usan el mismo TIMER)
5         LPC_TIM3->TCR |= 0x02; //resetea el timer
6         LPC_TIM3->TCR&=0xFC; //stop timer
7         analog[0]=1;
8         analog[1]=1;
9         analog[2]=0;
10        analog[3]=0;
11        LPC_TIM3->MR0 = (Fpclk/frecuencia3/50)-1; //interrumpe segun la frecuencia definida
12        LPC_TIM3->TCR|=0x01; //start timer
13    }
14 else if (str_scomp (var, "analog=Sierra") == __TRUE) {
15     if ((start[11]==0)&&(start[12]==0)&&(start[13]==0)) { //Activo las senales analogicas si no se
16         emiten las secuencias i y j (pues usan el mismo TIMER)
17         LPC_TIM3->TCR |= 0x02; //resetea el timer
18         LPC_TIM3->TCR&=0xFC; //stop timer
19         analog[0]=1;
20         analog[1]=0;
21         analog[2]=0;
22         analog[3]=1;
23         LPC_TIM3->MR0 = (Fpclk/frecuencia3/50)-1; //interrumpe segun la frecuencia definida
24
25         LPC_TIM3->TCR|=0x01; //start timer
26     }
27 }
28 else if (str_scomp (var, "analog=Triangular") == __TRUE) {

```

```

27     if((start[11]==0)&&(start[12]==0)&&(start[13]==0)){//Activo las senales analogicas si no se
        emiten las secuencias i y j (pues usan el mismo TIMER)
28         LPC_TIM3->TCR |= 0x02; //resetea el timer
29         LPC_TIM3->TCR&=0xFC; //stop timer
30         analog[0]=1;
31         analog[1]=0;
32         analog[2]=1;
33         analog[3]=0;
34         LPC_TIM3->MR0 = (Fpclk/frecuencia3/50)-1; //interrumpe segun la frecuencia definida
35         LPC_TIM3->TCR|=0x01; //start timer
36     }
37 }
38 else if (str_scomp (var, "analog=Stop") == __TRUE) {
39     if((start[11]==0)&&(start[12]==0)&&(start[13]==0)){//Activo las senales analogicas si no se
        emiten las secuencias i y j (pues usan el mismo TIMER)
40         LPC_TIM3->TCR |= 0x02; //resetea el timer
41         LPC_TIM3->TCR&=0xFC; //stop timer
42         analog[0]=0;
43         analog[1]=0;
44         analog[2]=0;
45         analog[3]=0;
46     }
47 }
48

```

Botones para el manejo del osciloscopio

Para modificar la escala del osciloscopio se implementaron los botones mostrados en la sección 3.4.1.5. Para evitar el bloqueo del osciloscopio debido a la activación simultánea de varios botones de este, cuando se pulse uno de ellos el resto se activarán a nivel alto para evitar dicho comportamiento.

Código 3.33: Tratamiento de petición debido a los botones de manejo del osciloscopio.

```

1  else if (str_scomp (var, "Osciloscopio=+") == __TRUE) {
2      if (Osciloscopio[0]==0){
3          Osciloscopio[1]=1;
4          Osciloscopio[2]=1;
5          LPC_GPIO1->FIOSET |=0x40000000; //nivel alto en P1.30
6          LPC_GPIO0->FIOSET |=0x08000000; //nivel alto en P0.27
7          LPC_GPIO1->FIOCLR |=0x80000000; //nivel bajo en P1.31
8          Osciloscopio[0]=1;
9      }
10     else if(Osciloscopio[0]){
11         LPC_GPIO1->FIOSET |=0x80000000; //nivel alto en P1.31
12         Osciloscopio[0]=0;
13     }
14 }
15 else if (str_scomp (var, "Osciloscopio=-") == __TRUE) {
16     if (Osciloscopio[1]==0){
17         Osciloscopio[0]=1;
18         Osciloscopio[2]=1;
19         LPC_GPIO1->FIOSET |=0x80000000; //nivel alto en P1.31
20         LPC_GPIO0->FIOSET |=0x08000000; //nivel alto en P0.27

```

```

21     LPC_GPIO1->FIOCLR |=0x40000000; //nivel bajo en P1.30
22     Osciloscopio[1]=1;
23 }
24 else if(Osciloscopio[1]){
25     LPC_GPIO1->FIOSET |=0x40000000; //nivel alto en P1.30
26     Osciloscopio[1]=0;
27 }
28 }
29 else if (str_scomp (var, "Osciloscopio=sel") == __TRUE) {
30     if (Osciloscopio[2]==0){
31         Osciloscopio[0]=1;
32         Osciloscopio[1]=1;
33         LPC_GPIO1->FIOSET |=0x40000000; //nivel alto en P1.30
34         LPC_GPIO1->FIOSET |=0x80000000; //nivel alto en P1.31
35         LPC_GPIO0->FIOCLR |=0x08000000; //nivel bajo en P0.27
36
37         Osciloscopio[2]=1;
38     }
39     else if (Osciloscopio[2]){
40         LPC_GPIO0->FIOSET |=0x08000000; //nivel alto en P0.27
41         Osciloscopio[2]=0;
42     }

```

3.4.2.5 Funciones de interrupción

Para la emisión de las secuencias digitales y la conversión digital a analógico se han empleado los cuatro Timers con los que cuenta la tarjeta de desarrollo mini-DK2, que fueron descritos en el apartado 2.1.2. Por tanto, contamos con un total de cuatro rutinas de atención a la interrupción donde la funcionalidad de cada una de ellas se describe a continuación.

TIMER0_IRQHandler()

Esta función implementa la rutina de interrupción del *Timer 0*. Dicho Timer se encarga de emitir las secuencias de las señales a y b una vez se ha atendido la petición en el servidor tal y como se explicó en el apartado 3.4.2.4.

Código 3.34: Rutina de atención a la interrupción del Timer 0.

```

1 void TIMER0_IRQHandler() {
2     if(start[0]==1){ //Instrucciones para MAT0.0
3         LPC_TIMO->IR|=(1<<0); //borra el flag de MRO
4         cnt_a++;
5         if(signal_seq[0][ia]==1){
6             LPC_TIMO->EMR|=(signal_seq[0][ia]<<0); //Emision de los simbolos a nivel alto de la
              secuencia a
7         }
8         else{
9             LPC_TIMO->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia a
10        }
11        ia++;

```

```

12  if(cnt_a==16){ //Fin de emison: reseteo variables y compruebo periodicidad
13      cnt_a=0;
14      ia=0;
15      if(signal_cyclic[0]==0){ //Si no se ha solicitado senal periodica
16          start[0]=0;
17          LPC_TIM0->TCR |= 0x02; //resetea el timer
18          LPC_TIM0->TCR&=0xFC; //stop timer
19      }
20  }
21 }
22 else if(start[1]==1){ //Instrucciones para MAT0.1
23     LPC_TIM0->IR|=(1<<1); //borra el flag de MR1
24     cnt_b++;
25     if(signal_seq[1][ib]==1){
26         LPC_TIM0->EMR|=(signal_seq[1][ib]<<1); //Emision de los simbolos a nivel alto de la
           secuencia b
27     }
28     else{
29         LPC_TIM0->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia b
30     }
31     ib++;
32     if(cnt_b==16){ //Fin de emison: reseteo variables y compruebo periodicidad
33         cnt_b=0;
34         ib=0;
35         if(signal_cyclic[1]==0){ //Si no se ha solicitado senal periodica
36             start[1]=0;
37             LPC_TIM0->TCR |= 0x02; //resetea el timer
38             LPC_TIM0->TCR&=0xFC; //stop timer
39         }
40     }
41 }
42 else if(start[2]==1){ //ejecuta las instrucciones si funcionan ambos
43     cnt_ab++;
44     if(((LPC_TIM0->IR&0x01)==1)&&(stop_only[0]==0)){ //secuencia a
45         if(signal_seq[0][iab]==1){
46             LPC_TIM0->EMR|=(signal_seq[0][iab]<<0); //Emision de los simbolos a nivel alto de la
           secuencia a
47         }
48         else{
49             LPC_TIM0->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia a
50         }
51     }
52     if(((LPC_TIM0->IR&0x02)==2)&&(stop_only[1]==0)){ //secuencia b
53         if(signal_seq[1][iab]==1){
54             LPC_TIM0->EMR|=(signal_seq[1][iab]<<1); //Emision de los simbolos a nivel alto de la
           secuencia b
55         }
56         else{
57             LPC_TIM0->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia b
58         }
59     }
60     LPC_TIM0->IR|=(1<<0); //borra el flag de MR0
61     LPC_TIM0->IR|=(1<<1); //borra el flag de MR1
62     iab++;
63     if(cnt_ab==16){ //Fin de emison: reseteo variables y compruebo periodicidad
64         cnt_ab=0;
65         iab=0;
66         if(signal_cyclic[2]==0){ //Si no se ha solicitado senal periodica
67             start[2]=0;

```

```

68     LPC_TIM0->TCR |= 0x02; //resetea el timer
69     LPC_TIM0->TCR&=0xFC; //stop timer
70 }
71 }
72 }
73 }
74

```

TIMER1_IRQHandler()

Esta función es la rutina de atención a la interrupción del *Timer 1*. Este segundo Timer es el encargado de la emisión de las secuencias de las señales c y d. El código implementado en dicha rutina es el mostrado en 3.35.

Código 3.35: Rutina de atención a la interrupción del Timer 1.

```

1  void TIMER1_IRQHandler() {
2  if(start[3]==1){ //Instrucciones para MAT1.0
3      LPC_TIM1->IR|=(1<<0); //borra el flag de MR0
4      cnt_c++;
5      if(signal_seq[2][ic]==1){
6          LPC_TIM1->EMR|=(signal_seq[2][ic]<<0); //Emision de los simbolos a nivel alto de la
          secuencia c
7      }
8      else{
9          LPC_TIM1->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia c
10     }
11     ic++;
12     if(cnt_c==16){ //Fin de emison: reseteo variables y compruebo periodicidad
13         cnt_c=0;
14         ic=0;
15         if(signal_cyclic[3]==0){ //Si no se ha solicitado senal periodica
16             start[3]=0;
17             LPC_TIM1->TCR |= 0x02; //resetea el timer
18             LPC_TIM1->TCR&=0xFC; //stop timer
19         }
20     }
21 }
22 else if(start[4]==1){ //Instrucciones para MAT1.1
23     LPC_TIM1->IR|=(1<<1); //borra el flag de MR1
24     cnt_d++;
25     if(signal_seq[3][id]==1){
26         LPC_TIM1->EMR|=(signal_seq[3][id]<<1); //Emision de los simbolos a nivel alto de la
          secuencia d
27     }
28     else{
29         LPC_TIM1->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia d
30     }
31     id++;
32     if(cnt_d==16){ //Fin de emison: reseteo variables y compruebo periodicidad
33         cnt_d=0;
34         id=0;
35         if(signal_cyclic[4]==0){ //Si no se ha solicitado senal periodica
36             start[4]=0;
37             LPC_TIM1->TCR |= 0x02; //resetea el timer

```



```

38     LPC_TIM1->TCR&=0xFC; //stop timer
39     }
40     }
41 }
42 else if(start[5]==1){ //ejecuta las instrucciones si funcionan ambos
43     cnt_cd++;
44     if(((LPC_TIM1->IR&0x01)==1)&&(stop_only[2]==0)){ //secuencia c
45         if(signal_seq[2][icd]==1){
46             LPC_TIM1->EMR|=(signal_seq[2][icd]<<0); //Emision de los simbolos a nivel alto de la
                secuencia c
47         }
48         else{
49             LPC_TIM1->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia c
50         }
51     }
52     if(((LPC_TIM1->IR&0x02)==2)&&(stop_only[3]==0)){ //secuencia d
53         if(signal_seq[3][icd]==1){
54             LPC_TIM1->EMR|=(signal_seq[3][icd]<<1); //Emision de los simbolos a nivel alto de la
                secuencia d
55         }
56         else{
57             LPC_TIM1->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia d
58         }
59     }
60     LPC_TIM1->IR|=(1<<0); //borra el flag de MR0
61     LPC_TIM1->IR|=(1<<1); //borra el flag de MR1
62     icd++;
63     if(cnt_cd==16){ //Fin de emison: reseteo variables y compruebo periodicidad
64         cnt_cd=0;
65         icd=0;
66         if(signal_cyclic[5]==0){ //Si no se ha solicitado senal periodica
67             start[5]=0;
68             LPC_TIM1->TCR |= 0x02; //resetea el timer
69             LPC_TIM1->TCR&=0xFC; //stop timer
70         }
71     }
72 }
73 }
74

```

TIMER2_IRQHandler()

La rutina de atención a la interrupción del Timer 2 es empleada para la emisión de las secuencias de las señales *e*, *f*, *g* y *h*. El código para dicha rutina es el mostrado en el fragmento 3.36.

Código 3.36: Rutina de atención a la interrupción del Timer 2.

```

1 void TIMER2_IRQHandler() {
2     if(start[6]==1){ //Instrucciones para MAT2.0
3         LPC_TIM2->IR|=(1<<0); //borra el flag de MR0
4         cnt_e++;
5         if(signal_seq[4][ie]==1){
6             LPC_TIM2->EMR|=(signal_seq[4][ie]<<0); //Emision simbolos a nivel alto de secuencia e
7         }

```

```

8     else{
9         LPC_TIM2->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia e
10    }
11    ie++;
12    if(cnt_e==16){ //Fin de emison: reseteo variables y compruebo periodicidad
13        cnt_e=0;
14        ie=0;
15        if(signal_cyclic[6]==0){ //Si no se ha solicitado senal periodica
16            start[6]=0;
17            LPC_TIM2->TCR |= 0x02; //resetea el timer
18            LPC_TIM2->TCR&=0xFC; //stop timer
19        }
20    }
21 }
22 else if(start[7]==1){ //Instrucciones para MAT2.1
23     LPC_TIM2->IR|=(1<<1); //borra el flag de MR1
24     cnt_f++;
25     if(signal_seq[5][iF]==1){
26         LPC_TIM2->EMR|=(signal_seq[5][iF]<<1); //Emision simbolos a nivel alto de secuencia f
27     }
28     else{
29         LPC_TIM2->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia f
30     }
31     iF++;
32     if(cnt_f==16){ //Fin de emison: reseteo variables y compruebo periodicidad
33         cnt_f=0;
34         iF=0;
35         if(signal_cyclic[7]==0){ //Si no se ha solicitado senal periodica
36             start[7]=0;
37             LPC_TIM2->TCR |= 0x02; //resetea el timer
38             LPC_TIM2->TCR&=0xFC; //stop timer
39         }
40     }
41 }
42 else if(start[8]==1){ //Instrucciones para MAT2.2
43     LPC_TIM2->IR|=(1<<2); //borra el flag de MR2
44     cnt_g++;
45     if(signal_seq[6][ig]==1){
46         LPC_TIM2->EMR|=(signal_seq[6][ia]<<2); //Emision simbolos a nivel alto de secuencia g
47     }
48     else{
49         LPC_TIM2->EMR&=~(1<<2); //Emision de los simbolos a nivel bajo de la secuencia g
50     }
51     ig++;
52     if(cnt_g==16){ //Fin de emison: reseteo variables y compruebo periodicidad
53         cnt_g=0;
54         ig=0;
55         if(signal_cyclic[8]==0){ //Si no se ha solicitado senal periodica
56             start[8]=0;
57             LPC_TIM2->TCR |= 0x02; //resetea el timer
58             LPC_TIM2->TCR&=0xFC; //stop timer
59         }
60     }
61 }
62 else if(start[9]==1){ //Instrucciones para MAT2.3
63     LPC_TIM2->IR|=(1<<3); //borra el flag de MR3
64     cnt_h++;
65     if(signal_seq[7][ih]==1){
66         LPC_TIM2->EMR|=(signal_seq[7][ih]<<3); //Emision simbolos a nivel alto de secuencia h

```

```

67     }
68     else{
69         LPC_TIM2->EMR&=~(1<<3); //Emision de los simbolos a nivel bajo de la secuencia h
70     }
71     ih++;
72     if(cnt_h==16){ //Fin de emison: reseteo variables y compruebo periodicidad
73         cnt_h=0;
74         ih=0;
75         if(signal_cyclic[9]==0){ //Si no se ha solicitado senal periodica
76             start[9]=0;
77             LPC_TIM2->TCR |= 0x02; //resetea el timer
78             LPC_TIM2->TCR&=0xFC; //stop timer
79         }
80     }
81 }
82 else if(start[10]==1){ //ejecuta las instrucciones si funcionan todos
83     cnt_efgh++;
84     if(((LPC_TIM2->IR&0x01)==1)&&(stop_only[4]==0)){ //secuencia e
85         if(signal_seq[4][iefgh]==1){
86             LPC_TIM2->EMR|=(signal_seq[4][iefgh]<<0); //Emision simbolos a nivel alto de secuencia e
87         }
88         else{
89             LPC_TIM2->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia e
90         }
91     }
92     if(((LPC_TIM2->IR&0x02)==2)&&(stop_only[5]==0)){ //secuencia f
93         if(signal_seq[5][iefgh]==1){
94             LPC_TIM2->EMR|=(signal_seq[5][iefgh]<<1); //Emision simbolos a nivel alto de secuencia f
95         }
96         else{
97             LPC_TIM2->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia f
98         }
99     }
100     if(((LPC_TIM2->IR&0x04)==4)&&(stop_only[6]==0)){ //secuencia g
101         if(signal_seq[6][iefgh]==1){
102             LPC_TIM2->EMR|=(signal_seq[6][iefgh]<<2); //Emision simbolos a nivel alto de secuencia g
103         }
104         else{
105             LPC_TIM2->EMR&=~(1<<2); //Emision de los simbolos a nivel bajo de la secuencia g
106         }
107     }
108     if(((LPC_TIM2->IR&0x08)==8)&&(stop_only[7]==0)){ //secuencia h
109         if(signal_seq[7][iefgh]==1){
110             LPC_TIM2->EMR|=(signal_seq[7][iefgh]<<3); //Emision simbolos a nivel alto de secuencia h
111         }
112         else{
113             LPC_TIM2->EMR&=~(1<<3); //Emision de los simbolos a nivel bajo de la secuencia h
114         }
115     }
116     LPC_TIM2->IR|=(1<<0); //borra el flag de MR0
117     LPC_TIM2->IR|=(1<<1); //borra el flag de MR1
118     LPC_TIM2->IR|=(1<<2); //borra el flag de MR2
119     LPC_TIM2->IR|=(1<<3); //borra el flag de MR3
120     iefgh++;
121     if(cnt_efgh==16){ //Fin de emison: reseteo variables y compruebo periodicidad
122         cnt_efgh=0;
123         iefgh=0;
124         if(signal_cyclic[10]==0){ //Si no se ha solicitado senal periodica
125             start[10]=0;

```

```

126     LPC_TIM2->TCR |= 0x02;    //resetea el timer
127     LPC_TIM2->TCR&=0xFC; //stop timer
128 }
129 }
130 }
131 }
132

```

TIMER3_IRQHandler()

Esta función implementa la rutina de atención a la interrupción del Timer 3. Dicho Timer es el encargado de la emisión de las secuencias i y j y también, de emitir la señal analógica. El fragmento de código 3.37 muestra cómo ha sido programada dicha rutina.

Código 3.37: Rutina de atención a la interrupción del Timer 3.

```

1 void TIMER3_IRQHandler() {
2     if(analog[0]==0){ //Caso en que se ha solicitado las secuencias de las senales i y j
3         if(start[11]==1){ //Instrucciones para MAT3.0
4             LPC_TIM3->IR|=(1<<0); //borra el flag de MRO
5             cnt_i++;
6             if(signal_seq[8][ii]==1){
7                 LPC_TIM3->EMR|=(signal_seq[8][ii]<<0); //Emision simbolos a nivel alto de secuencia i
8             }
9             else{
10                LPC_TIM3->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia i
11            }
12            ii++;
13            if(cnt_i==16){ //Fin de emison: reseteo variables y compruebo periodicidad
14                cnt_i=0;
15                ii=0;
16                if(signal_cyclic[11]==0){ //Si no se ha solicitado senal periodica
17                    start[11]=0;
18                    LPC_TIM3->TCR |= 0x02;    //resetea el timer
19                    LPC_TIM3->TCR&=0xFC; //stop timer
20                }
21            }
22        }
23        else if(start[12]==1){ //Instrucciones para MAT3.1
24            LPC_TIM3->IR|=(1<<1); //borra el flag de MR1
25            cnt_j++;
26            if(signal_seq[9][ij]==1){
27                LPC_TIM3->EMR|=(signal_seq[9][ij]<<1); //Emision simbolos a nivel alto de secuencia j
28            }
29            else{
30                LPC_TIM3->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia j
31            }
32            ij++;
33            if(cnt_j==16){ //Fin de emison: reseteo variables y compruebo periodicidad
34                cnt_j=0;
35                ij=0;
36                if(signal_cyclic[12]==0){ //Si no se ha solicitado senal periodica
37                    start[12]=0;
38                    LPC_TIM3->TCR |= 0x02;    //resetea el timer
39                    LPC_TIM3->TCR&=0xFC; //stop timer

```

```

40     }
41   }
42 }
43 else if(start[13]==1){ //ejecuta las instrucciones si funcionan ambos
44   cnt_ij++;
45   if(((LPC_TIM3->IR&0x01)==1)&&(stop_only[8]==0)){ //secuencia i
46     if(signal_seq[8][iij]==1){
47       LPC_TIM3->EMR|=(signal_seq[8][iij]<<0); //Emision simbolos a nivel alto de secuencia i
48     }
49     else{
50       LPC_TIM3->EMR&=~(1<<0); //Emision de los simbolos a nivel bajo de la secuencia i
51     }
52   }
53   if(((LPC_TIM3->IR&0x02)==2)&&(stop_only[9]==0)){ //secuencia j
54     if(signal_seq[9][iij]==1){
55       LPC_TIM3->EMR|=(signal_seq[9][iij]<<1); //Emision simbolos a nivel alto de secuencia j
56     }
57     else{
58       LPC_TIM3->EMR&=~(1<<1); //Emision de los simbolos a nivel bajo de la secuencia j
59     }
60   }
61   LPC_TIM3->IR|=(1<<0); //borra el flag de MRO
62   LPC_TIM3->IR|=(1<<1); //borra el flag de MR1
63   iij++;
64   if(cnt_ij==16){ //Fin de emison: reseteo variables y compruebo periodicidad
65     cnt_ij=0;
66     iij=0;
67     if(signal_cyclic[13]==0){ //Si no se ha solicitado senal periodica
68       start[13]=0;
69       LPC_TIM3->TCR |= 0x02; //resetea el timer
70       LPC_TIM3->TCR&=0xFC; //stop timer
71     }
72   }
73 }
74 }
75 else { //Caso en que se solicita conversion del DAC
76   static uint8_t indice_muestra;
77   LPC_TIM3->IR|=(1<<0); //borra el flag de MRO
78   if(analog[1]==1){ //Emision de una sinusoidal
79     LPC_DAC->DACR= ((Amplitud*fun[0][indice_muestra++])/3300) << 6; //conversion
80     if(indice_muestra==0x32){ //Cuando alcanzo las 50 muestras reseteo el indice
81       indice_muestra=0x00;
82     }
83   }
84   else if(analog[2]==1){ //Emision de una senal triangular
85     LPC_DAC->DACR= ((Amplitud*fun[1][indice_muestra++])/3300) << 6; //conversion
86     if(indice_muestra==0x32){ //Cuando alcanzo las 50 muestras reseteo el indice
87       indice_muestra=0x00;
88     }
89   }
90   else if(analog[3]==1){ //Emision de una diente de sierra
91     LPC_DAC->DACR= ((Amplitud*fun[2][indice_muestra++])/3300) << 6; //conversion
92     if(indice_muestra==0x32){ //Cuando alcanzo las 50 muestras reseteo el indice
93       indice_muestra=0x00;
94     }
95   }
96 }
97 }

```


Capítulo 4

Pruebas y resultados

En este capítulo se pretenden realizar una serie de pruebas con el fin de validar el sistema completo, tanto la aplicación e interfaz web (desarrolladas en los apartados [3.4.2](#) y [3.4.1](#), respectivamente) como la plataforma de prácticas.

Con el fin de realizar las pruebas bajo un escenario realista, estas consistirán en prácticas de laboratorio que se llevan a cabo de manera habitual en los laboratorios y que han sido extraídas en su mayoría de [\[5\]](#). Primero, se realizará una descripción de la plataforma de prácticas montada y, posteriormente, se analizarán los resultados obtenidos en las pruebas mencionadas.

4.1 Plataforma de prácticas

La plataforma de prácticas que se montará en el laboratorio es un sistema que debe permitir la realización de pruebas muy diferentes entre sí y por ello, deberá estar dotada con la mayor cantidad de elementos posibles para dar respuesta a cada práctica de manera satisfactoria, logrando que el resultado sea análogo a si el alumno tuviera el hardware en su hogar. A dicha plataforma se conectarán las señales generadas por el sistema objeto de este proyecto. En la imagen [4.1](#) se muestra la plataforma de prácticas implementada.

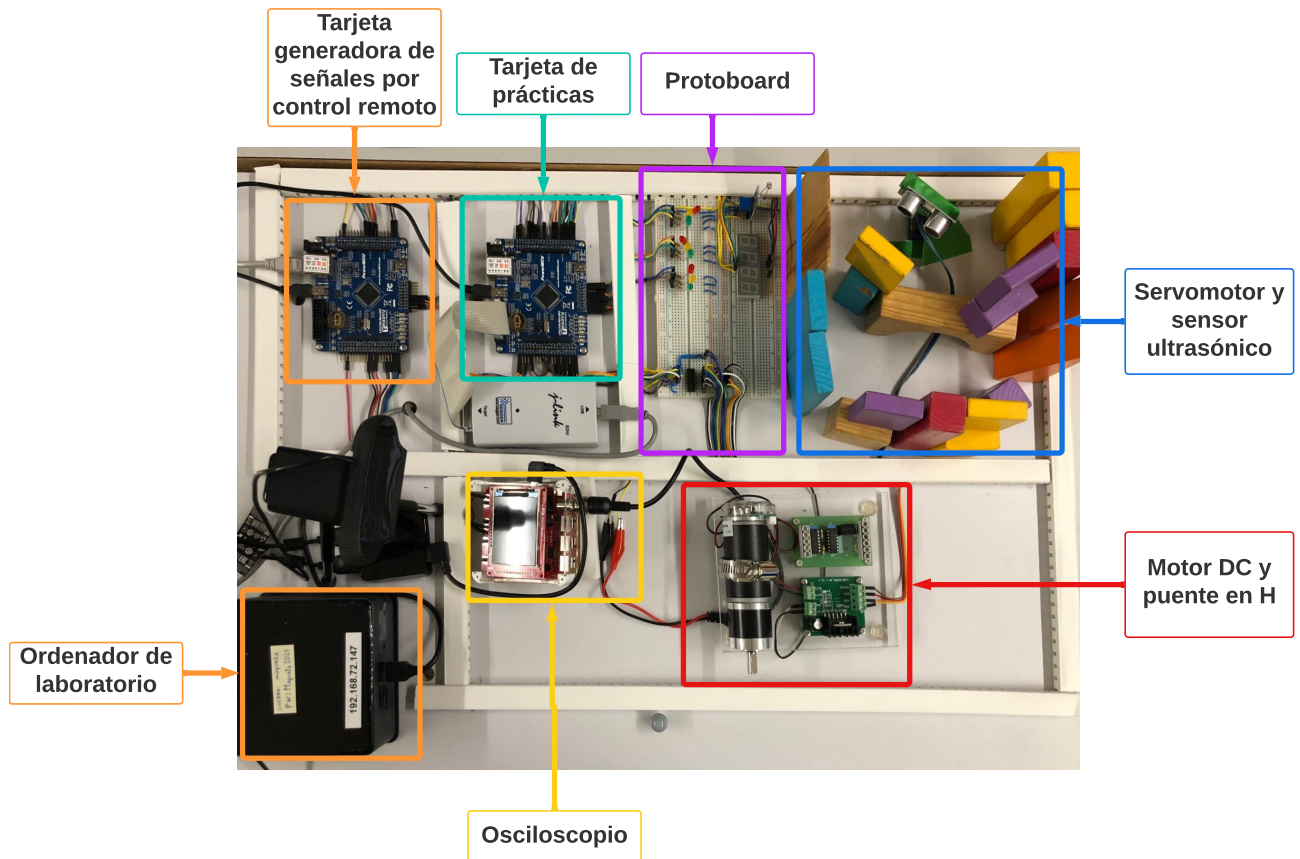


Figura 4.1: Plataforma de prácticas.

La plataforma tiene la capacidad suficiente para incorporar más elementos de prácticas. Actualmente se encuentra prevista con sensores y actuadores para realizar prácticas de sistemas electrónicos digitales de los grados de TICs e industriales.

A continuación, se va a realizar una descripción de cada uno de los bloques marcados en la figura 4.1 y que incorpora actualmente la plataforma de prácticas del laboratorio.

4.1.1 Protoboard

La placa protoboard cuenta con diversos elementos para la realización de múltiples prácticas. A continuación, se describe cada elemento incorporado en la placa protoboard.

4.1.1.1 Sensor de temperatura

Para ciertas aplicaciones, como prácticas donde se requiera el uso del periférico I2C de la tarjeta de prácticas, resulta interesante el uso de un sensor de temperatura.

El sensor que se ha integrado en la protoboard es del modelo BMP180 y mide presión en hPa y temperatura en $^{\circ}\text{C}$. El ratio por muestra del sensor puede ser de hasta 128

muestras por segundo para medidas dinámicas. El esquemático del conexionado del sensor es el mostrado en la figura 4.2, donde las resistencias R_p son de $4,7\text{ k}\Omega$.

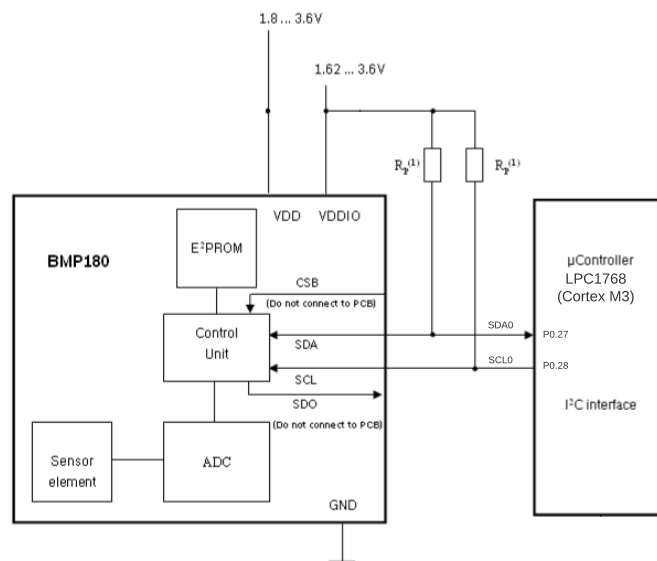


Figura 4.2: Conexionado del sensor de temperatura BMP180.

El cálculo de la temperatura y presión se realiza en pasos de $0,1^{\circ}\text{C}$ y 1Pa , respectivamente. Además, se ha seleccionado este modelo de sensor debido a que el algoritmo para la toma de medidas es aportado por el fabricante como código fuente al adquirir el equipo.

4.1.1.2 Display siete segmentos y LEDs de colores

Con el fin de que los usuarios de la plataforma puedan probar la activación y desactivación de puertos o prácticas relacionadas con temporizadores, como por ejemplo la programación de semáforos, se han integrado tanto un display siete segmentos como nueve leds de colores en la protoboard de la plataforma.

El display siete segmentos es del modelo CA56-12SRWA que es de ánodo común y ha sido elegido principalmente por su tamaño que permite una mejor visualización de los segmentos desde la distancia donde se colocará la cámara. Por otro lado, se han integrado nueve leds de colores siendo tres de color rojo, tres de color amarillo y tres de color verde, pudiendo realizar prácticas como la programación mediante temporizadores de semáforos. En el anexo II se muestra el conexionado de los segmentos del display y de cada uno de los leds.

4.1.2 Multiplexor analógico

Junto al osciloscopio disponible en la plataforma será necesario integrar un multiplexor analógico para poder multiplexar las distintas señales generadas según la práctica que se

esté realizando. Para ello se empleará el chip 74HC4851 que es un multiplexor analógico de ocho canales de entrada y será controlado desde la interfaz de la aplicación web que se describió en el apartado 3.4.1.5. La tabla de verdad del dispositivo se muestra en la tabla 4.1.

Tabla 4.1: Tabla de verdad del multiplexor analógico.

INPUTS				ON
INH	C	B	A	CHANNEL
L	L	L	L	Y0
L	L	L	H	Y1
L	L	H	L	Y2
L	L	H	H	Y3
L	H	L	L	Y4
L	H	L	H	Y5
L	H	H	L	Y6
L	H	H	H	Y7
H	X	X	X	None

Por último, se muestra la tabla 4.2 del conexionado de los canales del multiplexor a los terminales de la tarjeta de prácticas junto con la posible funcionalidad de estos.

Tabla 4.2: Conexionado de los canales a la tarjeta de prácticas.

Casillas interfaz	Canal	Terminal	Principal funcionalidad
000	0	P1.23	PWM1.4
001	1	P3.25	MAT0.0 y PWM1.2
010	2	P3.26	MAT0.1 y PWM1.3
011	3	P0.26	AD0.3 y AOUT
100	4	P0.11	MAT3.1
101	5	P0.4	CAP2.0
110	6	P1.29	MAT0.1 y CAP1.1
111	7	P1.28	MAT0.0 y CAP1.0

4.1.3 Motor DC y puente en H

En la plataforma de prácticas se ha incorporado un motor DC que será controlado mediante un puente en H LMD18200. Dicho puente en H está diseñado específicamente para aplicaciones de control de motores y ha sido incorporado para poder excitar adecuadamente al motor DC. En la figura 4.3 se muestra el motor DC junto con el puente en H.

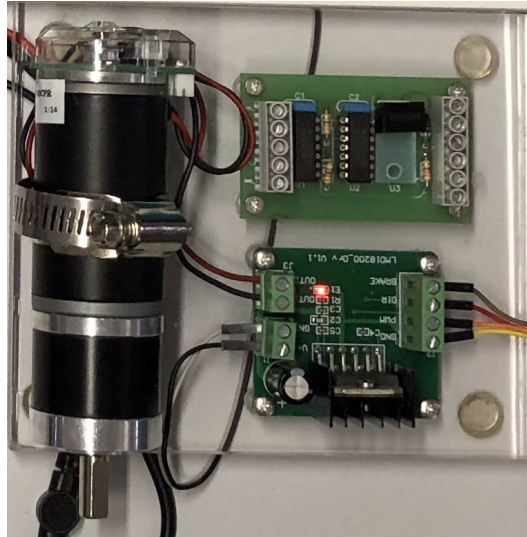


Figura 4.3: Motor DC junto al puente en H.

4.1.4 Servomotor

Los servomotores son sin duda uno de los dispositivos más útiles a la hora de realizar prácticas con sistemas electrónicos digitales, ya que permiten crear toda clase movimientos de una forma controlada.

Para controlar el movimiento del servomotor lo que se ha hecho es crear un sistema de control basado en el ancho de un pulso (PWM). Este pulso que por lo general es de 1,5ms mantiene el servo en la posición centrada. Si el pulso es más corto, por ejemplo 0,9ms el servo gira a la izquierda, si el pulso es mayor, por ejemplo 2,1ms, el servo gira a la derecha. El movimiento del servo, por tanto, es proporcional a la anchura del pulso que se le aplica. En definitiva, el control se realiza mediante una modulación en PWM. Otra particularidad que tiene este pulso es su frecuencia de refresco, que en este caso suele ser generalmente de 50Hz-60Hz, lo que equivale a mandar un pulso periódicamente cada 15ms-20ms, no siendo crítico el tiempo en que la señal de control está a nivel bajo.

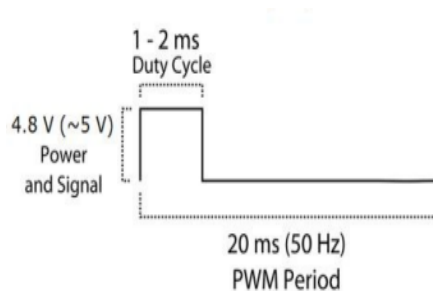
El servomotor que se implementará en la plataforma de prácticas es del modelo SG90, cuyas principales características son:

- Alimentación de 4,8 a 6 V.
- Período de la portadora de 20 ms (50 Hz).
- Posición central con un pulso de 1,5 ms. Giro a izquierdas con un pulso de 1 ms y giro a derechas con un pulso de 2 ms.

En la figura 4.4a se muestra el servomotor SG90 y en la figura 4.4b se ilustra un período de la señal portadora. Ambas figuras han sido extraídas de la hoja de características del fabricante.



(a) Servomotor SG90.



(b) Período de la portadora del servomotor.

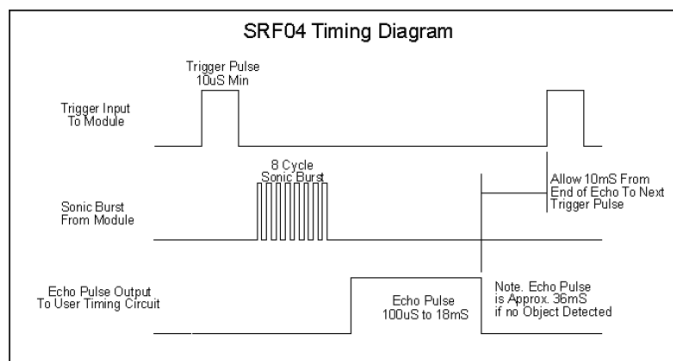
4.1.5 Medidor de distancia SRF04

El módulo SRF04 es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 3 a 300 cm. Su uso es tan sencillo como enviar el pulso de arranque y medir la anchura del pulso de retorno. Este sensor será montado sobre el servomotor descrito en el apartado 4.1.4, para permitir la realización de prácticas donde se integren ambos dispositivos para escanear medidas.

Si analizamos la funcionalidad del medidor de distancia SRF04, se debe activar el dispositivo con un pulso de disparo a nivel alto en el pin 3 (trigger) del SRF04 y después leer la anchura del impulso que nos proporciona en el pin 2 (echo). El pulso de disparo tiene que tener una anchura mínima de 10 μ s. Tras el disparo se inicia la medida. La duración del pulso de salida del eco el tiempo de ida y vuelta de la señal ultrasónica en caso de que encuentre un obstáculo. En caso de que no se produzca ningún eco, porque no se encuentra un objeto, el pulso de eco tiene una longitud aproximada de 36 ms. Para garantizar un correcto funcionamiento del sensor es necesario dejar que transcurran al menos 10 ms entre dos medidas consecutivas. En la Figura 4.5a se muestra el medidor de distancia SRF04 y en la figura 4.5b se muestran las características del sensor y los pines de conexión.



(a) Sensor de distancia SRF04.



(b) Diagrama de tiempos del sensor SRF04.

4.1.6 Osciloscopio

Otro elemento clave para poder verificar la correcta generación de las señales por parte del alumno es poder visualizarlas a través de un osciloscopio. Por tanto, se integrará un osciloscopio modelo DS0138mini el cual permite observar señales con escala de tiempos desde 10 μ s a 500 segundos y escala de voltaje desde 10 mV a 5V. Para las pruebas que se llevarán a cabo este dispositivo es suficiente, puesto que se observarán señales con frecuencia que abarcarán como máximo 15 kHz y como mínimo 1 Hz, siendo su voltaje máximo de 3,3 V.

Además, gracias a los botones con los que cuenta y que han sido implementados en la interfaz web de la aplicación desarrollada, que se describió en 3.4.1.5, se puede variar la escala de tiempos según la aplicación diseñada. En la figura 4.6 se muestra el osciloscopio DS0138mini descrito en este apartado.

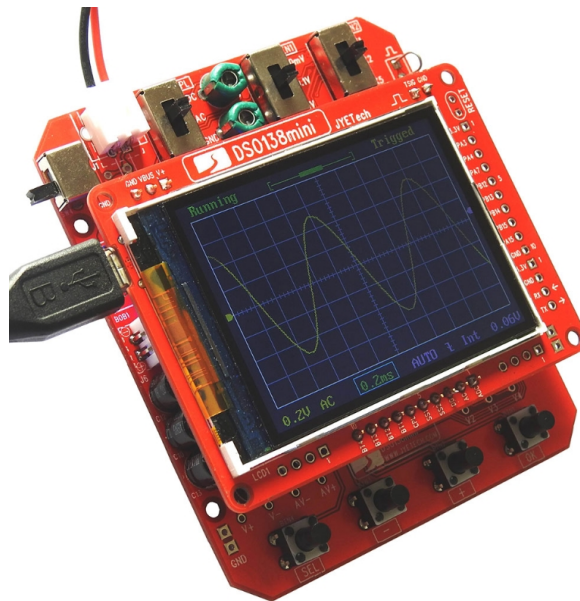


Figura 4.6: Osciloscopio DS0138mini.

4.2 Pruebas prácticas

Una vez descritos los elementos montados en la tarjeta de prácticas se procede a realizar diversas pruebas para verificar el sistema completo formado por la aplicación web y la plataforma desarrollados en el proyecto. En los anexos I y II se muestran las tablas que indican el conexionado realizado tanto de entradas como de salidas de la tarjeta de prácticas de la plataforma.

La realización de la pruebas se hará de forma remota accediendo mediante la aplicación AnyDesk al ordenador remoto de la plataforma de prácticas. Una vez realizada la conexión

remota se empleará Keil uVision para la descarga de la práctica en la tarjeta de prácticas y se excitará mediante la generación de señales de la aplicación web desarrollada en el proyecto.

4.2.1 Control y generación de una señal digital mediante puertos E/S

El programa a desarrollar deberá generar una señal digital de 1Hz con un ciclo de trabajo que dependerá de dos entradas, D1 D0. La generación de los tiempos será realizada mediante un algoritmo, es decir, sin hacer uso de los Timers con los que cuenta la tarjeta de prácticas. El esquema de los terminales empleados se muestra en la Figura 4.7. Además, como señales de entrada al programa se tomarán las señales “a”, “b” y “m” generadas desde la aplicación web.

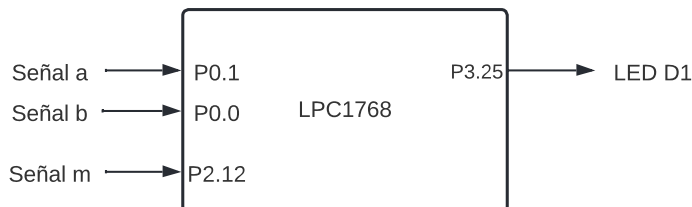


Figura 4.7: Terminales empleados para el programa desarrollado.

El modo de funcionamiento del sistema es el siguiente:

- Inicialmente el programa generará una señal digital de 1Hz con un ciclo de trabajo del 20 % por la salida P3.25.
- Cada vez que se produzca un flanco de bajada en el puerto P2.12, que está conectado a la señal “m” de la aplicación web desarrollada, se leerá el valor digital generado desde las señales “b” y “a” conectadas a los terminales P0.1 y P0.0, respectivamente. El código digital leído desde dichos terminales determina el nuevo ciclo de trabajo, que corresponderá con el indicado en la tabla 4.3.

Tabla 4.3: Código digital y ciclo de trabajo de la señal emitida en P3.25.

b	a	Ciclo de trabajo
0	0	20 %
0	1	40 %
1	0	60 %
1	1	80 %

Mediante la simulación del sistema se ha podido verificar el correcto funcionamiento del programa antes de su verificación sobre la tarjeta de prácticas de la plataforma. Cada vez que se modifica el código digital y se produce un flanco de bajada en el terminal P2.12 la salida generada cambia su ciclo de trabajo, tal y como podemos observar en la figura 4.8. Además, se comprueba que debido al uso de un algoritmo para la generación de tiempos muertos poco eficiente, no se logra la frecuencia deseada para la portadora, siendo su frecuencia de 0,83Hz frente a la frecuencia de 1 Hz deseada.

En la captura se muestra la señal portadora en color rojo (terminal P3.25), el terminal P2.12 en color negro y los terminales P0.1 y P0.0 en color azul. La escala del eje abscisas empleada es de 0,2 seg/div.

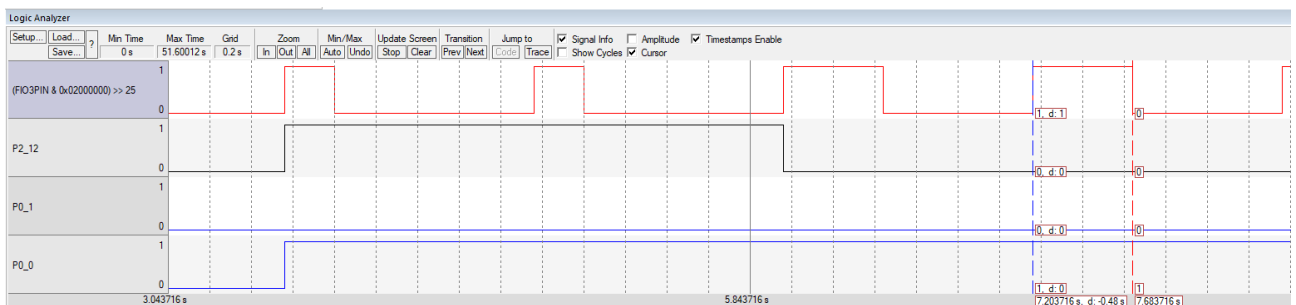


Figura 4.8: Simulación para un ciclo de trabajo del 40 %.

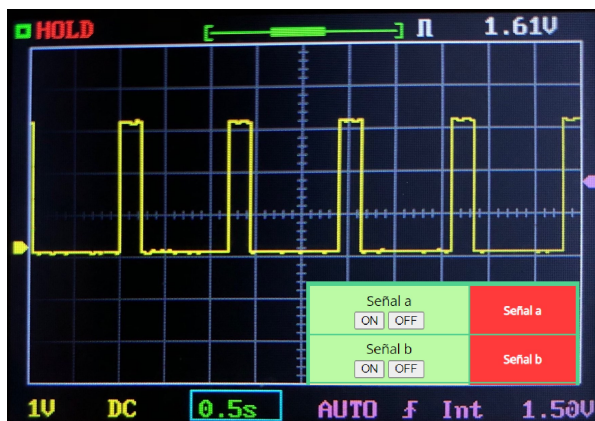
Cuando P0.1 y P0.0 son "01" el ciclo de trabajo es del 40 %, es decir, se mantiene a nivel alto 0,48 segundos del período de 1,2 segundos. Se puede observar en la figura 4.8 que hasta que no se produce un flanco de bajada en el terminal P2.12, el ciclo de trabajo de la portadora no se modifica.

4.2.1.1 Verificación en tarjeta de prácticas de la plataforma

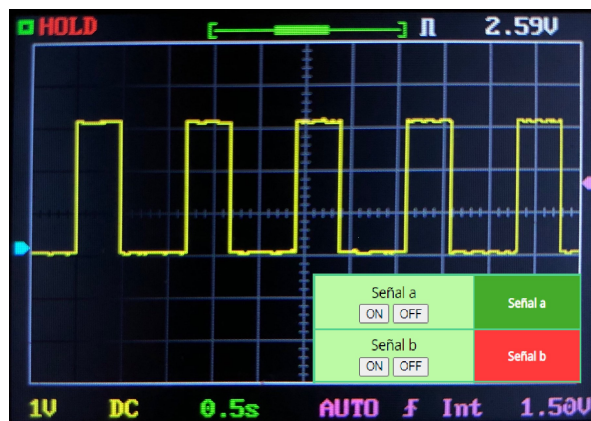
Tras realizar la verificación del programa mediante simulación, se procede a su verificación. Las pruebas realizadas sobre tarjeta, al igual que en simulación, consistirán en modificar las señales de entrada del sistema desde la aplicación web desarrollada en el proyecto para observar la respuesta del programa. En las figuras 4.9, se observa la respuesta esperada según el valor del código digital introducido en P0.0 y P0.1 mediante las señales "a" y "b" generadas por el sistema y mostradas junto a la señal generada en las figuras. Además, al igual que en la simulación, debido a que no se ha empleado un temporizador para la generación de la señal digital la frecuencia obtenida no es de 1Hz, sino de 0,83Hz (un período de 1,2 segundos).

La escala utilizada en el eje abscisas del osciloscopio es 0,5 seg/div y en el eje ordenadas es de 1V/div. Si se analiza uno de los casos mostrados, por ejemplo el caso de la figura 4.9b, se observa que la señal se encuentra a nivel alto 0,48 segundos de 1,2 segundos (ciclo

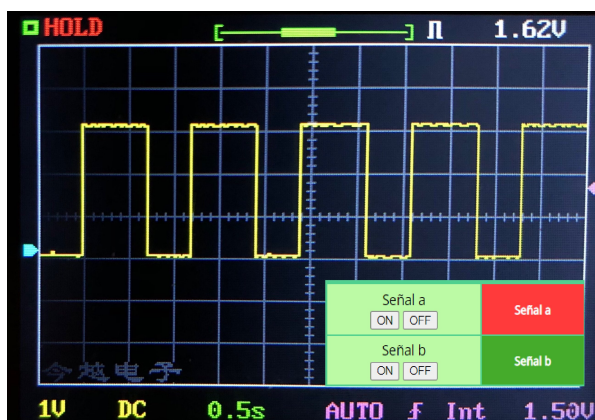
de trabajo al 40%) que dura un período de la portadora y que el código digital de las señales “b” y “a” es “01”, siendo el color rojo en la señal un nivel bajo y el color verde un nivel alto.



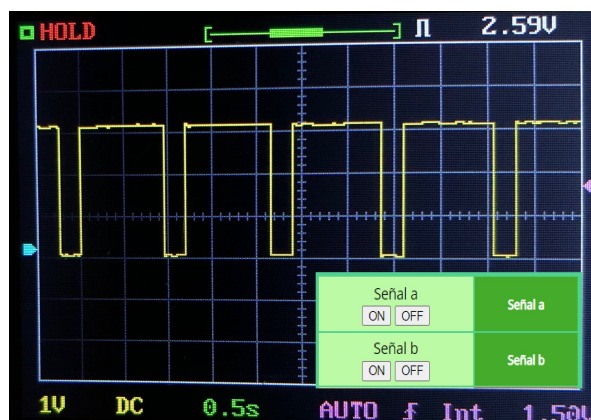
(a) Ciclo de trabajo del 20%.



(b) Ciclo de trabajo del 40%.



(c) Ciclo de trabajo del 60%.



(d) Ciclo de trabajo del 80%.

Figura 4.9: Pruebas realizadas.

4.2.2 Control y generación de una señal digital mediante Timers

El programa a desarrollar en esta práctica generará una señal cuadrada de 2kHz por el puerto P1.29 con un ciclo de trabajo del 20% inicialmente. Además, según el código digital leído desde los puertos P0.0 y P0.1, conectados a las señales a y b, respectivamente, se modificará el ciclo de trabajo de la portadora. En la figura 4.10 se muestra un esquema de las señales E/S del programa.

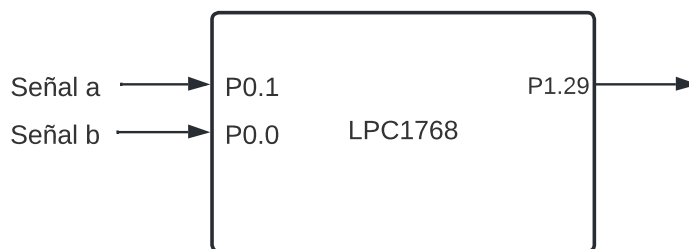


Figura 4.10: Terminales empleados en el programa.

Para la generación de la señal cuadrada, se configura el Timer 1 para que interrumpa tanto cuando el TC (contador del Timer) alcance el valor Match0 como el Match1. Además, cada vez que se interrumpa por Match0 se debe resetear el TC. A continuación, se muestra una figura que esquematiza el funcionamiento:

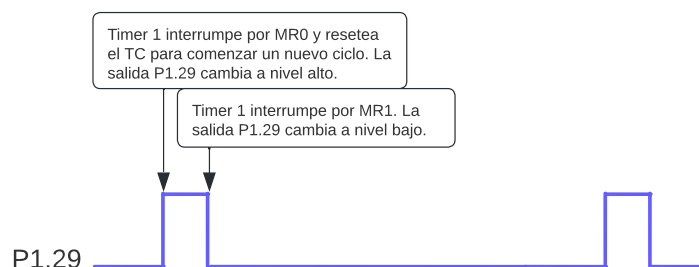


Figura 4.11: Esquema temporal del programa.

Además, los códigos digitales que modifican el ciclo de trabajo de la portadora son los mostrados en la tabla 4.4.

Tabla 4.4: Código digital y ciclo de trabajo de la señal emitida en P1.29.

b	a	Ciclo de trabajo
0	0	20 %
0	1	40 %
1	0	60 %
1	1	80 %

A continuación, se mostrarán los resultados obtenidos tras simular el programa desarrollado. Para su simulación se han modificado los terminales de entrada P0.1 y P0.0 desde la ventana de depuración y así, se ha ido variando el ciclo de trabajo de la señal portadora. En la figura 4.12, se muestra la señal cuadrada generada en color rojo y además, los terminales correspondientes al puerto 1 de la tarjeta junto con una variable denominada *estado* cuyo valor es el resultado de la concatenación de los bits P0.1 y P0.0. La escala empleada en este caso para el eje abscisas es de 0,1 ms/div.

En la figura se observa la portadora cuando el ciclo de trabajo es del 20 %, puesto que P0.1 y P0.0 son "00". Además, se comprueba que la señal generada tiene un período de 0,5 ms equivalente a una frecuencia de 2 kHz y que, como cabía esperar, el tiempo a nivel alto es de 0,1 ms.

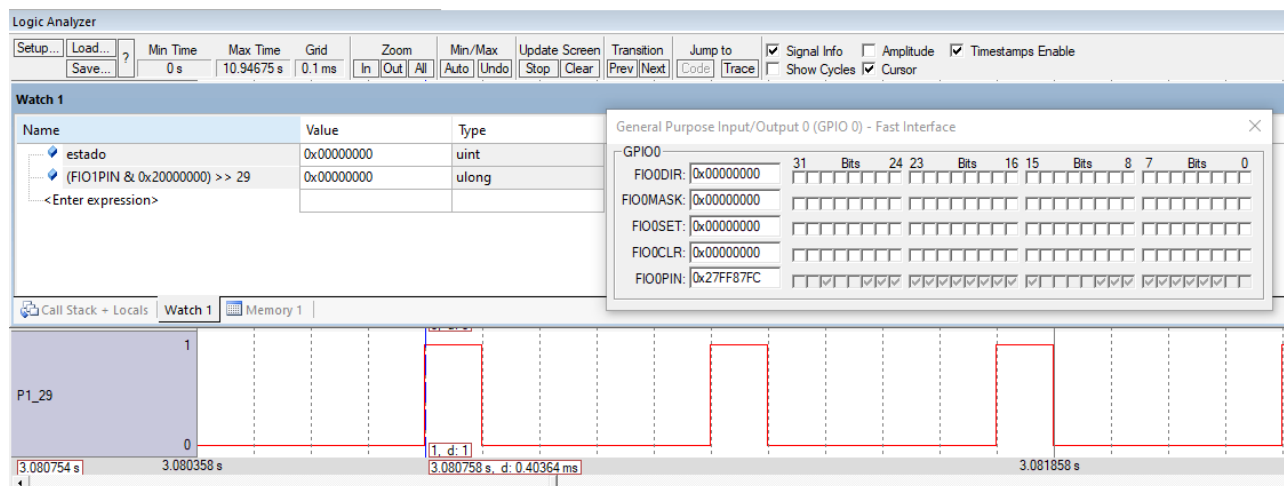
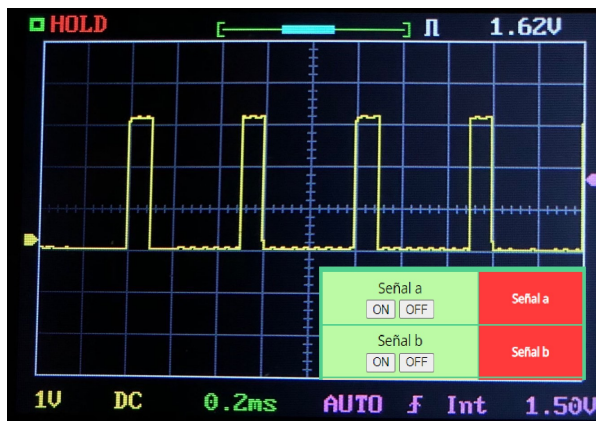


Figura 4.12: Simulación para un ciclo de trabajo del 20 %.

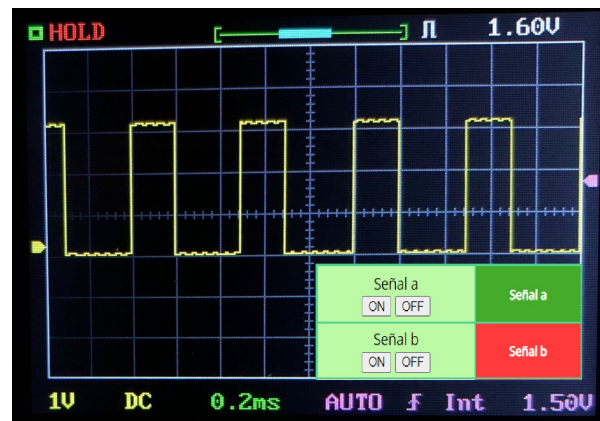
4.2.2.1 Verificación en tarjeta de prácticas de la plataforma

Tras realizar la simulación del programa, se procede a su verificación en la tarjeta de prácticas implementada en la plataforma del laboratorio. En las figuras 4.13, se observa la respuesta esperada según el valor del código digital introducido en P0.0 y P0.1 mediante las señales "a" y "b" de la aplicación web, indicado en la tabla 4.4.

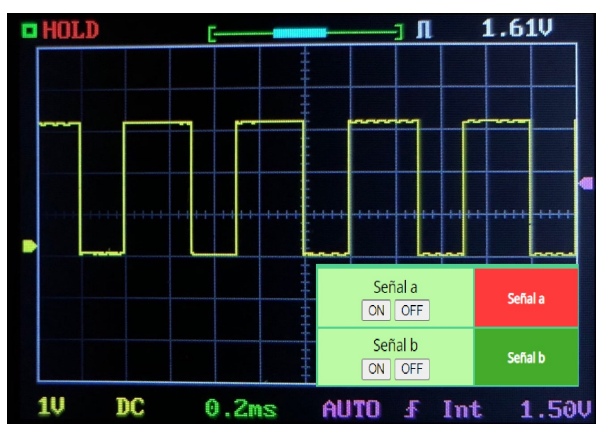
La escala utilizada en el eje abscisas del osciloscopio es 0,2 ms/div y en el eje ordenadas es de 1V/div, modificadas desde el apartado de manejo del osciloscopio de la interfaz. Por ejemplo, en la figura 4.13a se muestra el código digital introducido desde la interfaz web que en este caso es "00" y la señal cuadrada con un ciclo de trabajo del 20 %, por tanto, el tiempo en alto es de 0,1 ms de los 0,5 ms que dura un período de la señal.



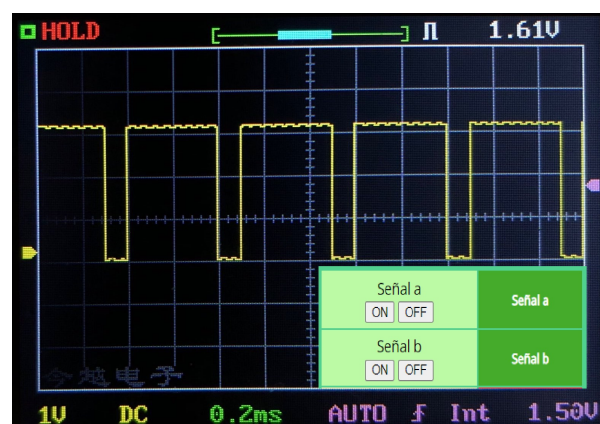
(a) Ciclo de trabajo del 20 %.



(b) Ciclo de trabajo del 40 %.



(c) Ciclo de trabajo del 60 %.



(d) Ciclo de trabajo del 80 %.

Figura 4.13: Pruebas realizadas.

Finalmente, se puede observar que, en comparación con la práctica del apartado 4.2.1, sí se logra una señal portadora con la frecuencia requerida en el enunciado, siendo en este caso de 2kHz. Esto es gracias al uso de temporizadores frente al algoritmo que incrementaba una variable que depende del tiempo de ejecución del programa.

4.2.3 Manejo del convertor digital-analógico

Se pretende generar en la salida AOUT (pin P0.26) de la tarjeta LPC1768, una señal sinusoidal cuya frecuencia varíe entre 1kHz y 15kHz, que aproveche al máximo el margen de tensiones de salida.

La frecuencia podrá ser modificada a través de la interfaz mediante la señal “m” (terminal P2.12 de la tarjeta de prácticas). Esta pulsación provocará la interrupción EINT2 y la frecuencia será proporcional al valor digital de los 4 bits de las señales “f” a “i” (terminales P2.0 a P2.3), de tal forma que para el valor “i”..“f” = “0001” la frecuencia

debe ser 1kHz y para el valor "i"..."f" = "1111" debe ser de 15kHz. Para el valor "i"..."f" = "0000" la frecuencia será de 100 Hz.

La forma de onda se almacenará en un array de 50 elementos y para el cálculo del seno se usará la función $\sin()$, que admite como parámetro el ángulo en radianes y devuelve un valor entre -1 y 1. Por tanto, en un ciclo habrá 50 muestras (desde 0 a 49) con valores entre 0 y 1023.

Las 50 muestras de la forma de onda almacenadas en el array se deben sacar por el DAC, que se encuentra en el terminal P0.26 de la tarjeta LPC1768, en un tiempo que depende de la frecuencia seleccionada para la señal. Para ello, se hará uso de un temporizador de manera que cada vez que se ejecute su interrupción permita sacar por el DAC una nueva muestra. El temporizador empleado para ello será el Timer 0.

El diagrama que indica los terminales y su conexionado a las señales de la aplicación web se muestra en el figura 4.14.

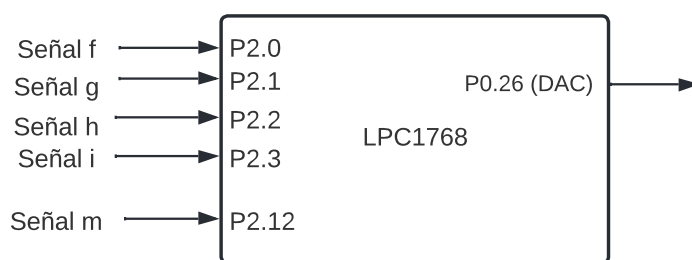


Figura 4.14: Terminales empleados en el programa.

Al igual que en las pruebas realizadas anteriormente, se procede a verificar el programa mediante simulación antes de trabajar sobre la tarjeta de prácticas. Para ello, se varían las entradas P2.3...P2.0 desde la ventana de depuración obteniéndose los resultados mostrados a continuación.

En las capturas se muestra la señal sinusoidal generada en color rojo, los terminales del puerto 2 de la tarjeta y la variable *Frecuenciakhz* que es resultado de la concatenación de los bits correspondientes a P2.3...P2.0.

Para la primera de las pruebas, donde la señal tiene una frecuencia de 100 Hz, la escala del eje de tiempos es de 5 ms/ div. Observando la figura 4.15 se comprueba que cuando todos los bits de entrada son P2.3...P2.0 = "0000", se obtiene una onda sinusoidal con frecuencia de 100 Hz. Esto se comprueba observando un período de la señal, donde se obtiene que un ciclo dura 10 ms.

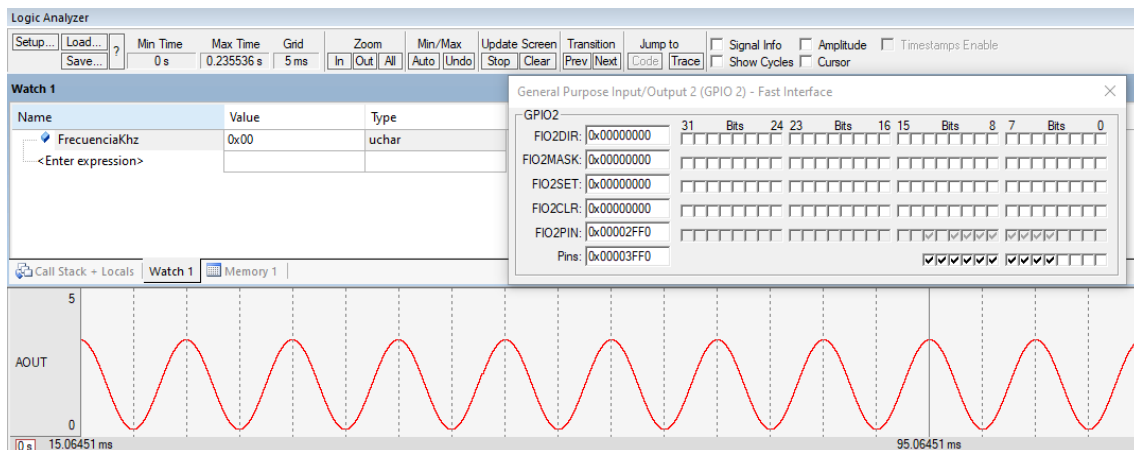


Figura 4.15: Simulación cuando los terminales P2.3...P2.0 = "0000"

Una vez simulado el caso inicial, se procede a simular los casos donde la frecuencia a obtener es de 1 kHz y 15 kHz. En la figura 4.16 se observa que el período de la señal es de 1 ms, es decir, se logra una onda sinusoidal de 1 kHz cuando las entradas P2.3...P2.0 = "0001".

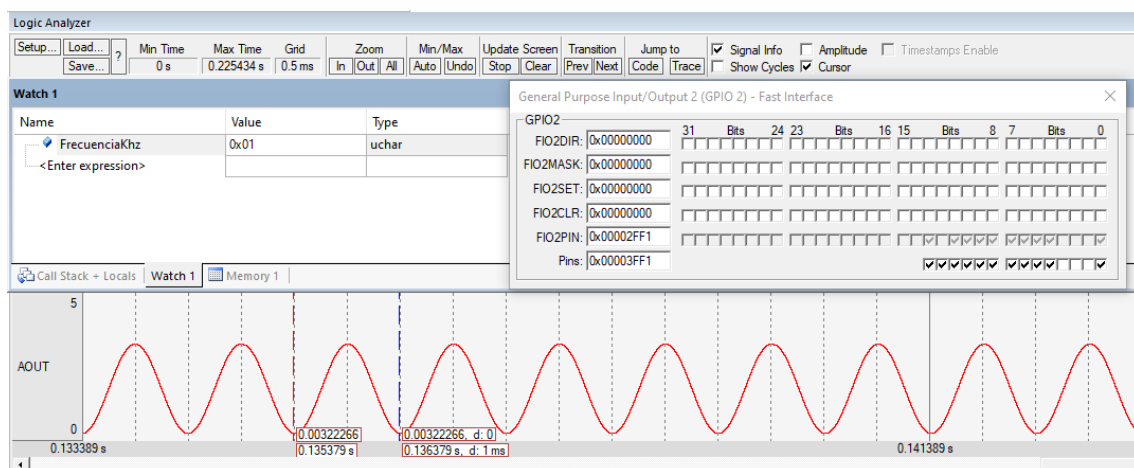


Figura 4.16: Simulación cuando los terminales P2.3...P2.0 = "0001"

Por último, se simula el caso donde se desea generar una señal sinusoidal trabajando a la máxima frecuencia indicada por el enunciado de la práctica, es decir, 15 kHz. Para ello, los terminales de entrada P2.3...P2.0 = "1111". En la figura 4.17 se muestra la señal generada con un período de 66,7 us.

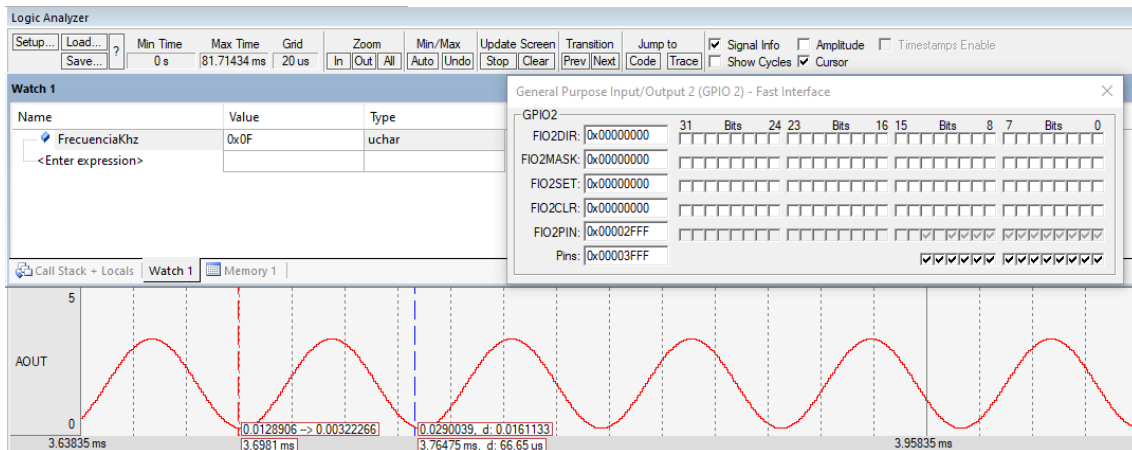


Figura 4.17: Simulación cuando los terminales P2.3...P2.0 = "1111"

4.2.3.1 Verificación en tarjeta de prácticas de la plataforma

Una vez comprobado el correcto funcionamiento del programa en simulación, se procede a realizar las pruebas sobre la tarjeta de prácticas. Se recuerda que las señales "i"... "f" de la interfaz desarrollada se conectaron a los pines P2.3...P2.0, respectivamente. Además, cuando la señal se muestra en color rojo en la interfaz corresponde con un nivel bajo en el terminal y en color verde con un nivel alto.

Para la primera de las pruebas se modifica desde la interfaz web la escala de los ejes, empleando 5 ms/div (eje abscisas) y 1 voltio/div en el eje de ordenadas. Tal y como se ha mencionado con anterioridad, cuando se introduce desde la interfaz de la aplicación en las señales "i"... "f" el código digital "0000" la señal generada tiene una frecuencia de 100 Hz. En la figura 4.18 se puede comprobar que el período de la señal es de 10 ms.

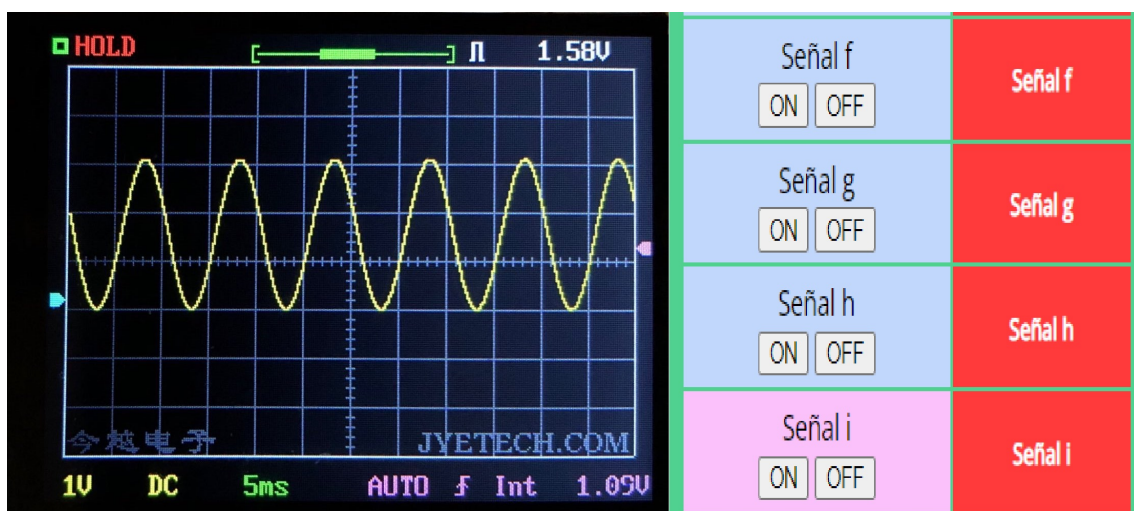


Figura 4.18: Señal sinusoidal con frecuencia de 100 Hz (i...f = "0000").

Para la segunda prueba se modificó desde la interfaz el código de las señales $i...f = "0001"$, obteniéndose una sinusoidal con frecuencia de 1kHz, como se muestra en la figura 4.19. Además, se adaptó la escala desde la aplicación web para el eje de tiempos a 0,2 ms/div y 1 voltio/div en el eje de ordenadas.

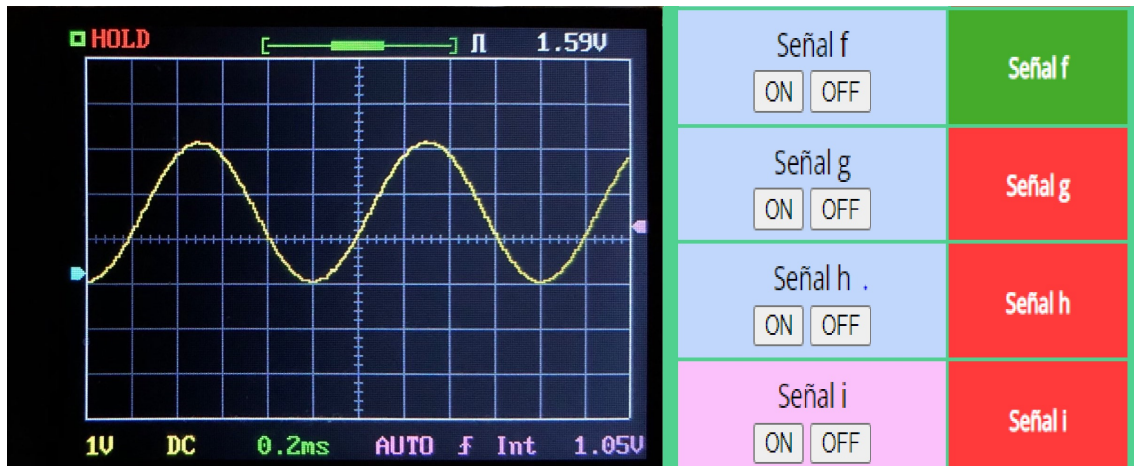


Figura 4.19: Señal sinusoidal con frecuencia de 1 kHz ($i...f = "0001"$).

Por último, se generó una señal sinusoidal con frecuencia 15 kHz (figura 4.20), caso donde $i...f = "1111"$. Las escalas para los ejes fueron modificadas desde la interfaz a 0,1 ms/div en el eje abscisas y 1 voltio/div en el eje ordenadas. Al igual que en las pruebas anteriores, se logran los mismos resultados que en simulación generando una señal sinusoidal de 15kHz.

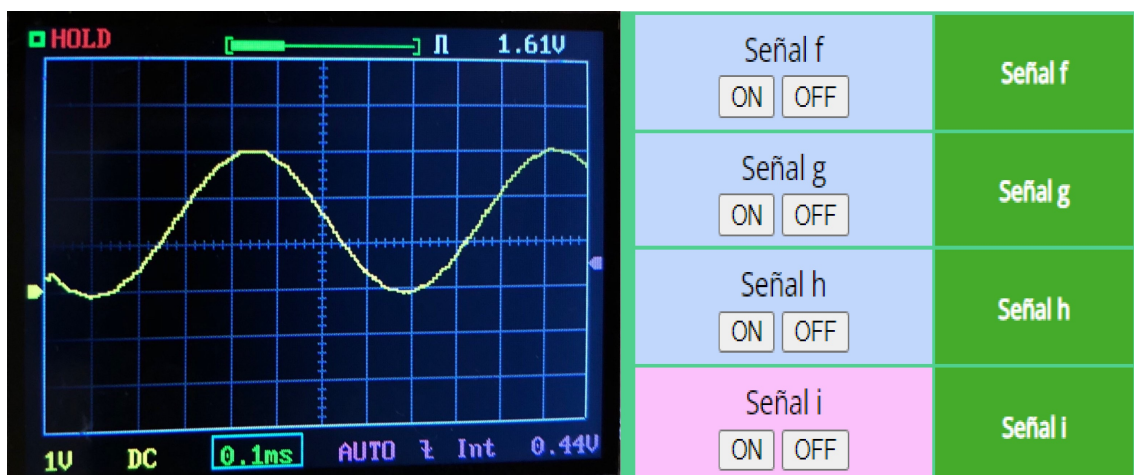


Figura 4.20: Señal sinusoidal con frecuencia de 15 kHz ($i...f = "1111"$).

4.2.4 Diseño de un sistema para un sonar ultrasónico

Se propone diseñar un sistema empujado basado en la tarjeta de laboratorio (microcontrolador LPC1768) que implemente un sonar ultrasónico capaz de extraer las medidas de distancia a los objetos más próximos, en un plano y sobre un entorno de 180° , con posibilidad de ser controlado de forma manual a través de las entradas P2.11 y P2.12. El sistema, además, tendrá la posibilidad de detectar obstáculos permitiendo configurar un umbral de detección de manera que, ante la presencia de un obstáculo, emita a través de un pequeño altavoz un tono cuya frecuencia sea proporcional a la distancia de detección. El diagrama de bloques del sistema se representa en la Figura 4.21.

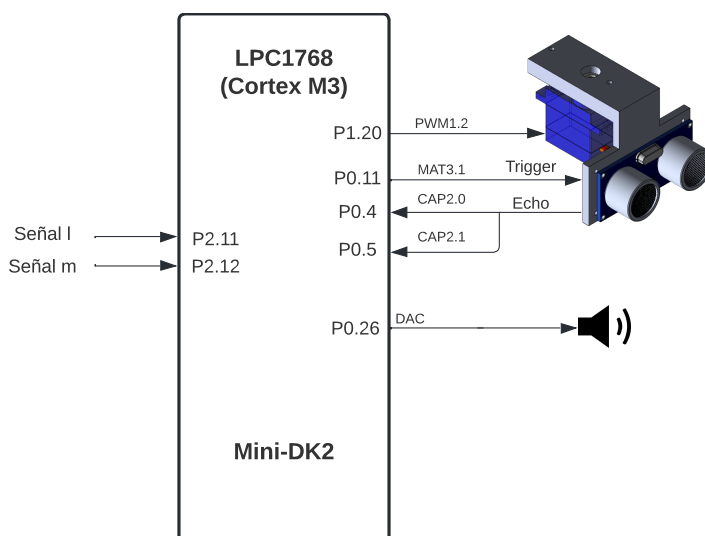


Figura 4.21: Diagrama de bloques del s3nar ultras3nico.

Las especificaciones de funcionamiento del sistema son las siguientes:

- Se controlará la posición del servo mediante las entradas P2.11 y P2.12 (conectados a las señales “l” y “m” de la aplicación desarrollada, respectivamente) de manera que su pulsación desde la interfaz provoque el giro de su posición en 10° a izquierdas si se pulsa la señal “m” (terminal P2.12 de la tarjeta) y 10° a derechas al pulsar la señal “l” (terminal P2.11) mediante las interrupciones EINT2 y EINT1, respectivamente.
- Al inicio del programa se deberá emitir una señal sinusoidal con frecuencia de 500 Hz. Además, si la medida de la distancia es inferior a 3 cm o superior a 15 cm la señal sinusoidal generada también deberá tener un frecuencia de 500 Hz.

Dado que para la generación de la sinusoidal es necesario que el sonar tome medidas, para verificar el correcto funcionamiento del sistema solo ha podido realizarse directamente sobre la tarjeta de prácticas.

4.2.4.1 Verificación en tarjeta de prácticas de la plataforma

Al cargar el programa en la tarjeta de prácticas el servo se posiciona en su posición de inicio y se comienza la toma de medidas por parte del sonar. Además, como se ha mencionado anteriormente, se emite una sinusoidal de frecuencia 500 Hz como puede observarse en la figura 4.22. La escala se modifica desde la interfaz web a 1 ms/div para el eje de abscisas y de 1 voltio/div para el eje de ordenadas.

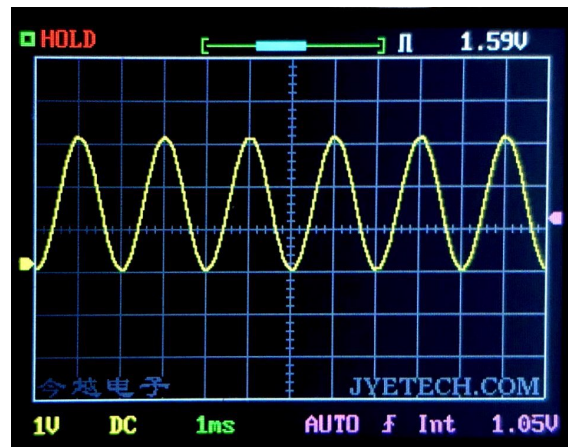


Figura 4.22: Señal sinusoidal generada al inicio del programa.

Tras comprobar esto, se posiciona el servomotor haciendo uso de la señal “m” (terminal P2.12) para hacerlo girar a la izquierda de tal manera que la distancia entre el sonar y la primera pieza a medir sea lo más cercano al límite inferior de 3 cm, pero sin sobrepasarlo. En la figura 4.23 se muestra el servomotor posicionado en esta posición.

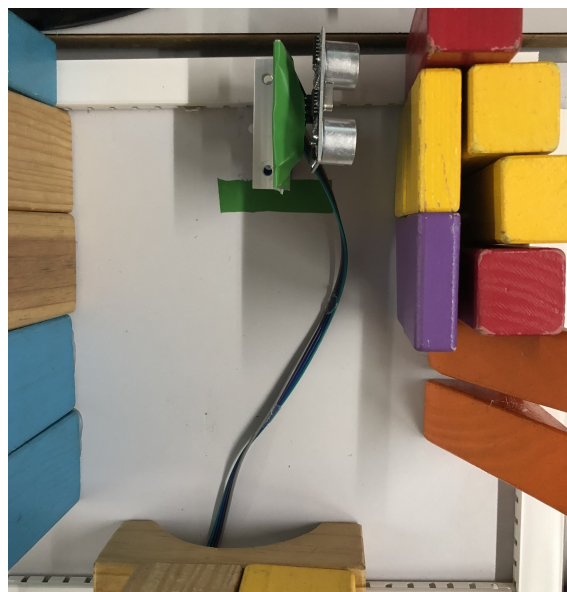


Figura 4.23: Servomotor posicionado a 3cm del obstáculo.

Tras esto, haciendo uso de la ventana de watch, se puede comprobar la medida de distancia tomada por el sonar, que se muestra en la variable "*distancia*". Dicha medida se convierte en frecuencia en kHz, mostrándose en la variable "*FrecuenciaKhz*". Es decir, para el ejemplo de la figura 4.24 donde se mide una distancia de 3,009 cm la frecuencia de la señal sinusoidal será de 3kHz.

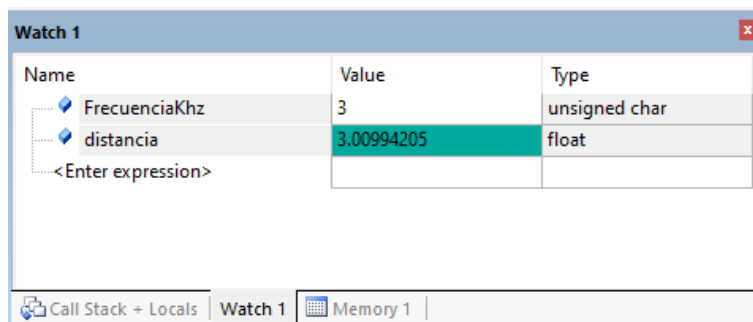


Figura 4.24: Medida de distancia para un obstáculo a 3 cm.

Dado que la medida tomada se encuentra dentro de los límites establecidos por el enunciado de la práctica, la onda sinusoidal generada tiene una frecuencia de 3 kHz como se muestra en la figura 4.25. La escala empleada para el eje de tiempos en este caso es de 0,1 ms/div y para el eje de ordenadas de 1 voltio/div.

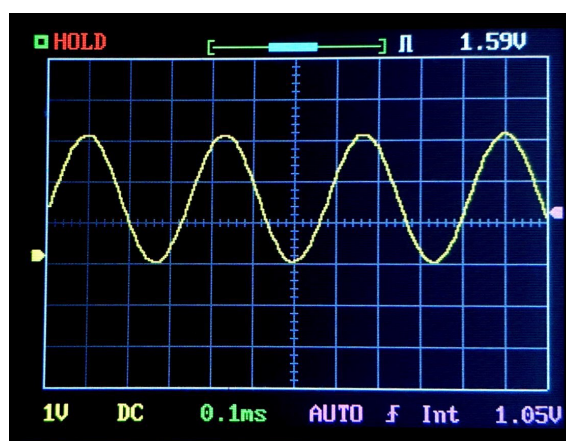


Figura 4.25: Señal sinusoidal generada con una distancia al obstáculo de 3 cm.

La siguiente medida se toma a una distancia posicionada en el medio del rango establecido. Para ello se mueve el servomotor hasta conseguir una distancia entre la pieza y el sonar de alrededor de 7 cm, como se muestra en la figura 4.26.



Figura 4.26: Servomotor posicionado a 7,4cm del obstáculo.

Por otro lado, en la figura 4.27 se muestra la medida tomada por el sonar en la variable "*distancia*" y su frecuencia proporcional en kHz en la variable "*FrecuenciaKhz*". Como puede observarse, la frecuencia resultante es el entero más próximo por debajo de la medida tomada.

Name	Value	Type
FrecuenciaKhz	7	unsigned char
distancia	7.48261833	float
<Enter expression>		

Figura 4.27: Medida de distancia para un obstáculo a 7,4cm.

La señal sinusoidal generada se muestra en la figura 4.28 cuya frecuencia es de 7 kHz. La escala de los ejes del osciloscopio se adapta desde la interfaz web a 0,1ms/div en el eje abscisas es y 1 voltio/div en el eje de ordenadas.

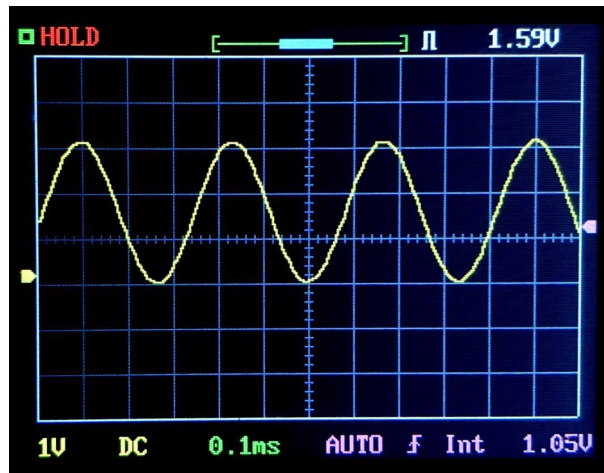


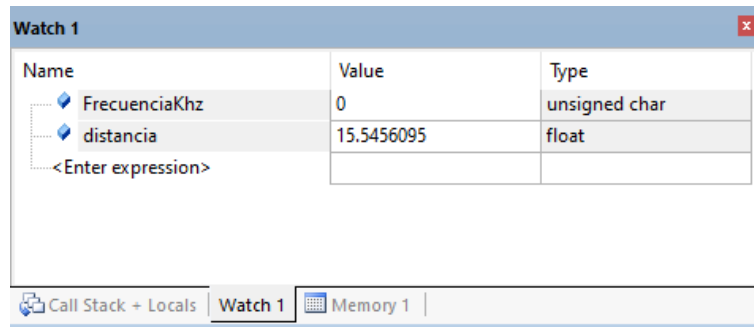
Figura 4.28: Señal sinusoidal generada con una distancia al obstáculo de 7,4 cm.

Por último, se toma una medida que supere el límite marcado por el enunciado de la práctica. Para ello, se posiciona el servomotor girando a la derecha mediante múltiples pulsaciones de la señal I (terminal P2.11) hasta lograr una medida mayor a 15 cm . En la figura 4.29 se muestra el servomotor posicionado en la distancia requerida al obstáculo.



Figura 4.29: Servomotor posicionado a una distancia superior a 15cm del obstáculo.

En la ventana de watch mostrada en 4.30 se observa la medida tomada y como la variable "*FrecuenciaKhz*" se pone a cero puesto que en este caso no se toma en cuenta esta variable.



Name	Value	Type
FrecuenciaKhz	0	unsigned char
distancia	15.5456095	float
<Enter expression>		

Figura 4.30: Medida de distancia para un obstáculo a 15 cm.

Finalmente, la señal sinusoidal generada, que se muestra en la figura 4.31, tiene una frecuencia de 500 Hz debido a que la distancia medida supera el límite indicado por el enunciado de la práctica. La escala de tiempos empleada en este caso es de 1 ms/div y para el eje de ordenadas de 1 voltio/div.

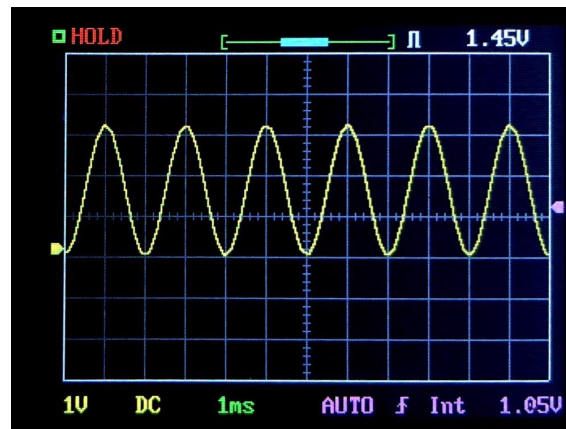


Figura 4.31: Señal sinusoidal generada con una distancia al obstáculo superior a 15 cm.

Capítulo 5

Conclusiones y líneas futuras

En este apartado se resumen las conclusiones obtenidas en cada una de las etapas del proyecto y se proponen futuras líneas de investigación que se deriven del trabajo.

5.1 Conclusiones

Al comienzo del trabajo se establecieron unos objetivos que fueron definidos en el apartado 1.3. Una vez se ha concluido con el proyecto se procede a revisar las principales etapas del mismo para comprobar si se han cumplido dichos objetivos. A continuación se citan las distintas etapas y los logros obtenidos en cada una de ellas.

- En la primera etapa se ha realizado un estudio e investigación acerca de las diversas técnicas existentes para crear y diseñar un servidor web en microcontroladores de ARM. También, se analizaron los posibles métodos de comunicación cliente-servidor a implementar.

Gracias al manual aportado por ARM [2] para desarrollar aplicaciones web en microcontroladores basados en su arquitectura fue sencillo entender las herramientas y junto a los ejemplos del manual comenzar a entender el funcionamiento de un servidor web basado en microcontrolador.

- En la segunda etapa, una vez seleccionado el método para desarrollar la aplicación web a partir de la librería RL-ARM que se describió 3.3, se comenzó con el desarrollo de la aplicación. Primero, se implementó el frontend, tal y como fue descrito en el apartado 3.4.1, empleando HTML y CSS. Para comprender el funcionamiento de dichos lenguajes de marcado se emplearon diversas fuentes como [4] o [6].
- En la tercera etapa comenzó el desarrollo del backend de la aplicación, tal y como se describió en el apartado 3.4.2. Durante este proceso se encontraron diversos problemas relacionados con el método seleccionado previamente para la comunicación

cliente-servidor puesto que no era lo suficientemente eficiente y esto provocaba caídas continuas del servidor. Sin embargo, esta situación pudo solucionarse empleando dos métodos de comunicación de manera paralela solucionándose así los problemas de eficiencia.

También se encontraron ciertas incompatibilidades en los pines elegidos para las señales conectadas a los pulsadores de la tarjeta de prácticas, puesto que dichos terminales tienen resistencias de pull-up. Para solucionarlo se inicializaron las señales conectadas a nivel alto, solventando el problema.

- En la cuarta etapa, con la aplicación web desarrollada, se procedió al montaje de la plataforma de prácticas. Para facilitar el conexionado entre el sistema de generación de señales objeto del proyecto y la tarjeta de prácticas de la plataforma se realizaron una serie de tablas para la asignación de pines, como se muestra en los Anexos. También se conectó la tarjeta de prácticas a los diversos elementos de las prácticas descrito en el apartado 4.1.
- Tras esto, comenzó la realización de pruebas donde se ejecutaron diversas prácticas con el fin de verificar el funcionamiento del sistema completo, formado por la aplicación web y la plataforma de prácticas. Los resultados obtenidos fueron muy buenos cumpliéndose los objetivos marcados por el enunciado en todas ellas.

En conclusión, se ha desarrollado e implementado un sistema completo de prácticas en acceso remoto para que los estudiantes de futuros años puedan realizar sus pruebas sin la necesidad de adquirir todo el hardware necesario para ello.

Además, se ha logrado desarrollar una aplicación web cuya interfaz es lo suficientemente intuitiva (pero a la vez compleja en su contenido) para que cualquier estudiante sin experiencia en microcontroladores sepa manejarse en ella. También, se ha montando una plataforma de prácticas con un gran número de elementos que permiten la realización de múltiples prácticas de laboratorio.

5.2 Líneas futuras

La experiencia obtenida a lo largo del proyecto junto al análisis de los resultados y pruebas realizadas han permitido enfocarse en futuras líneas de investigación y ciertas mejoras que pueden realizarse en el sistema de prácticas remoto.

Una de estas mejoras sería incorporar más elementos a la plataforma de prácticas como podría ser el display LCD a la tarjeta de prácticas o diversos sensores. Sin embargo, sería necesario ampliar la plataforma de prácticas puesto que el espacio restante actualmente es bastante pequeño. También, habría que reestructurar los terminales que pueden usarse

de la tarjeta de prácticas dado que el uso del display LCD consume muchos terminales de la tarjeta.

Otra posible mejora sería el uso de lenguaje Java para el desarrollo del front-end de la aplicación, puesto que haría más simple y sobre todo, más eficiente el uso de dicha interfaz, respecto a su implementación en HTML.

Capítulo 6

Presupuesto

En la siguiente tabla se muestra una estimación del presupuesto del proyecto:

Descripción	Cantidad	Unidad	Precio Unitario [€/Ud]	Precio total [€]
Hardware				
Ordenador				
Ordenador con windows 10, RAM 64 Gb, Disco duro 256 SSD, 1TB HDD.	1	Ud.	500	500
Tarjeta Mini-DK2				
LPC1768-Mini-DK2 basado en los procesadores NXP de la serie LPC1700 (Cortex-M3).	2	Uds.	30	60
Cable JTAG				
Cable de programación y depuración para microcontroladores basados en arquitectura ARM. Comercializado por Segger.	1	Ud.	70	70
Placa protoboard				
Placa de pruebas con orificios para conectar elementos electrónicos.	1	Ud.	10	10
Servomotor				
Modelo SG90	1	Ud.	2	2
Sensor de distancia ultrasónico				
Modelo SRF04	1	Ud.	2	2
Multiplexor analógico				
Modelo 74HC485	1	Ud.	1	1
Display siete segmentos				
Modelo CA56-12SRWA	1	Ud.	2	2
LEDs de colores				
-	9	Uds.	0,79	7,11
Motor DC				
Modelo GP36E1000CPR	1	Ud.	30	30
Puente en H				
Modelo LMD18200	1	Ud.	10	10
Osciloscopio				
Modelo DSO138Mini	1	Ud.	9	9
Cámaras				
Modelo C922 PRO HD stream webcam	2	Uds.	70	140
Total Hardware				843,11

Descripción	Cantidad	Unidad	Precio Unitario [€/Ud]	Precio total [€]
Software				
Keil uVision5				
Software producido por ARM para la programación de microcontroladores basados en su arquitectura	1	Ud.	5.412	5.412
AnyDesk				
Software para el acceso remoto a equipos	1	Ud.	9,90	9,90
Total Software				5421,9
Personal				
Ingeniero electrónico				
Estudio, diseño y síntesis del proyecto, así como su testado y documentación.	1	Ud.	2.000€/mes	16.000€ (8 meses trabajados)

Por tanto, el presupuesto total del proyecto es de 22.265€.

Bibliografía

- [1] “Um10360 lpc17xx user manual - keil 2010,” Aug 2010. [Online]. Available: https://www.keil.com/dd/docs/datashts/philips/lpc17xx_um.pdf
- [2] “Getting started building applications with rl-arm.” [Online]. Available: https://www.keil.com/product/brochures/rl-arm_gs.pdf
- [3] “Formato de mensajes http.” [Online]. Available: <https://developer.mozilla.org/es/docs/Web/HTTP/Messages>
- [4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol,” Jun 1999. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc2616.html>
- [5] A. Gardel Vicente, C. Mataix Gómez, C. Luna Vázquez, J. L. Lázaro Galilea, J. J. García Domínguez, and J. M. Miguel Jiménez, *Prácticas de laboratorio de Sistemas Electrónicos Digitales*. Universidad de Alcalá, 2018, i.s.b.n.: 978-84-16978-96-0.
- [6] “Patch method for http.” [Online]. Available: <https://datatracker.ietf.org/doc/html/draft-dusseault-http-patch-16>
- [7] “Servomotor sg90 datasheet.” [Online]. Available: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
- [8] “Srf04 datasheet.” [Online]. Available: <https://datasheetspdf.com/pdf/523589/ETC/SRF04/1>
- [9] “Ds0138mini oscilloscope user’s manual.” [Online]. Available: https://jyotech.com/Products/LcdScope/UserManual_138mini.pdf

Anexos

Anexo I : Conexionado de señales con terminales de entrada de la tarjeta de prácticas.

Tabla 6.1: Conexionado de señales con terminales de entrada de la tarjeta de prácticas.

Señal	Terminal de entrada
Señal a	P0.0
Señal b	P0.1
Señal c	P1.16
Señal d	P1.17
Señal e	P1.19
Señal f	P2.0
Señal g	P2.1
Señal h	P2.2
Señal i	P2.3
Señal j	P2.4
Señal k	P2.10
Señal l	P2.11
Señal m	P2.12
Señal n	P2.13

Señal	Terminal de entrada
PWM1.1	P2.5
PWM1.2	P2.6
PWM1.4	P2.7
PWM1.6	P2.8
Señal analógica	P1.30

Anexo II : Conexionado de elementos con terminales de la tarjeta de prácticas

Tabla 6.2: Conexionado de elementos con terminales de la tarjeta de prácticas.

Terminal	Conexión
P0.4	Echo Sonar Osciloscopio
P0.5	Echo Sonar
P0.11	Trigger Sonar Osciloscopio
P0.15	LED rojo
P0.16	LED amarillo
P0.17	LED verde
P0.18	LED rojo
P0.19	LED amarillo
P0.20	LED verde
P0.21	LED rojo
P0.22	LED amarillo
P0.23	LED verde
P0.26	Osciloscopio
P0.27	SDA
P0.28	SCL
P1.9	Freno Motor DC
P1.10	Dirección Motor DC
P1.18	PWM Motor DC
P1.20	PWM Servomotor
P1.22	Segmento a
P1.23	Segmento b
P1.24	Segmento c
P1.25	Segmento d
P1.26	Segmento f
P1.27	Segmento g
P1.28	Segmento h Osciloscopio
P1.29	Segmento DP Osciloscopio
P3.25	Osciloscopio
P3.26	Osciloscopio

Anexo III : Conexionado de los canales del multiplexor con la tarjeta de prácticas.

Tabla 6.3: Conexionado de los canales con la tarjeta de prácticas.

Canal	Terminal	Principal funcionalidad
0	P1.23	PWM1.4
1	P3.25	MAT0.0 y PWM1.2
2	P3.26	MAT0.1 y PWM1.3
3	P0.26	AD0.3 y AOUT
4	P0.11	MAT3.1
5	P0.4	CAP2.0
6	P1.29	MAT0.1 y CAP1.1
7	P1.28	MAT0.0 y CAP1.0

Apéndice A

Manual de usuario

A continuación, se muestra el manual de usuario desarrollado para la aplicación de generación y control de señales.

Manual de Usuario
Aplicación de control y generación de señales
Version 2022-2



Índice

1. Introducción	3
2. Acceso remoto al ordenador de la plataforma	3
3. Programas disponibles en el ordenador de la plataforma de prácticas	4
4. Carga de un programa desde Keil uVision5 del ordenador remoto a la tarjeta de prácticas	5
5. Acceso a la aplicación web	7
6. Control y generación de señales	9
6.1. Interfaz para el control y generación de señales digitales	9
6.1.1. Cambio de nivel de las señales digitales	10
6.1.2. Generación y control de secuencias	11
6.2. Interfaz para el control y generación de señales PWM y analógicas	14
6.2.1. Generación de señales analógicas	14
6.2.2. Generación de señales PWM	16
6.2.3. Selección de los canales del multiplexor	17
6.2.4. Manejo del osciloscopio	18
7. Anexos	19

Índice de figuras

1.	Icono de AnyDesk.	3
2.	Conexión al ordenador de la plataforma de prácticas.	3
3.	Programas del ordenador de la plataforma.	4
4.	Modificar opciones del programa.	5
5.	Selección de carga sobre tarjeta.	6
6.	Compilación del programa.	6
7.	Carga del programa.	6
8.	Ejecución del programa.	7
9.	Icono de la aplicación web.	7
10.	Acceso a la aplicación web.	8
11.	Ventana de bienvenida de la aplicación.	8
12.	Interfaz para el control y generación de señales digitales.	9
13.	Encabezado de la interfaz de señales digitales.	10
14.	Señal "a" a nivel bajo.	10
15.	Señal "a" a nivel alto.	10
16.	Modificar el tiempo por símbolo de una secuencia.	11
17.	Introducir valores por símbolo de una secuencia.	11
18.	Botones de control de una secuencia.	12
19.	Botones de control de las secuencias de un mismo grupo	12
20.	Interfaz para el control y generación de señales PWM y analógicas.	14
21.	Configuración de la señal analógica.	15
22.	Generación y control de la señal analógica.	15
23.	Configuración de las señales PWM.	16
24.	Generación y control de las señales PWM.	17
25.	Selección del canal del multiplexor.	17
26.	Ejemplo de selección del canal cuatro del multiplexor.	18
27.	Manejo del osciloscopio de la plataforma.	18

1. Introducción

El objetivo de este manual es guiar al alumno en el uso de la aplicación para el control y generación de señales, de manera que sea capaz de poder realizar pruebas sobre los programas desarrollados de forma remota.

Los pasos que se seguirán para aprender el manejo de la aplicación son los siguientes:

- Acceso a la aplicación web y carga de un programa en Keil uVision5.
- Manejo de la interfaz de señales digitales.
- Manejo de la interfaz de señales analógicas y PWM.

2. Acceso remoto al ordenador de la plataforma

Para acceder al ordenador de la plataforma de prácticas el usuario deberá tener instalada la aplicación **AnyDesk** que permite acceder y controlar un ordenador de manera remota. El icono de la aplicación es el siguiente:

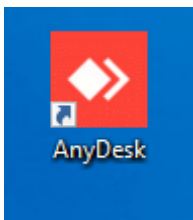


Figura 1: Icono de AnyDesk.

Una vez instalada dicha aplicación la ejecutamos apareciendo la ventana de bienvenida de la aplicación. En ella deberemos clickar sobre la barra de **Introduzca la dirección destino para la creación de sesión** marcado en color verde en la figura 2 y escribir el número del puesto de trabajo al que se desea acceder. Una vez escrito se deberá clickar sobre la flecha marcada en rojo en la figura 2, iniciándose así la conexión al ordenador de la plataforma de prácticas. El número del puesto que se encuentra montado en el laboratorio es 773998482.

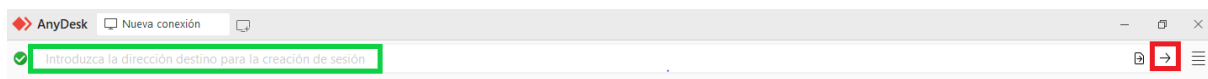


Figura 2: Conexión al ordenador de la plataforma de prácticas.

3. Programas disponibles en el ordenador de la plataforma de prácticas

Una vez hemos accedido al ordenador de la plataforma, nos encontraremos con las siguientes aplicaciones en el escritorio:

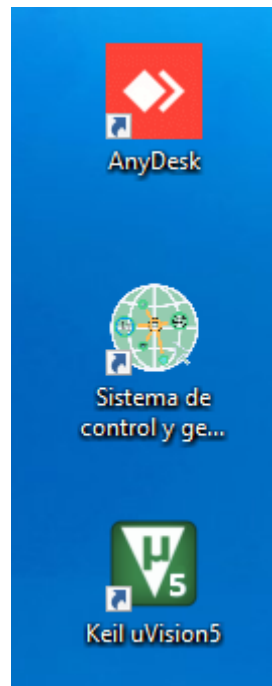


Figura 3: Programas del ordenador de la plataforma.

Se describe la utilidad de cada una de estas aplicaciones a continuación:

- **AnyDesk.** Aplicación utilizada para acceder al ordenador de la plataforma. Deberá mantenerse abierta durante todo el proceso de pruebas sobre la plataforma pues en caso contrario se perderá la conexión al ordenador remoto.
- **Sistema de control y generación de señales.** Aplicación a partir de la cual se generarán y controlarán diversas señales con las que se excitará a la tarjeta de prácticas con el fin de realizar las pruebas que el usuario necesite.
- **Keil uVision5.** Aplicación para programar microcontroladores que el usuario empleará para el desarrollo de sus prácticas y posterior depuración sobre la plataforma de prácticas.

4. Carga de un programa desde Keil uVision5 del ordenador remoto a la tarjeta de prácticas

Para poder realizar pruebas de las prácticas es necesario cargar el programa sobre la tarjeta de prácticas y posteriormente trabajar sobre las ventanas de depuración. Para ello, se deben seguir los pasos que se detallan a continuación:

1. Primero debemos asegurarnos que se va a trabajar sobre la tarjeta de prácticas y no sobre simulación. Para ello, se debe hacer click derecho sobre la carpeta principal del proyecto y seleccionar *Options for Target ...* tal y como se muestra en la figura 4.

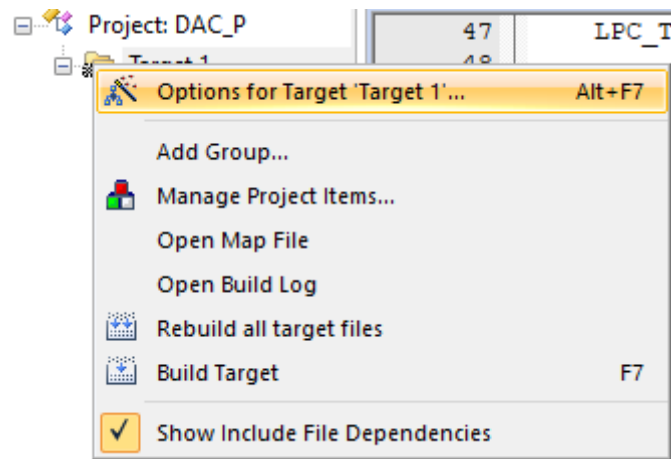


Figura 4: Modificar opciones del programa.

2. Dentro de esa ventana deberemos seleccionar *Debug*, marcar *Use J-LINK/J-TRACE Cortex* y clickar sobre *Settings* para comprobar que el ordenador ha reconocido correctamente la sonda. En la figura 5 se muestra en rojo los pasos seguidos.

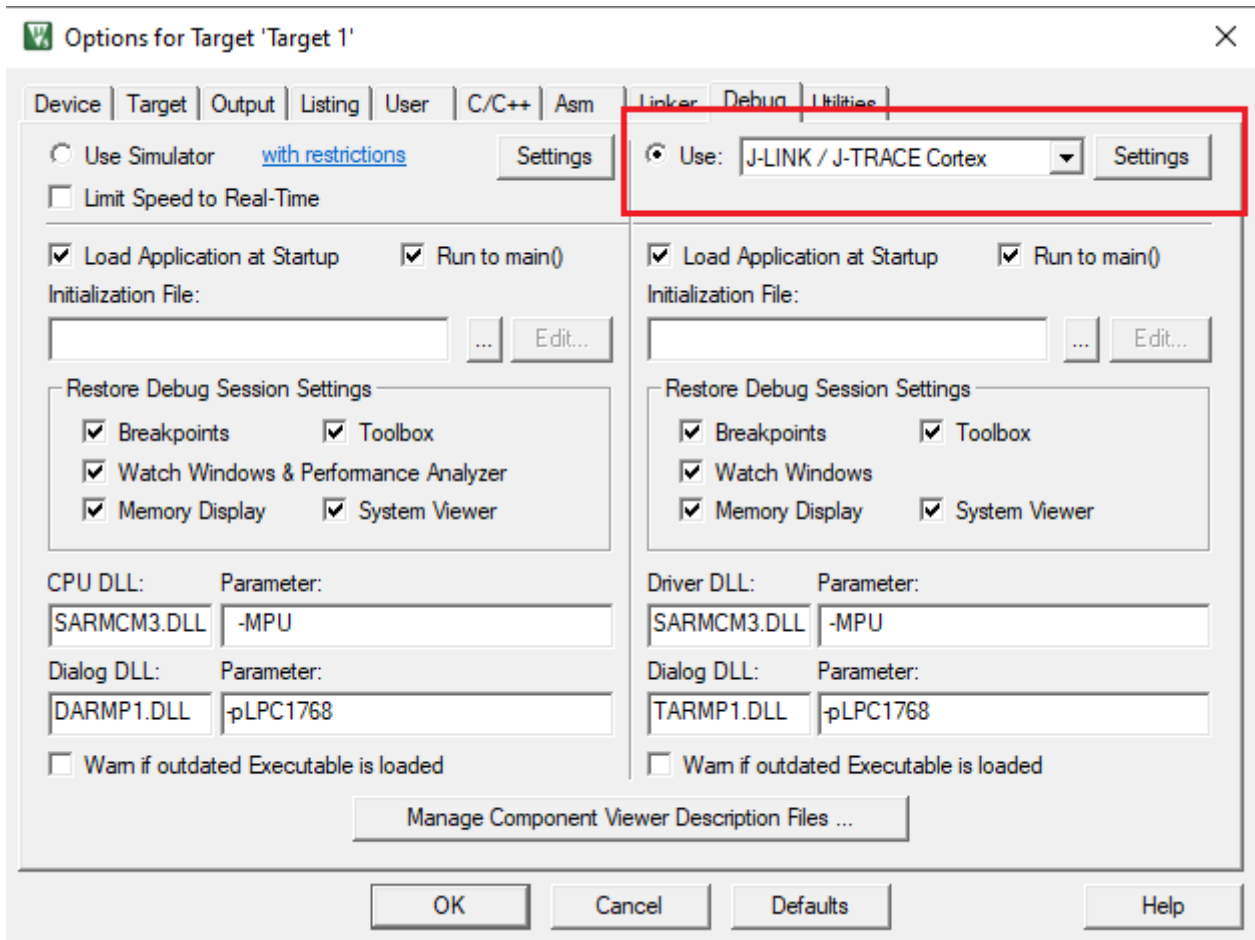


Figura 5: Selección de carga sobre tarjeta.

- Una vez configurado el método de programación, se debe compilar el programa pulsando sobre el botón marcado en color rojo de la figura 6.

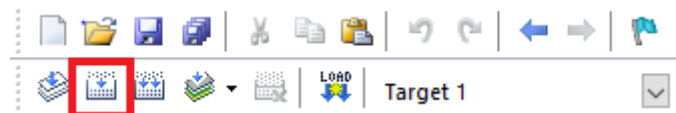


Figura 6: Compilación del programa.

- Tras esto, se realiza la carga del programa accediendo a la ventana de simulación sobre tarjeta que se muestra en la figura 7.

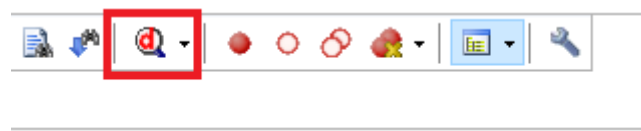


Figura 7: Carga del programa.

5. Finalmente, se debe ejecutar el programa pulsando sobre el botón marcado en rojo en la figura 8.



Figura 8: Ejecución del programa.

5. Acceso a la aplicación web

Una vez cargado el programa sobre la tarjeta de prácticas, se debe acceder a la aplicación de control y generación de señales, con el fin de poder realizar las pruebas necesarias sobre el programa cargado. Los pasos a seguir son los siguientes:

1. Para acceder a la aplicación web debemos seleccionar el icono que se muestra en la figura 9 que se encuentra en el escritorio del ordenador remoto.

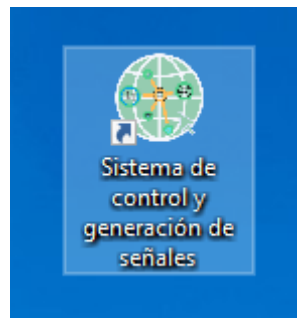


Figura 9: Icono de la aplicación web.

2. Una vez ejecutada la aplicación, saldrá una ventana emergente que solicitará los credenciales de acceso como la mostrada en la figura 10. Los credenciales a introducir son *admin/admin*.

Iniciar sesión

http://169.254.67.114

Tu conexión con este sitio web no es privada

Nombre de usuario

Contraseña

Figura 10: Acceso a la aplicación web.

3. Dentro de la aplicación, se muestra la pantalla principal que consta de dos botones: El botón *Señales digitales* da acceso a la interfaz para controlar y generar señales digitales. Por otro lado, el botón *PWM y señales analógicas* da acceso a la interfaz para el control de este tipo de señales. En la figura 11 se muestra la interfaz de la ventana de bienvenida de la aplicación.



Figura 11: Ventana de bienvenida de la aplicación.

6. Control y generación de señales

A continuación se describirá el funcionamiento de las interfaces desarrolladas en la aplicación y cómo generar y controlar las distintas señales que en ellas se encuentran. Además, en el Anexo I se puede encontrar el conexionado entre las señales generadas en la interfaz y los terminales de la tarjeta de prácticas. De igual manera, en el Anexo II se muestra el conexionado realizado desde la tarjeta de prácticas a los distintos elementos de la plataforma de prácticas.

6.1. Interfaz para el control y generación de señales digitales

Si se selecciona la página de generación y control de señales digitales, la interfaz que se muestra es la siguiente:

The screenshot shows the 'Control de señales digitales' web interface. At the top, there is a header with the University of Alcalá logo and the Department of Electronics. Below the header, there is a warning message: 'AVISO: Si se quiere utilizar la funcionalidad de generar señales analógicas, no debe modificarse el tiempo de símbolo de las señales i y j, pues emplean el mismo TIMER.' The main configuration section is titled 'Configuración del tiempo por símbolo' and contains four input fields for signal timing: 'Tiempo por símbolo señales a y b: 1000000 µs', 'Tiempo por símbolo señales c y d: 1000000 µs', 'Tiempo por símbolo señales e, f, g y h: 1000000 µs', and 'Tiempo por símbolo señales i y j: 1000000 µs'. Below this is a 'Send data' button. The main area is titled 'Activación señales digitales' and contains a grid of controls for signals a through j. Each signal row includes 'ON/OFF' buttons, a 'Señal' label, and options for signal sequence and cycling. At the bottom, there are additional controls for signals k through n.

Figura 12: Interfaz para el control y generación de señales digitales.

El primer elemento con el que nos encontraremos en la interfaz es el encabezado de la página, que a parte de mostrar los diversos logos referentes a la Universidad de Alcalá y el Departamento de Electrónica, tiene unos botones que permiten la navegación entre páginas.

En la figura 13 se muestra el encabezado de manera más detallada. Marcado en color verde se encuentra el icono de la casa, que al seleccionarlo nos redirigirá a la ventana de bienvenida de la aplicación. Por otro lado, el icono de la sinusoidal marcado en color azul nos redirigirá a la ventana de generación y control de PWM y señales analógicas.

Figura 13: Encabezado de la interfaz de señales digitales.

La interfaz se encuentra dividida en dos partes, una de configuración para las secuencia (que será tratado más adelante) y una para la activación de las señales digitales.

6.1.1. Cambio de nivel de las señales digitales

En total la interfaz cuenta con catorce señales digitales. Todas ellas permiten el cambio de nivel de la señal mediante el uso de los botones de **ON** y **OFF**, mostrándose, además, en color **rojo** cuando las señales están a nivel bajo (OFF) y en color **verde** cuando se encuentran a nivel alto (ON). De manera predeterminada, las señales "a" hasta la "j" se encuentran a nivel bajo y desde la señal "k" hasta la "n" a nivel alto.

Cuando la señal se encuentra a nivel bajo, implica 0 voltios en el respectivo terminal de entrada de la tarjeta de prácticas. De manera análoga un nivel alto en la señal supone 3,3 voltios en el respectivo terminal de entrada de la tarjeta de prácticas.

En la figura 14 se muestra la señal "a" a nivel bajo y como al pulsar el botón **ON** cambia a nivel alto (figura 15).



Figura 14: Señal "a" a nivel bajo.

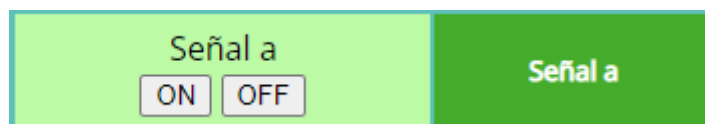


Figura 15: Señal "a" a nivel alto.

6.1.2. Generación y control de secuencias

Diez de las catorce señales con las que cuenta la interfaz, además de permitir su cambio de nivel, cuentan con la funcionalidad de generación de secuencias. Cabe resaltar que mientras se está generando una secuencia en una señal no se puede cambiar su nivel y viceversa. La señales que permiten generar secuencias son desde la señal "a" hasta la "j".

A continuación, se va a describir cómo generar y controlar una secuencia. Se informa que las señales se encuentran agrupadas según el temporizador que emplean, es decir, las señales "a" y "b" (grupo de color verde) emplean el mismo temporizador y por tanto, poseen una misma temporización por símbolo. De manera análoga las señales de la "e" hasta la "h" emplean el mismo temporizador y se han agrupado en color azul. Además, el tiempo por símbolo de cada secuencia es el mismo por cada grupo de señales.

Como ejemplo de generación de una secuencia se tomará la señal "a". Los pasos para generar una secuencia son los siguientes:

1. Debe introducirse el tiempo de emisión de cada símbolo de una secuencia. En el ejemplo, se desea emitir una secuencia por la señal "a", por tanto, debe modificarse el valor del recuadro naranja de la figura 16 por el valor deseado. Una vez hecho esto, se debe pulsar el botón **Send data** marcado en color rojo, para enviar el nuevo valor de tiempo por símbolo de las secuencias. Se debe remarcar que mientras se está produciendo la emisión de secuencias no se permite modificar el valor del tiempo por símbolo de estas. Por defecto, el tiempo por símbolo de las secuencias es de un segundo.

Configuración del tiempo por símbolo			
Tiempo por símbolo señales a y b:	1000000 μs	Tiempo por símbolo señales c y d:	1000000 μs
Tiempo por símbolo señales e, f, g y h:	1000000 μs	Tiempo por símbolo señales i y j:	1000000 μs
Send data			

Figura 16: Modificar el tiempo por símbolo de una secuencia.

2. Una vez modificado el tiempo por símbolo de la secuencia por el deseado se deberán asignar los valores a esta. Cada secuencia cuenta con dieciséis símbolos y para modificar el estado de cada uno de ellos la interfaz cuenta con dieciséis casillas donde se selecciona el nivel de cada símbolo. Si la casilla se encuentra marcada, el símbolo se emitirá a nivel alto y si la casilla se encuentra desmarcada a nivel bajo. En la figura 17 se muestra un ejemplo de valores para los símbolos de la secuencia de la señal "a".

Secuencia señal a:	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Secuencia señal b:	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figura 17: Introducir valores por símbolo de una secuencia.

3. Finalmente, nos encontramos con una serie de botones que permiten controlar la emisión de la secuencia. Los botones son los mostrados en la siguiente figura:



Figura 18: Botones de control de una secuencia.

- **Casilla ciclica.** La casilla que indica Ciclica X, donde X es la señal desde la “a” hasta la “j”, permite seleccionar si la secuencia se emitirá de manera periódica o una sola vez.
- **Botón Start.** Da inicio a la emisión de la secuencia.
- **Botón Stop.** Detiene la emisión de la secuencia, permitiendo reanudarla si se pulsa nuevamente el botón *Start*.
- **Botón Clear.** Termina con la emisión de la secuencia y restablece los valores de esta a los predeterminados (símbolos a nivel bajo y periodicidad desactivada). Este botón debe pulsarse si se requiere modificar algún parámetro de la secuencia/as en emisión, pulsando el Botón **Start** para volver a emitirla/as.

Cuando se desea generar más de una secuencia del grupo, se deben emplear los botones mostrados en la figura 18 para controlar todas las secuencias a la vez. Se recomienda revisar los últimos puntos de este apartado para tener en cuenta las restricciones en el uso de las secuencias.

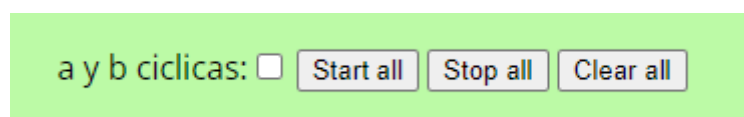


Figura 19: Botones de control de las secuencias de un mismo grupo

Para el uso de las secuencias se deberán tener una serie de consideraciones en cuenta, las cuales son citadas a continuación:

- Cuando se desea emitir más de una secuencia del grupo, sólo podrá usarse el botón de **Start all** para dar comienzo a la emisión de todas las secuencias del grupo de manera simultánea, quedando anulados por tanto los botones de **Start** individuales de cada señal.
- Cuando se emiten todas las secuencias pertenecientes a un mismo grupo, la selección de su periodicidad solo puede realizarse mediante la casilla que abarca a todas las secuencias del mismo grupo.

- Si se está emitiendo una secuencia de manera independiente y se desea emitir otra secuencia perteneciente al mismo grupo, deberá pararse la primera de ellas pulsando su botón **Stop** y activar al conjunto desde el botón de **Start all**.
- Si se desea modificar algún parámetro de una secuencia que se emite de forma individual, ya sea su periodicidad, tiempo por símbolo o valores de cada símbolo, debe terminarse la emisión de la secuencia pulsando el botón **Clear**, modificar los parámetros y reiniciar la emisión pulsando **Start**.
- Si están en emisión todas las secuencias pertenecientes a un mismo grupo y se desea parar una de ellas, esto se realizará desde el botón de **Stop** de dicha secuencia, parando únicamente la secuencia indicada. De igual manera, si se desea modificar algún parámetro de una de las secuencias en concreto, se deberá pulsar el botón **Clear** de dicha secuencia, modificarla y para volver a emitirla, deberá pulsarse el botón **Start all**.
- Si se desea emitir secuencias con las señales i y j, debe tener en cuenta que no podrá usarse la generación de la señal analógica, pues emplean el mismo temporizador en su emisión.

6.2. Interfaz para el control y generación de señales PWM y analógicas

La interfaz dedicada a la generación y control de señales PWM y analógicas es la mostrada en la siguiente figura:

Control de señales analógicas

Universidad de Alcalá

Departamento de Electrónica

AVISO: Si se quiere utilizar la funcionalidad de secuencia de las señales 1 y 2, no debe modificarse la frecuencia de las señales analógicas, pues emplean el mismo TIMER.

Configuración señales analógicas

Frecuencia de la señal analógica: 1 Hz

Amplitud de la señal analógica: 0 mV

Configuración señales PWM

Frecuencia de las señales PWM: 100 Hz

Ciclo de trabajo PWM1.1 (single edged): 0 %

Ciclo de trabajo PWM1.2 (single edged): 0 %

Ciclo de trabajo PWM1.3 (doubled edged): 0 %

Ciclo de trabajo PWM1.4 (doubled edged): 0 %

Ciclo de trabajo PWM1.5 (doubled edged): 0 %

Ciclo de trabajo PWM1.6 (doubled edged): 0 %

Send data

Activación señales analógicas

Start Sinusoidal

Start Sierra

Start Triangular

Stop

Activación señales PWM

Start Stop

Selección multiplexor de la plataforma

Manejo osciloscopio

0 P1.23

2 P3.26

4 P0.11

6 P1.29

P3.25 1

P0.26 3

P0.4 5

P1.28 7

Send

+ - sel

Figura 20: Interfaz para el control y generación de señales PWM y analógicas.

Al igual que sucedía en la interfaz de señales digitales nos encontramos con un encabezado que permite la navegación entre las distintas páginas de la aplicación. El icono de la casa redirige a la página de bienvenida y el de la señal cuadrada a la interfaz de las señales digitales.

Por otro lado, la interfaz está diseñada con el mismo formato de la interfaz de señales digitales, encontrándonos con un apartado que permite configurar los parámetros de las distintas señales a generar y otro para la activación de dichas señales. Además, la interfaz cuenta con un apartado dedicado a la selección de los canales del multiplexor analógico de la plataforma de prácticas. A continuación se describirá el manejo de la interfaz.

6.2.1. Generación de señales analógicas

La interfaz permite la generación de una señal analógica que puede tener forma de onda sinusoidal, diente de sierra o triangular. Los pasos a seguir para generar la señal analógica son los siguientes:

1. Debe configurarse la frecuencia de la señal analógica a generar introduciendo el valor

en Hz en el recuadro marcado en verde de la figura 21. Por defecto, la frecuencia de la señal es de 1 Hz y el máximo de 1MHz.

Además, se puede modificar la amplitud de la señal introduciendo el valor en mV en el recuadro marcado en color azul de la figura 21. De manera predeterminada, el valor de la amplitud es de 0 mV. El valor **máximo** de amplitud que puede introducirse es de **3300 mV**.



Figura 21: Configuración de la señal analógica.

2. Una vez introducida la configuración de la señal analógica, debe pulsarse el botón de **Send data** situado debajo de la configuración de las PWM y marcado en rojo en la figura 23.
3. Por último, para generar la señal analógica la interfaz cuenta con el menú mostrado en la figura 22.



Figura 22: Generación y control de la señal analógica.

En dicho menú, contamos con los siguientes botones:

- **Start Sinusoidal**. Genera una señal analógica con forma de onda sinusoidal.
- **Start Sierra**. Genera una señal analógica con forma de diente de sierra.
- **Start Triangular**. Genera una señal analógica con forma de triangular.
- **Stop**. Detiene la emisión de la señal analógica.

La aplicación permite conmutar la forma de la señal analógica generada sin necesidad de pulsar el botón de **Stop**, es decir, si se está generando una sinusoidal y se requiere generar una señal triangular, el usuario puede pulsar sobre **Start Triangular** y la forma de onda conmutaría de manera instantánea.

Por otro lado, si se modifica algún parámetro de la configuración de la señal analógica deberá ser pulsado el botón de **Start**, tras pulsar **Send data**, de la forma de onda a generar para que los cambios sean notables.

6.2.2. Generación de señales PWM

La aplicación es capaz de generar hasta cuatro señales PWM, dos de tipo single edged y dos de tipo doubled edged. El tipo de PWM a generar está indicado en la descripción de la señal PWM entre paréntesis (Figura 23).

Para las PWM double edged, la señal se genera por el terminal del PWM mayor. Por ejemplo, una señal doubled edged es la formada por la PWM1.3 y PWM1.4 del grupo azul (Figura 23) y la señal PWM será emitida por el terminal de PWM1.4.

Los pasos a seguir para generar una señal PWM son los siguientes:

1. Primero, debe configurarse los parámetros de las señales PWM. La frecuencia es la misma para todas las PWM y está expresada en Hz, por defecto, su valor es de 100 Hz (marcado en verde en la figura 23). Por otro lado, se debe configurar el ciclo de trabajo que tendrá cada señal PWM, siendo por defecto, del 0 %. Los recuadros para modificar el ciclo de trabajo de las PWM se marcan en azul en la figura 23.
2. Una vez configurados los parámetros, debe pulsarse el botón de **Send data** marcado en color rojo en la figura 23 para enviar la configuración realizada.

Configuración PWMs	
Frecuencia de las señales PWM:	<input type="text" value="100"/> Hz
Ciclo de trabajo PWM1.1 (single edged):	<input type="text" value="0"/> %
Ciclo de trabajo PWM1.2 (single edged):	<input type="text" value="0"/> %
Ciclo de trabajo PWM1.3 (doubled edged):	<input type="text" value="0"/> %
Ciclo de trabajo PWM1.4 (doubled edged):	<input type="text" value="0"/> %
Ciclo de trabajo PWM1.5 (doubled edged):	<input type="text" value="0"/> %
Ciclo de trabajo PWM1.6 (doubled edged):	<input type="text" value="0"/> %
<input type="button" value="Send data"/>	

Figura 23: Configuración de las señales PWM.

3. Finalmente, para generar las señales PWM la interfaz cuenta con dos botones. El botón de **Start** que da comienzo a la emisión de las PWM y el botón de **Stop** que detiene la emisión de estas. Dichos botones los los mostrados en la figura 24.



Figura 24: Generación y control de las señales PWM.

Cada vez que se realice alguna modificación en algún parámetro de una señal PWM, tras pulsar el botón **Send data** deberá pulsarse el botón de **Start** de la figura 24 para aplicar los cambios.

6.2.3. Selección de los canales del multiplexor

La plataforma de prácticas cuenta con un osciloscopio y un multiplexor analógico de ocho canales que permite la selección del canal a mostrar en el osciloscopio según requiera el usuario.

Para la selección del canal a mostrar en el osciloscopio, la interfaz cuenta con un apartado que se describe a continuación:

1. En las casillas de la interfaz, marcadas en color azul en la figura 25, debe seleccionarse el canal a mostrar donde si la casilla esta marcada es un nivel alto ('1') y si está desmarcada es un nivel bajo ('0').
2. Una vez introducido el canal a visualizar, se deberá pulsar el botón **Send** (marcado en rojo en la figura 25) para enviar la información del canal seleccionado. Por defecto se encuentra seleccionado el canal 0.

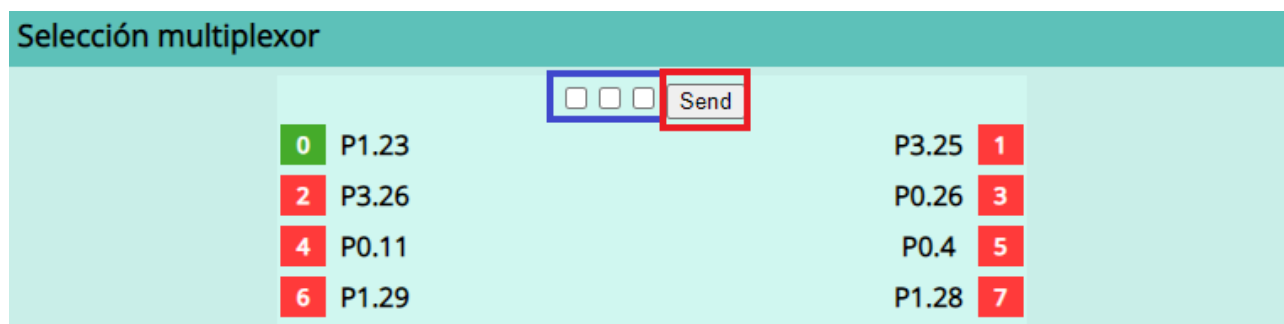


Figura 25: Selección del canal del multiplexor.

El canal que se visualiza en el osciloscopio es el que se encuentra iluminado en verde, estando el resto en color rojo. Por ejemplo, al introducir en las casillas de selección el valor '101' y pulsar el botón **Send** se cambia el canal seleccionado al canal 4, mostrándose este en color verde como en la figura 26.

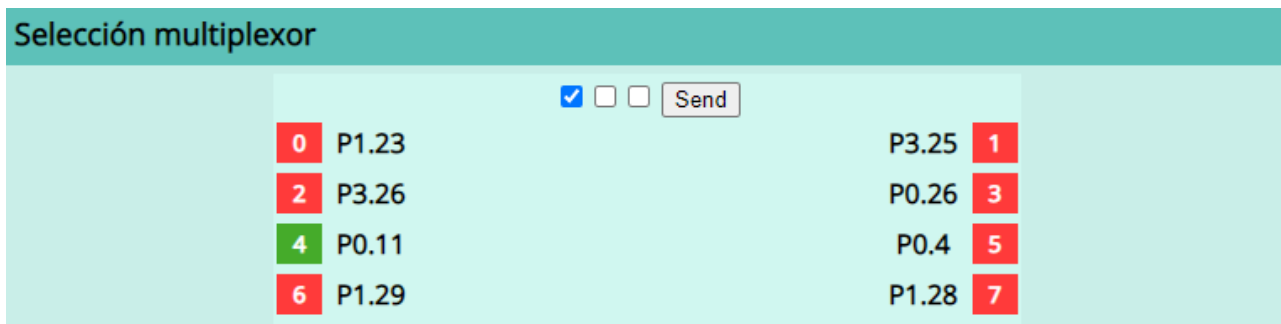


Figura 26: Ejemplo de selección del canal cuatro del multiplexor.

En el Anexo III se muestra el conexionado realizado en cada uno de los canales del multiplexor, así como la principal funcionalidad del terminal conectado.

6.2.4. Manejo del osciloscopio

Para controlar el osciloscopio de la plataforma de prácticas la interfaz cuenta con los botones mostrados en la figura 27.

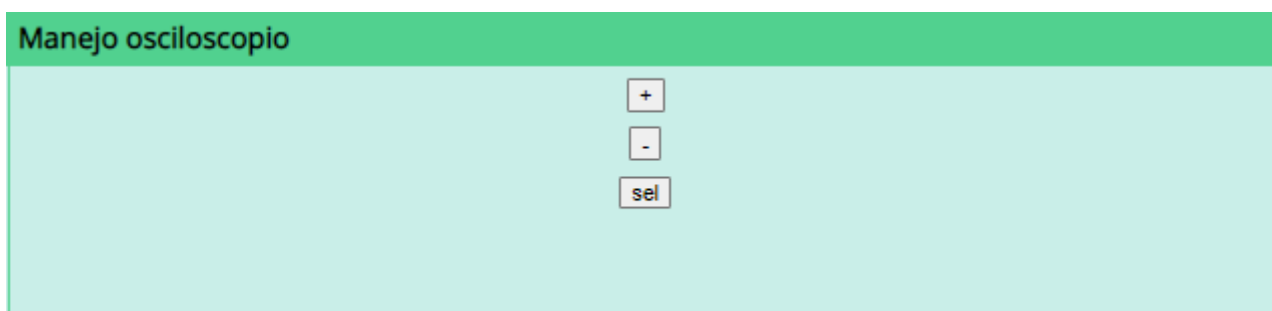


Figura 27: Manejo del osciloscopio de la plataforma.

Para accionar los botones será necesario hacer **dobles clics** sobre el botón deseado. A continuación se hace una descripción de cada uno de los botones:

- **Botón +.** Amplia la escala de tiempos del osciloscopio cuando se tiene seleccionado el elemento de tiempo del osciloscopio.
- **Botón -.** Disminuye la escala de tiempos del osciloscopio cuando se tiene seleccionado el elemento de tiempo del osciloscopio.
- **Botón sel.** Permite navegar entre los elementos mostrados en la pantalla del osciloscopio.

7. Anexos

Anexo I : Conexionado de señales con terminales de entrada de la tarjeta de prácticas

Tabla 1: Conexionado de señales con terminales de entrada de la tarjeta de prácticas.

Señal	Terminal de entrada	Señal	Terminal de entrada
Señal a	P0.0	PWM1.1	P2.5
Señal b	P0.1	PWM1.2	P2.6
Señal c	P1.16	PWM1.4	P2.7
Señal d	P1.17	PWM1.6	P2.8
Señal e	P1.19	Señal analógica	P1.30
Señal f	P2.0		
Señal g	P2.1		
Señal h	P2.2		
Señal i	P2.3		
Señal j	P2.4		
Señal k	P2.10		
Señal l	P2.11		
Señal m	P2.12		
Señal n	P2.13		

Anexo II : Conexionado de elementos con terminales de la tarjeta de prácticas

Tabla 2: Conexionado de elementos con terminales de la tarjeta de prácticas.

Terminal	Conexión
P0.4	Echo Sonar Osciloscopio
P0.5	Echo Sonar
P0.11	Trigger Sonar Osciloscopio
P0.15	LED rojo
P0.16	LED amarillo
P0.17	LED verde
P0.18	LED rojo
P0.19	LED amarillo
P0.20	LED verde
P0.21	LED rojo
P0.22	LED amarillo
P0.23	LED verde
P0.26	Osciloscopio
P0.27	SDA
P0.28	SCL
P1.9	Freno Motor DC
P1.10	Dirección Motor DC
P1.18	PWM Motor DC
P1.20	PWM Servomotor
P1.22	Segmento a
P1.23	Segmento b
P1.24	Segmento c
P1.25	Segmento d
P1.26	Segmento f
P1.27	Segmento g
P1.28	Segmento h Osciloscopio
P1.29	Segmento DP Osciloscopio
P3.25	Osciloscopio
P3.26	Osciloscopio

Anexo III : Conexionado de los canales del multiplexor analógico

Tabla 3: Conexionado de los canales del multiplexor analógico.

Canal	Terminal	Principal funcionalidad
0	P1.23	PWM1.4
1	P3.25	MAT0.0 y PWM1.2
2	P3.26	MAT0.1 y PWM1.3
3	P0.26	AD0.3 y AOUT
4	P0.11	MAT3.1
5	P0.4	CAP2.0
6	P1.29	MAT0.1 y CAP1.1
7	P1.28	MAT0.0 y CAP1.0

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR



Universidad
de Alcalá