



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Plataforma Robótica cartesiana para la medición de distancias mediante sensor de ultrasonidos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: José Eduardo Reinoso Andrade

Tutor: Enrique Jorge Bernabeu Soler
Fernando Juan García Diego

Curso 2017-2018

Agradecimientos y dedicatoria

Este proyecto no habría sido posible sin la ayuda de muchas personas. Primeramente, gracias a Fernando Juan Garcia Diego, que aun siendo el co-tutor de este proyecto, siempre me dio su apoyo, aún con todas las dificultades que surgieron. También agradecer a Enrique Jorge Bernabeu Soler, por ser el tutor de mi proyecto.

Quiero agradecer a mi familia por su apoyo constante en mis estudios y su fe ciega en mi. A mi madre Sarita de los Angeles Andrade Sucunuta, la cual siempre ha estado a mi lado y me enseñó lo que una persona puede hacer sin ayuda de nadie, solo con fuerza de voluntad e inteligencia. A mi padre Luis Eduardo Reinoso Sanchez que me enseñó con su ejemplo, a ser una buena persona y que todo esfuerzo en esta vida tiene su recompensa. A mis tíos Patricia Reinoso y Jesús Acosta, por preocuparse siempre de mí como unos padres. Finalmente, a Michelle Montoya la persona que ha llenado de color mi vida y que me ayudó en la redacción de este proyecto.

Muchas gracias a todos.

Resumen

En este proyecto se presenta una plataforma robótica cartesiana para la medición de distancias con un sensor de ultrasonidos. Se desarrolla un sistema de medición de distancias, basado en la emisión y recepción de ultrasonidos, que funciona haciendo la medición de dos parámetros distintos, el tiempo de vuelo (ToF) y el desfase de la señal (PSA). El sistema se ha desarrollado, a partir, de un sistema estándar de bajo coste. El hardware inicial se modifica, asociándolo a un condensador que permite la cuantificación del desfase atendiendo a tiempos de carga distintos, los cuales varían en función del desfase de la señal. Con esto, se consigue que la precisión del sensor de ultrasonidos aumente sustancialmente. Además, se desarrolla y describe un sistema robótico escalable, para el desplazamiento del sensor de ultrasonidos por el plano horizontal XY.

Palabras clave: Mediciones de distancia, Sensor de ultrasonidos, Arduino, Robot cartesiano, Tiempo de vuelo, Desfase, PWM.

Abstract

In this project, a Cartesian robotic platform for measuring distances with an ultrasonic sensor is presented. A distance measurement system is developed, based on the emission and reception of ultrasound, which works by measuring two different parameters, the time of flight (ToF) and the phase shift of the signal (PSA). The system has been developed from a standard low-cost ultrasonic transmitter/receiver, the initial hardware is modified using a capacitor. Taking the information derived from the charging time of the capacitor, it is possible to measure a new parameter, the phase shift analysis. With this, the accuracy of the ultrasonic sensor has been substantially increased. A scalable robotic system is also developed to position the ultrasonic sensor over a study surface by the horizontal plane XY.

Key words: Distance measurement, Ultrasound sensor, Arduino, Cartesian robot, Time of flight, Phase shift, PWM.

Índice general

Índice general	V
Índice de figuras	VII
Índice de tablas	VIII
<hr/>	
1 Introducción	1
1.1 Motivación	2
1.2 Objetivos	2
1.3 Estructura	3
2 Estado del arte y crítica	5
2.1 Propuesta	8
3 Materiales y métodos	9
3.1 Microcontroladores ATmega328 y ATmega32u4	10
3.1.1 Problemas con el microcontrolador ATmega32u4	11
3.1.2 Generación de señales PWM y timers en ATmega32u4.	12
3.1.2.1 Descripción de las señales PWM	12
3.1.2.2 Timers y generación de señales PWM	13
3.1.2.3 Manipulación de Timers	14
3.2 Módulo de ultrasonidos	18
3.2.1 Descripción básica	18
3.2.2 Funcionamiento, datos técnicos y especificaciones	19
3.2.3 Problemas y limitaciones.	20
3.2.3.1 Problema con la temperatura y condiciones de medida.	20
3.2.3.2 Solución a la temperatura y condiciones de medida.	21
3.2.3.3 Problema con la señal de recepción del módulo	21
3.2.3.4 Atenuación del problema con la señal de recepción del modulo.	27
3.3 Robot cartesiano XY	28
3.3.1 Motores paso a paso.	28
3.3.2 Controladores de los motores paso a paso.	28
3.3.3 Implementación final	30
4 Software y adaptaciones	33
4.1 Implementación software del robot cartesiano	34
4.2 Adaptación hardware del sensor de ultrasonidos e implementación software.	37
4.2.1 Adaptación hardware para la emisión del sensor de ultrasonidos.	37
4.2.2 Adaptación hardware para la recepción del sensor de ultrasonidos.	40
4.2.3 Adaptación hardware para la detección del phase shift.	41
4.2.4 Consideraciones para la implementación software del sensor de ul- trasonidos.	42
4.2.5 Implementación software del sensor de ultrasonidos.	42
4.3 Implementación software del calibrado	47
5 Calibrado	49
6 Resultados y discusiones	53

6.1	Detección de la tendencia del potencial a subida o bajada.	53
6.2	Calibración de los datos.	54
6.3	Distancia real y distancia medida.	54
6.4	Error absoluto.	56
7	Conclusiones	57
7.1	Relación del trabajo desarrollado con los estudios cursados.	58
Bibliografía		59

Apéndices

A	Glosario	61
B	Timers del microcontrolador ATmega32u4	63

Índice de figuras

2.1	Esquema de funcionamiento de un sensor láser.	6
2.2	Esquema de funcionamiento de un sensor capacitivo	6
2.3	Esquema de funcionamiento de un sensor inductivo.	7
2.4	Esquema de funcionamiento de un sensor de ultrasonidos.	7
3.1	Esquema final del sensor de ultrasonidos.	9
3.2	Arduino Leonardo	10
3.3	Arduino Nano	10
3.4	Señales PWM.	13
3.5	Pendiente simple PWM	14
3.6	Pendiente doble PWM	14
3.7	Timer/Counter1 Control Register A - TCCR1A	15
3.8	Timer/Counter1 Control Register A - TCCR3A	15
3.9	Timer/Counter1 Control Register B - TCCR1B	15
3.10	Timer/Counter1 Control Register B - TCCR3B	15
3.11	Configuración final del timer.	18
3.12	Front View	18
3.13	Back View	18
3.14	Módulo HC-SR04 Schematic	19
3.15	Señal emit	21
3.16	Señal teórica y real de signal	22
3.17	Señal para la emision de ultrasonidos	22
3.18	Señal de recepción de ultrasonidos	22
3.19	Distancia 100 mm	23
3.20	Distancia 100 mm - 0.5mm	23
3.21	Distancia 100 mm - 1.0 mm	24
3.22	Distancia 100 mm - 1.25 mm	24
3.23	Distancia 100 mm - 1.50 mm	24
3.24	Distancia 100 mm - 2.0 mm	25
3.25	Distancia 100 mm - 2.5 mm	25
3.26	Distancia 100 mm - 3.0 mm	25
3.27	Distancia 100 mm - 3.25 mm	26
3.28	Distancia 100 mm - 3.50 mm	26
3.29	Atenuación del problema con la señal signal	27
3.30	Motor paso a paso	28
3.31	Controlador A4988	29
3.32	Esquema del robot cartesiano XY.	30
3.33	Robot cartesiano XY	31
4.1	El diagrama de flujo de la comunicación de los programas desarrollados para obtener la orografía	34
4.2	Diagrama de flujo de ejecución del robot cartesiano	35
4.3	Malla de puntos	37
4.4	Esquema del circuito del módulo HC-SR04	38

4.5	Señales para la inicialización del proceso de generación de ultrasonidos. . .	39
4.6	Esquema de señales generadas, activación y REMIT generadas por software Tx1 y Tx2 por hardware	39
4.7	Componentes hardware	40
4.8	Proceso de captación de la señal de ultrasonidos	41
4.9	Señales usadas para el cálculo de PSA	41
4.10	Diagrama de flujo de ejecución para realizar la medición con el sensor de ultrasonidos	43
4.11	Código de la activación de ultrasonidos.	45
5.1	Dispositivo de calibración	49
5.2	Tornillo micrométrico para el control del calibrador.	49
6.1	En azul el valor del conversor A/D (1024 = 5 Voltios) y en rojo, la detección de potencial de subida (eje vertical = 100) o de bajada (eje vertical = 1000).	54
6.2	Valor del conversor A/D (1024 = 5 Voltios) por (a) potencial de subida, (b) potencial de bajada	54
6.3	Distancia real y medida	55
6.4	Error absoluto	56
A.1	Desfase de ondas, PSA.	61

Índice de tablas

3.1	Especificaciones técnicas del Arduino Leonardo.	11
3.2	Especificaciones técnicas del Arduino Nano.	11
3.3	Distintas configuraciones del preescaler.	16
3.4	Configuraciones del timer.	16
3.5	Comportamiento del canal de salida.	17
3.6	Características del controlador A4988.	29
3.7	Configuraciones del controlador A4988.	30
4.1	Tipos de activadores para una interrupción.	42
4.2	Valores de las señales para el estado de reposo.	44
5.1	Características del controlador A4988.	51

CAPÍTULO 1

Introducción

El hacer una medición de una distancia entre dos puntos, puede resultar algo trivial en estos tiempos. Cuando la medición es puntual y no es muy rigurosa, un humano puede realizarla manualmente, con algún instrumento de medición, que puede ser eléctrico (sensor láser) o no (metro o una regla). Conforme la medición vaya adoptando un carácter más crítico y riguroso, o se necesite hacer con más frecuencia. Las personas son sustituidas por robots, que realizan la función desplazarse al punto necesario y realizar la medición. Mientras que los instrumentos, a su vez, son sustituidos por dispositivos de medición más avanzados y con mayor precisión.

Sin embargo, aún es necesario en algunos campos conseguir técnicas y dispositivos que proporcionen buenas mediciones a distancias cortas. Aunque, no sea muy conocido, durante mucho tiempo empresas e instituciones públicas han estudiado y creado métodos para medir distancias de punto a punto, principalmente con el fin de detectar obstáculos. La mayoría de trabajos que existen sobre el tema, son extensiones de otros o modificaciones ingeniosas, como por ejemplo *“Design of a CMOS APD array for a 3-D camera based on the time of flight distance measurement”* [1], en el que se consiguen detectar objetos haciendo uso de metodologías ópticas con una cámara 3-D.

En este proyecto se presenta un sistema de medición basado en un sensor de ultrasonidos. La precisión de sistema es relativamente alta en lo que se refiere a mediciones de distancias. Se desarrolla, también, una plataforma robótica cartesiana, que se puede desplazar por encima del plano XY. Para la verificación de las mejoras del sistema de medición, se desarrolla, además, un sistema, para la calibración y testeo del dispositivo de medición. Todo esto, está controlado por medio de microcontroladores de bajas prestaciones, para así conseguir un diseño de alto rendimiento con componentes asequibles.

Una alta precisión en la medición de las distancias se consigue mediante el uso de sensores de ultrasonidos. En este proyecto se presentan un sistema basado en la combinación de las dos metodologías en relación con la medición de distancia basada en sensores ultrasónicos, ToF[2] y PSA[3]. Por un lado, se emite una ráfaga ultrasónica de 8 ciclos, mediante un sensor ultrasónico de 40 kHz, la diferencia de tiempo entre la salida y recepción de la señal da la información del tiempo de vuelo (ToF). Por otro lado, apoyándose en la carga de un condensador es posible determinar la variación de fase (PSA), lo que permite una medida de distancia de alta precisión a un coste muy reducido. Para el procesamiento de las mediciones se emplea un microcontrolador de 16 MHz soportado en Arduino Leonardo.

Una plataforma robótica cartesiana para el plano horizontal XY se implementa en el proyecto. Ésta sirve para realizar el movimiento a cualquier punto de la superficie del plano. El robot, también, es controlado por un microcontrolador Arduino de bajo coste.

El sistema final corresponde a una plataforma robótica cartesiana con zona de trabajo en el plano XY que realiza mediciones de distancias con una precisión relativamente alta. La medición de distancias se consigue haciendo uso del dispositivo que se desarrolla en este trabajo.

Al final del documento se puede encontrar un glosario para los términos que más se repiten en el trabajo, así como una breve explicación de éstos.

Además, se puede encontrar un anexo, el cual describe lo que son los **timers** y como manipularlos a petición. Ésta es una característica muy importante de los microcontroladores, de la cual se hace un uso crítico en este trabajo.

1.1 Motivación

El trabajo fue pensado para que su aplicación final sea en el campo del patrimonio cultural, concretamente, en el estudio de papiros o cualquier obra de arte antigua, como pinturas o mapas. Pudiendo así, estudiar la evolución de la orografía, en relación con los cambios de humedad y temperatura ambiente, sin que las mediciones que hagan los sensores sobre estas obras sean directas. Una medición láser, aunque es la más precisa, termina deteriorando la obra de arte. El uso de sensores de ultrasonidos a 40 Khz no dañan la superficie sobre la que incide, pudiendo estos sensores incluso limpiarlas de polvo microscópico. La conservación de cuadros y documentos antiguos es complicada. Muchas de las técnicas que se aplican en este campo para la evaluación del estado de estas obras de arte son caras e incluso, invasivas y peligrosas para su integridad.

En este proyecto se genera un prototipo de sensor de ultrasonidos para obtener la orografía, que por su naturaleza, no afecte a las obras y tenga una precisión muy buena con un coste de producción bajo.

Con ayuda del robot cartesiano se puede desplazar el sensor de ultrasonidos por encima de toda la superficie de la obra de arte, sin que exista contacto directo en ningún momento. Los datos de estas mediciones se guardan para luego poder ser tratados, procesados y estudiados.

1.2 Objetivos

El objetivo global del proyecto es doble. Por una parte, se busca implementar un dispositivo para medir distancias con una resolución muy alta. Por otra, se desea implementar un dispositivo robótico para desplazar el sensor sobre el plano horizontal XY.

Para la consecución de cada uno de los objetivos globales, se tienen que cumplir una serie de objetivos específicos.

Para el dispositivo de medición:

1. Estudiar el estado del arte, de dispositivos de medición.

Se va a analizar los distintos tipos de dispositivos y metodologías existentes para la medición de distancias. A fin de buscar, la mejor metodología y dispositivo para este proyecto.

2. Mejorar la precisión en la medición.

Se busca obtener una precisión en la medición mayor a 3 mm, realizándose las mediciones en distancias cortas y ambientes controlados.

3. El dispositivo para la medición debe ser *low cost*

Se quiere generar un dispositivo de altas prestaciones a un coste relativamente bajo, con respecto a los demás tipos de dispositivos de medición en el sector.

4. Sustituir el microcontrolador del sensor escogido

La mayoría de los dispositivos de medición tienen su propio microcontrolador específico. Éste se va a eliminar para ser sustituido por otro microcontrolador programable. De esta forma, se aumenta la complejidad del proyecto y mejora el funcionamiento del mismo.

Para el dispositivo robótico:

1. Implementar un robot cartesiano que permita el movimiento de un cabezal.

Se implementa un robot cartesiano, que permite el movimiento por encima de una superficie en el plano XY. El robot tiene acoplado un cabezal, a éste se puede unir el dispositivo de medición que se haya escogido.

2. El robot cartesiano debe ser escalable.

Para el proyecto se elabora un robot cartesiano, que tiene unas dimensiones de trabajo útil específicas. Sin embargo, esto no impide que el mismo robot pueda adaptarse a otras dimensiones de trabajo.

1.3 Estructura

En este capítulo se ha realizado una introducción al proyecto de forma global y concisa. También se ha expuesto la motivación del proyecto y los objetivos marcados.

En el Capítulo 2 se hace un análisis al estado del arte, con respecto a los sensores y técnicas de medición disponibles, centrándose en las técnicas de ultrasonidos.

En el Capítulo 3 se describen todos los materiales y dispositivos que se usan en el proyecto, también se especifican sus pros y contras, así como las mejoras llevadas a cabo en el hardware del proyecto.

En el Capítulo 4 se especifica y explica el software diseñado para que los distintos dispositivos del proyecto funcionen correctamente y puedan comunicarse, con el objetivo de realizar mediciones. En este capítulo también, se detallan modificaciones hardware realizadas sobre el módulo de ultrasonidos.

En el Capítulo 5 se describe como se ha elaborado el calibrado del proyecto y algunas especificaciones técnicas de éste.

En el Capítulo 6 se presentan los resultados que se han obtenido y como se han analizado. Los resultados y análisis están acompañados de gráficas explicativas.

En el Capítulo 7 se presentan las conclusiones del proyecto, así como la relación del trabajo realizado por el alumno y la carrera universitaria cursada.

Finalmente, se puede encontrar un glosario de los términos más repetidos y complejos en el proyecto y de un anexo, correspondiente a una sección concreta del documento de especificaciones del microcontrolador ATmega32u4.

Estado del arte y crítica

Un sensor electrónico en robótica es un dispositivo que puede medir distintas magnitudes físicas, como variaciones de luz, temperatura o sonido, así como otras alteraciones de su entorno. La medición que hace el sensor se convierte en una señal eléctrica, que puede ser captada por la unidad de control del robot o controlador asociado.

Los sensores se basan en varios tipos de mediciones (posición, velocidad, presencia, fuerza, temperatura, color, luz, etc.) con distintos principios de funcionamiento (infrarrojos, ultrasonidos, láser, imagen, sonar, etc.). Debido a esto, existen una gran variedad de sensores en el mercado que se pueden usar con el mismo propósito. Sin embargo, tienen diferentes prestaciones y características: tipo del sensor, precisión, resolución, rango, interfaz de control, condiciones de entorno, calibración, coste, etc.

En aplicaciones robóticas, los sensores más comunes son los empleados de forma habitual, como los sensores de distancia y proximidad (ultrasónicos, capacitivos, fotoeléctricos, inductivos, magnéticos) [4] los de posición y movimiento (basados en luz infrarroja, ultrasonido, tecnología de microondas / radar) [5], los sensores de imagen (cámaras digitales y dispositivos de imagen) [6] o los sensores de presión y fuerza (con piezoresistivos, optoelectrónicos o extensímetros) [7].

Estos sensores externos son críticos para aplicaciones robóticas. Por medio de ellos, el robot puede interactuar con entornos que no están estrictamente controlados. Esto se puede ver en el sector industrial, un robot encargado de manipular piezas puede realizar tareas de coger y colocar ("*pick&place*") en una cinta transportadora sin intervención humana [8]. No obstante, en aplicaciones en las que el robot debe hacer algún tipo de movimiento, como en la navegación autónoma [9], la prevención de obstáculos en entornos dinámicos o estáticos es un problema importante que debe resolverse. Resulta fundamental, proporcionar la distancia a un obstáculo o entre dos obstáculos con una precisión determinada.

La visión estéreo se ha utilizado con mucha frecuencia en robótica para calcular un mapa de navegación y rangos [10], pero esta técnica conlleva altos requisitos de cálculo y un precio elevado.

La precisión que se necesita en la medición de distancia, mediante un sensor, va a depender exclusivamente de la aplicación final.

En la historia de los sensores se han desarrollado varios métodos de medición de distancia sin contacto directo, basados en diferentes tecnologías. Para clasificarlos, se debe considerar la distancia a la que se va a realizar la medición y la superficie o condiciones del medio donde se realiza.

Los métodos ópticos permiten mediciones de alta precisión, a distancias cortas y largas, pero están limitados por las propiedades del medio. Afectando a la transparencia o

la capacidad de reflexión. Por lo que, si se estudian elementos como el vidrio, plástico transparente, agua, fibra o espumas, el error en la medición aumenta significativamente.

Las arquitecturas de infrarrojos proporcionan alta velocidad y alta resolución para mediciones de larga distancia.

Otra forma de determinar la posición de un robot móvil es por medio de un análisis de distancias diferenciales, obtenidas de una variación de fase, "*phase-shift*" [11]. Ambos métodos requieren un procesamiento de alta velocidad de la señal y altas prestaciones en el hardware que lo soporta.

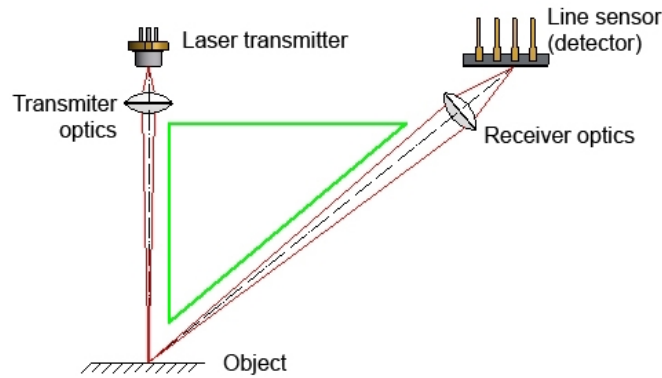


Figura 2.1: Esquema de funcionamiento de un sensor láser.

Los sensores capacitivos permiten realizar mediciones sin contacto directo de distancias cortas [12], [13] y tienen como limitación que requieren un ajuste de su sensibilidad dependiendo del material sobre el que se realiza la prueba. Algunos resultados interesantes con estos sensores se presentan en trabajos como: "*An infrastructure for freehand manipulation on interactive surfaces*" [12], en el que los autores identifican los gestos humanos, midiendo las distancias en la mano.

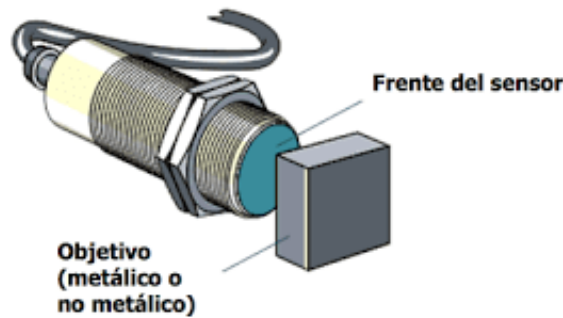


Figura 2.2: Esquema de funcionamiento de un sensor capacitivo

También, existen sensores inductivos para medir distancia, generalmente se utilizan para hacer mediciones de corta distancia y exclusivamente sobre metales [14].

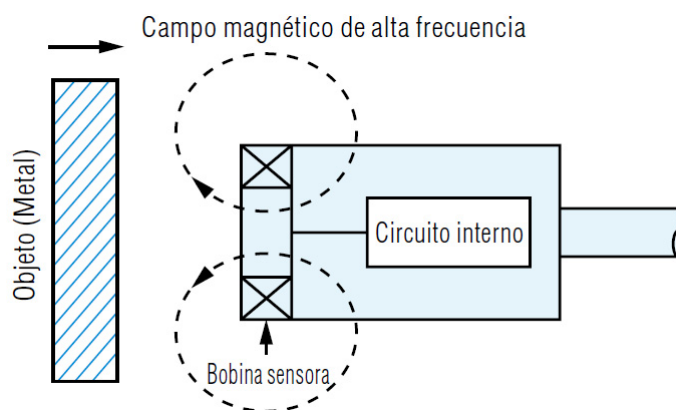


Figura 2.3: Esquema de funcionamiento de un sensor inductivo.

El uso de sensores de ultrasonidos para medir distancias es ampliamente utilizado en muchos campos y sectores. Se puede ver implementado en vehículos autónomos para la detección de obstáculos o en robótica, como detectores de posición o separación entre elementos, incluso en medios contaminados donde la visión es limitada por el ambiente y/o gases de baja densidad [15],[16].

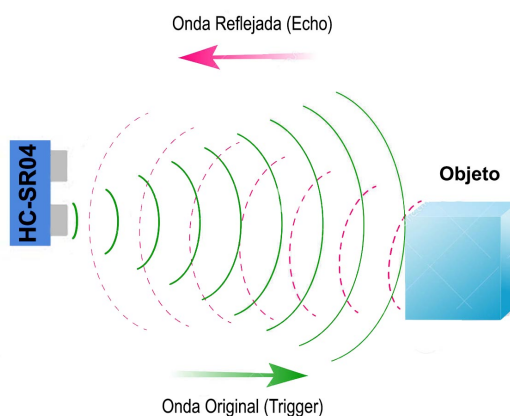


Figura 2.4: Esquema de funcionamiento de un sensor de ultrasonidos.

En los sensores ultrasónicos, cuyas mediciones se basan en el tiempo de vuelo (ToF), se cubren distancias máximas del orden de 3 - 4 metros al obstáculo. Estas medidas generalmente permiten una precisión del orden de centímetros, que se puede mejorar aumentando el valor de la frecuencia (pulsos emitidos). Sin embargo, por encima de 50 kHz, existen fuertes limitaciones debido a la absorción atmosférica y otro tipo de interferencias. Una de sus ventajas, es tener un ángulo de haz ancho.

Otra forma para aumentar sustancialmente la precisión de la medición, es mediante la emisión de una señal continua, que permita medir las variaciones de fase de forma constante. El análisis de la variación de fase (PSA) podría determinarse mediante el cálculo de la Transformada de Fourier discreta (DFT). Esta metodología utilizada generalmente para el estudio de distancias por debajo de la longitud de onda de la señal utilizada, permite mejorar la precisión hasta valores que están por debajo de los milímetros.

En trabajos como: *Accurate distance measurement by an autonomous ultrasonic system combining time-of-flight and phase-shift methods*, [17] se presenta una combinación de los métodos de ToF y PSA en sensores de ultrasonidos. La evaluación del tiempo de vuelo (ToF) se lleva a cabo calculando la correlación cruzada entre las señales de ultrasonidos transmitidas y recibidas. La variación de fase (PSA) se determina calculando la transformada de Fourier discreta (DFT). En este experimento, la precisión fue inferior a un

milímetro para una distancia de aproximadamente un metro. Sin embargo, el método propuesto en el trabajo citado [17] requiere un cálculo matemático complejo y costoso computacionalmente, por lo que necesita un microcontrolador de altas prestaciones para su funcionamiento.

2.1 Propuesta

Para el desarrollo de este trabajo se escoge usar los ultrasonidos como método para la medición de distancias. Y como dispositivo de medición, el módulo de ultrasonidos comercial HC-SR04 [18].

El problema principal que se detecta en [18], es la carga computacional y por ende las mayores prestaciones del microcontrolador. En este trabajo, se presentan varias modificaciones en el módulo de ultrasonidos, que se hacen para mejorar la precisión de las mediciones.

Además, para el estudio y mantenimiento de obras de arte, el método más apropiado a usar es el de ultrasonidos.

CAPÍTULO 3

Materiales y métodos

En este apartado, se van a describir tanto el proyecto desarrollado, como la metodología seguida para la medición de distancias con alta resolución. Para este trabajo se ha elegido el módulo HC-SR04 como sensor de distancias, por su eficacia relativamente buena, con un bajo precio. Este módulo es una pieza muy importante de nuestro dispositivo final.

Se emplea un robot cartesiano para el desplazamiento por el plano horizontal XY del dispositivo de medición y dos microcontroladores, un ATmega328 y un ATmega32u4. Éstos, están soportados por Arduino Nano y un Arduino Leonardo, respectivamente. El primero de ellos, es usado exclusivamente para el control del robot cartesiano. Mientras que el segundo, es usado para la medición de distancias. También se usan dispositivos electrónicos básicos como condensadores, resistencias y puertas lógicas.

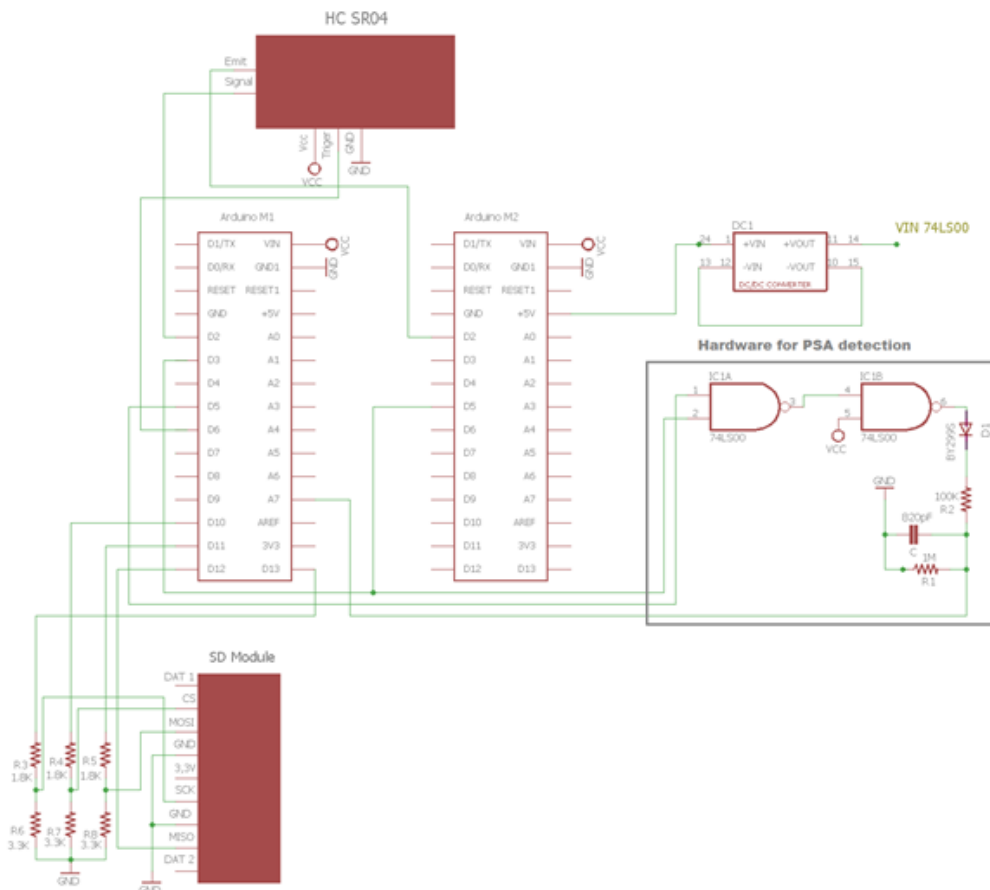


Figura 3.1: Esquema final del sensor de ultrasonidos.

El sistema de medición está controlado por el microcontrolador ATmega32u4, desde ahora **M1**. En la figura 3.1 se puede observar el dispositivo desarrollado. Por una parte, se aprecia, un módulo de ultrasonidos (HC-SR04), un hardware analógico para medir el desfase de ondas, desde ahora PSA, compuesto por dos puertas lógicas, un diodo, un condensador y dos resistencias. Por otra parte, dos divisores resistivos necesarios para el circuito. Realizando modificaciones en el hardware del módulo de ultrasonidos y desarrollando un programa, basado en los *timers* de los microcontroladores, se consigue mejorar la precisión de las mediciones que realiza dicho módulo.

El sistema de movimiento del robot cartesiano está controlado por el microcontrolador ATmega328, desde ahora **M2**.

3.1 Microcontroladores ATmega328 y ATmega32u4

Para este proyecto, se seleccionan los microcontroladores ATmega328 y el ATmega32u4. Están disponibles en el mercado a un bajo precio y se ofrecen como una solución con muchas funcionalidades para diversos tipos de proyectos. Además, ambos se encuentran implementados en la plataforma Arduino.

Arduino es una plataforma *open source* [19] y *open hardware* [20]. Está basado en una sencilla placa con entradas y salidas, analógicas y digitales. Controlado todo por un microcontrolador integrado específico, que puede ser programado desde el IDE de Arduino, que es el entorno de desarrollo. Arduino se emplea como procesador principal en muchos tipos proyectos alrededor del mundo. Una gran comunidad detrás de esta plataforma sirve como soporte y desarrollo de sus funcionalidades. Al ser *open source* y *open hardware*, cualquiera con los conocimientos necesarios, puede moldear a su gusto o necesidad, tanto el software como el hardware. Y si se quiere compartirlo con la comunidad, contribuyendo así al desarrollo de funcionalidades y características de la plataforma Arduino.

En las figuras 3.2 y 3.3 , se muestran dos de los Arduinos empleados en el proyecto.

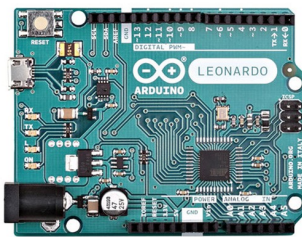


Figura 3.2: Arduino Leonardo

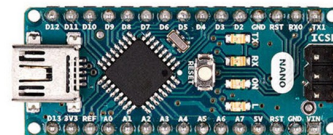


Figura 3.3: Arduino Nano

En cuanto a las características técnicas de los microcontroladores se puede destacar que la memoria programable del ATmega328 y del ATmega32u4 es de 32 KB, suficiente para almacenar todas las bibliotecas y los programas necesarios. Además, está equipado con entradas y salidas digitales, así como analógicas moduladas por PWM, explicadas en el apartado 3.1.2 . Las interrupciones externas (explicadas en el apartado 4.2.4) son necesarias en este proyecto, pero sólo se usan en el microcontrolador ATmega32u4. El cual, tiene 5 interrupciones disponibles configurables para activarse en el cambio del valor del voltaje, ya sea por flanco ascendente, descendente o un cambio de valor.

Microcontrolador	ATmega32u4
Tensión de funcionamiento	5 V
Voltaje de entrada (recomendado)	7 – 12 V
Voltaje de entrada (límites)	6 – 20 V
Pines digitales I/O	20
Canales PWM	7
Canales de entradas analógicas	12
DC Corriente por I/O pin	40 mA
DC Corriente por I/O pin	50 mA
Memoria flash	32 KB (ATmega32u4), 4 KB
SRAM	2,5 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Velocidad de reloj	16 MHz
Longitud	68,6 mm
Anchura	53,3 mm
Peso	20 g

Tabla 3.1: Especificaciones técnicas del Arduino Leonardo.

Microcontrolador	ATmega328
Arquitectura	AVR
Tensión de funcionamiento	5 V
Memoria flash	32 KB, 2 KB
SRAM	2 KB
Velocidad de reloj	16 MHz
Pines digitales I/O	8
EEPROM	1 KB
DC Current por I/O pins	40 mA (I/O pins)
Voltaje de entrada	7 – 12 V
Pines digitales I/O	22
PWM Output	6
Poder de consumición	19 mA
Tamaño PCB	18 x 45 mm
Peso	7 g
Código del producto	A000005

Tabla 3.2: Especificaciones técnicas del Arduino Nano.

En las tablas 3.1 y 3.2 se pueden observar las características técnicas proporcionadas por Arduino para cada microcontrolador.

3.1.1. Problemas con el microcontrolador ATmega32u4

Aunque el microcontrolador ATmega32u4 sea una opción muy económica y versátil para una gran variedad de proyectos, presenta problemas a la hora de utilizar el módulo HC-SR04 de la forma especificada por el fabricante.

Para realizar las mediciones, en última instancia, el módulo comercial HC-SR04 (tal y como está diseñado) nos envía un pulso, por su pin **ECHO**. El tiempo que la señal de este pin esté en HIGH (5 V), es igual al tiempo que ha tardado una onda en realizar su recorrido en micro segundos desde el emisor hasta el receptor, es decir, el ToF.

El microcontrolador ATmega32u4 debería poder contar este tiempo para luego realizar sus cálculos. Esto da lugar a un problema, ya que al contar el tiempo que dura el pulso que se recibe por el pin ECHO presenta una tasa de error bastante alta. Por ejemplo, para una distancia de 9,00 cm se recibe como respuesta valores en un intervalo de 498 - 549 ms.

A una temperatura ambiente de 20° la velocidad del aire es de 343,3 m/s. Para calcular la distancia se emplea la fórmula explicada en el apartado 3.2.2 en el punto 9 de la enumeración.

$$\text{EspacioReal} = 0,01716(\text{cm/us}) \times \text{tiempo}$$

Por lo que la distancia medida, está entre el intervalo 8,54568 - 9,42084 cm. Esto añade una incertidumbre adicional de error al resultado final.

A la vista de estos hechos se puede deducir que no se pueden medir tiempos de señales en un valor lógico específico, cuando se busca una precisión elevada con el microcontrolador ATmega32u4.

3.1.2. Generación de señales PWM y timers en ATmega32u4.

En este proyecto se necesita generar un tipo de señales específicas. Generar una señal digital con una frecuencia muy determinada y un *duty cycle* exacto, no es algo sencillo. Las funciones implementadas por Arduino no cumplen las características necesarias, ya que para generar una señal de estas características, se bloquea el procesador (el encargado de producir la señal se delega en la CPU). Se busca trasladar la carga de esta función al hardware interno del procesador.

Se descarta el uso de *analogWrite()*, ya que solo permite la generación de frecuencias determinadas y en un rango muy bajo. También, usar *tone()*, ya que no permite generar varias señales a la vez, entre otros problemas.

La opción más factible para reproducirlas es usar el principio básico de generación de las señales PWM, en inglés *pulse width modulation*, o modulación de ancho de pulso, desde ahora PWM.

3.1.2.1. Descripción de las señales PWM

La gran mayoría de microcontroladores de propósito general, no pueden generar una señal analógica real. El microcontrolador ATmega32u4 no es una excepción. Únicamente, genera salidas digitales, que corresponden a HIGH o LOW, en este microcontrolador Vcc (5 V) y GND (0 V), respectivamente.

PWM es una técnica para obtener señales pseudo-analógicas, a partir de señales digitales. Su funcionamiento consiste en modificar el ciclo de trabajo (*duty cycle*) de una señal periódica para controlar la cantidad de voltaje que se envía a una salida. Como se muestra en la figura 3.4, consiste en activar y desactivar una salida digital durante tiempos determinados. El promedio de la tensión de salida es igual al valor analógico deseado.

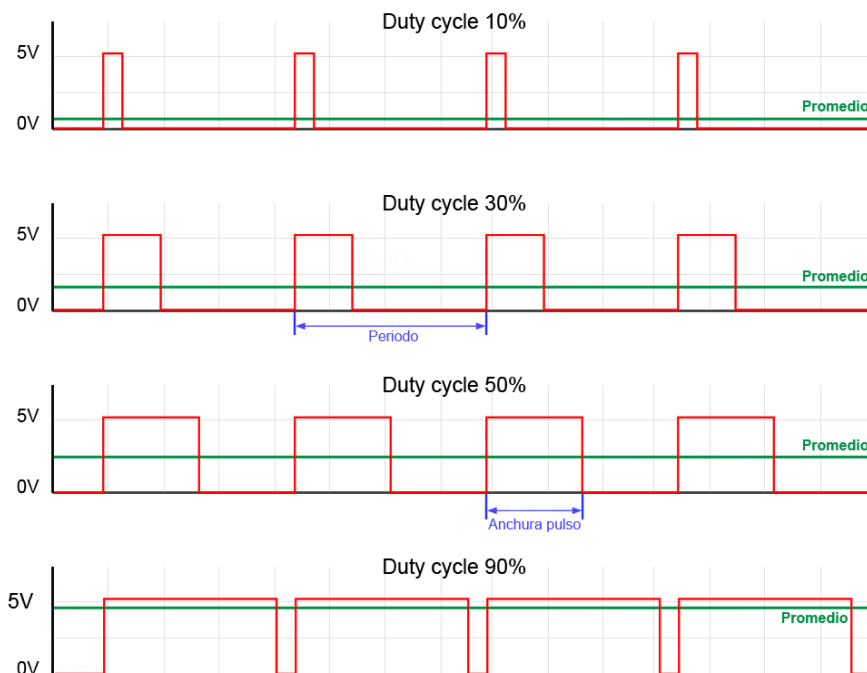


Figura 3.4: Señales PWM.

Solo en algunos pines de la placa de Arduino se permiten generar estas señales pseudo-analógicas, usando la función `analogWrite()`. Pero, hay que tener claro que la función no proporciona una salida analógica real, sino una señal PWM.

3.1.2.2. Timers y generación de señales PWM

Las señales PWM en el microcontrolador ATmega32u4, se generan por hardware utilizando timers. De esta forma, la CPU no utiliza apenas sus recursos en la generación de la señal.

El microcontrolador ATmega32u4 tiene cinco timers, desde el timer 0 al timer 4. Cada timer tiene un número limitado de señales PWM que puede generar. Existen distintas técnicas para generar una señal PWM haciendo uso de éstos, como la basada en pendiente simple, *single slope PWM* o la basada en pendiente doble PWM, *dual slope PWM*.

En la primera de ellas, el timer se basa en un contador con registros de comparación asociados. El contador incrementa su valor en cada flanco activo del reloj, es decir, por cada ciclo de reloj se incrementa el contador del timer en uno. Una vez que se alcanza el valor máximo, el contador vuelve a cero y el proceso se repite. El contador tiene un comparador asociado, el cual está vinculado a una señal de salida digital. Ésta tiene un valor HIGH, si el contador está por debajo o es igual al valor del registro de comparación y LOW, si está estrictamente por encima del registro de comparación.

Este comportamiento provoca que para cada registro de comparación asociado al contador, se genere una señal PWM, cuyo duty cycle depende del valor marcado en el comparador. Todas las señales generadas con el mismo timer tienen la misma frecuencia.

En la figura 3.5 se pueden observar dos señales PWM de la misma frecuencia, pero distinto duty cycle, generadas gracias a un único contador y dos comparadores. El inicio de los pulsos generados es el mismo.

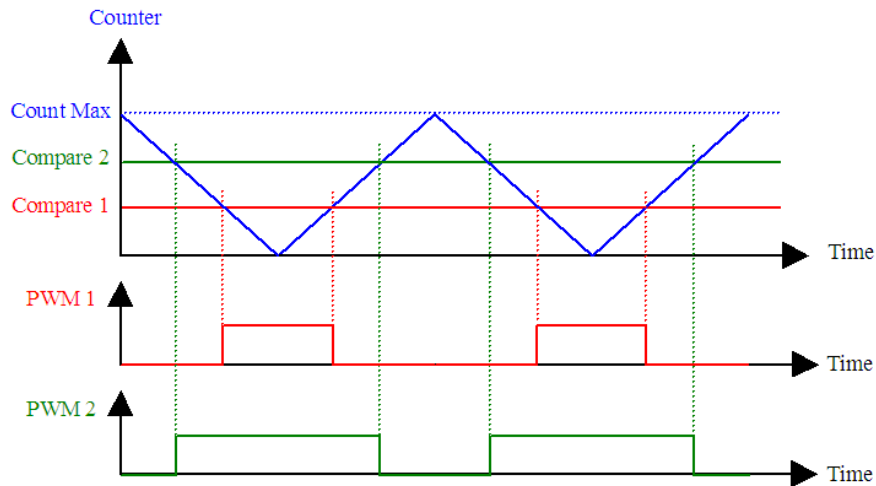


Figura 3.5: Pendiente simple PWM

En la segunda técnica, se puede configurar el timer para que el contador incremente su valor en cada flanco activo del reloj. Una vez que se alcanza el valor máximo, se decrementa el contador en cada ciclo de reloj hasta llegar a cero. Al llegar a 0 el proceso completo se repite otra vez. El contador tiene un comparador asociado, el cual está vinculado a una señal digital de salida. Ésta, tiene un valor HIGH, si el contador está por debajo o es igual al valor del registro de comparación y LOW, si está estrictamente por encima del registro de comparación. En la figura 3.6, se pueden observar dos señales PWM de la misma frecuencia, pero distinto duty cycle, generadas gracias a un único contador y dos comparadores. Las señales PWM generadas se sincronizan en el centro de los pulsos generados.

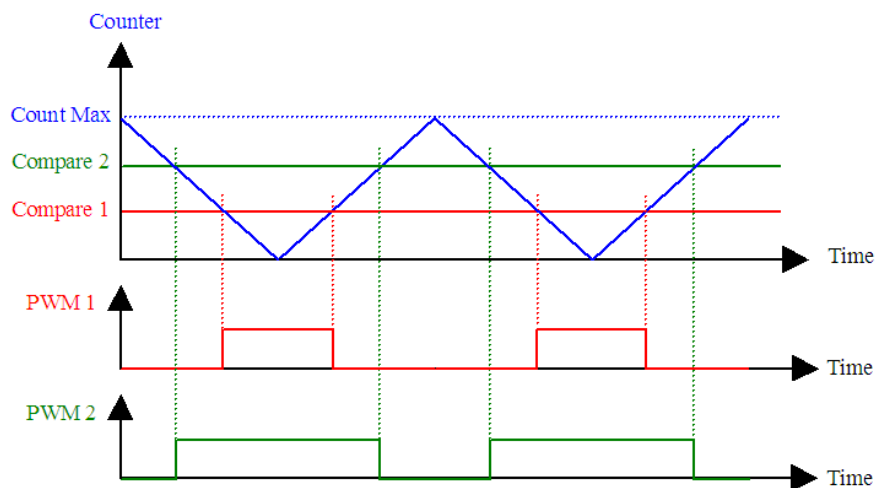


Figura 3.6: Pendiente doble PWM

3.1.2.3. Manipulación de Timers

Este apartado no pretende ser una explicación extensa del cómo funcionan y se manipulan los timers. Toda la documentación se puede encontrar en la hoja de especificaciones del microcontrolador Atmega32u4. En el Anexo I, se puede encontrar la documentación específica en relación con estos timers.

Los timers 1 y 3, tienen implementados contadores de 16 *bits*. Esto significa que pueden contar valores desde 0 hasta 65535 antes de desbordar¹. Estos timers pueden configurarse para 12 distintos modos de operación, con diferentes frecuencias de reloj. La frecuencia del timer es igual a la frecuencia del reloj del sistema (F_{CLK}). Alternativamente, una de las cuatro tomas del preescalador se puede usar como frecuencia del timer. El reloj preescalado puede tener una frecuencia de $F_{CLK}/8$, $F_{CLK}/64$, $F_{CLK}/256$ o $F_{CLK}/1024$.

Dependiendo del modo de operación y la frecuencia establecidas, el timer se comporta de una forma u otra, el contador desborda en un valor u otro y el modo de operación se establece en *single slope* o *dual slope*. La configuración del timer 1 y 3, queda determinada por los registros TCCR1A, TCCR1B, TCCR3A, TCCR3B.

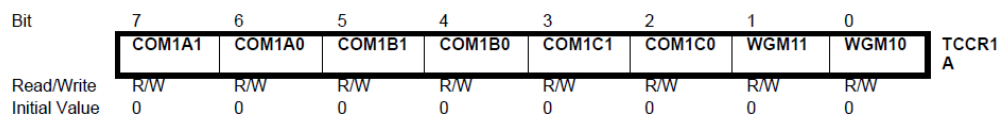


Figura 3.7: Timer/Counter1 Control Register A - TCCR1A

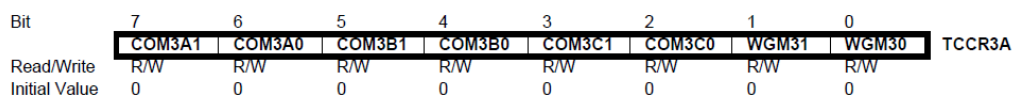


Figura 3.8: Timer/Counter1 Control Register A - TCCR3A

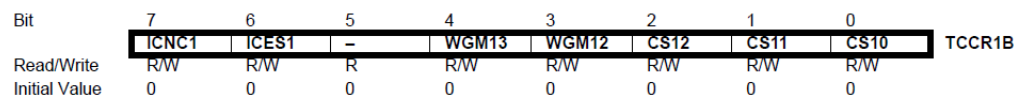


Figura 3.9: Timer/Counter1 Control Register B - TCCR1B

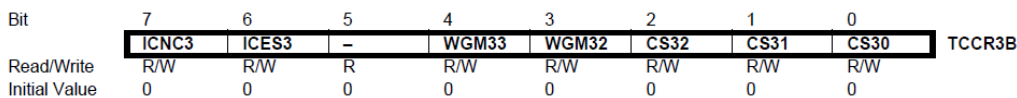


Figura 3.10: Timer/Counter1 Control Register B - TCCR3B

Los bits 0, 1 y 2 de TCCRnB (CSn0, CSn1 y CSn2), siendo n igual a 1 o 3, establecen las opciones de reloj de acuerdo en la tabla 3.3 :

¹https://es.wikipedia.org/wiki/Desbordamiento_de_b%C3%BAfer

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer Counter stopped)
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Tabla 3.3: Distintas configuraciones del preescalador.

Los bits 0 y 1 de TCCRnA (WGMn0 y WGMn1) y los bits 3 y 4 de TCCRnB (WGMn2 y WGMn3) establecen el modo de funcionamiento del timer. Como se muestra en la tabla 3.4.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	T0Vn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM,Phase COrrrect, 8 bit	0xFFFF	TOP	BOTTOM
2	0	0	0	1	PWM,Phase COrrrect, 9 bit	0x00FF	TOP	BOTTOM
3	0	0	1	1	PWM,Phase COrrrect, 10 bit	0x01FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immedate	MAX
5	0	1	0	1	PWM,Fast PWM, 8 bit	0x00FF	TOP	TOP
6	0	1	1	0	PWM,Fast PWM, 9 bit	0x01FF	TOP	TOP
7	0	1	1	1	PWM,Fast PWM, 10 bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM,Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	1	0	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	TOP	TOP
15	1	1	1	1	Fast PWM	OCRnA	TOP	TOP

Tabla 3.4: Configuraciones del timer.

Cada uno de los timers tiene tres canales de comparación, es decir, se pueden generar 3 señales PWM distintas A, B y C. El modo de funcionamiento establecido del timer y los bits COMnA1:0, COMnB1:0, y COMnC1:0, controlan las salidas y el comportamiento de las señales OCnA, OCnB, y OCnC.

En este proyecto se usa el modo *Fast PWM*. La tabla 3.5 describe el comportamiento del canal.

COMnA1/COMnB1/COMnC0	COMnA0/COMnB0/COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 OR 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OC1A/OC1B/OC1C on compare match, set OC1A/OC1B/OC1C at TOP
1	1	Set OC1A/OC1B/OC1C on compare match, clear OC1A/OC1B/OC1C at TOP

Tabla 3.5: Comportamiento del canal de salida.

Para este proyecto se necesita generar señales de 40 KHZ con un duty cycle del 50 % por cualquier pin. Para conseguir esto se ha hecho lo siguiente.

1. Se selecciona el preescaler 8, poniendo el bite CSn1 a 1.
2. Se establece el modo del funcionamiento del timer al 15 (Fast PWM), poniendo los bites WGMn0, WGMn1, WGMn2 y WGMn3 a 1.
3. Únicamente, se activa el canal B de salida en modo 3, poniendo el bite COMnB1 a 1. Esto hace que, cuando el contador sea igual al registro del comparador de B, se establece la señal de salida a LOW (0V), y cuando el contador sea igual a TOP (OCRnA) se establece a HIGH (5V).
4. Se establece TOP, como se quiere una frecuencia de 40 KHz, teniendo en cuenta que el procesador es de 16 MHz y se escoge un preescaler de 8, se tiene:

$$F_{pwm} = F_{clk} / (TOP - 1) \Rightarrow TOP = \left(\frac{F_{clk}}{F_{pwm}} - 1 \right)$$

$$TOP = \left(\frac{16000}{8} \right) - 1 \Rightarrow TopValue = 49$$

Es decir, el registro OCRnA (TOP) debe ser igual a 49, así se establece una frecuencia en el timer de 40 KHz

5. Se establece el valor del comparador OCRnB, como se quiere un duty cycle del 50 %, se tiene que:

$$OCRnB = \left(\frac{TOP + 1}{2} \right) - 1$$

De esta forma se consigue un timer que tiene un comportamiento como el que se muestra en la figura 3.11. Con el que se obtiene una señal de 40 KHz con un duty cycle del 50 % por el canal B.

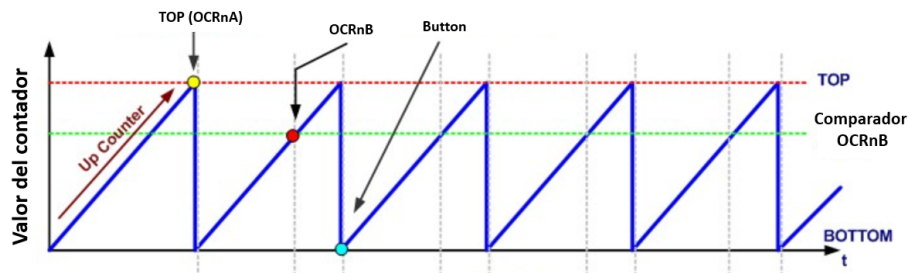


Figura 3.11: Configuración final del timer.

3.2 Módulo de ultrasonidos

Uno de los objetivos más importantes de este trabajo es desarrollar un dispositivo para medir distancias modificando drásticamente el módulo HC-SR04. Por lo que se eliminan varios de sus componentes, aumentando así la complejidad del proyecto, a la vez que se mejora la comprensión del funcionamiento de este dispositivo comercial.

3.2.1. Descripción básica

El módulo comercial HC-SR04 es un sensor de distancias que funciona con dos transductores de ultrasonidos de 40 KHz. Uno para la emisión y otro para la recepción de un pulso de ultrasonidos, que genera y recibe el mismo módulo. De esta forma, puede detectar objetos y calcular la distancia a la que se encuentran en un rango de 2 a 450 cm con un error de 3 mm (según el fabricante). Las figuras 3.12 y 3.13 muestran las vistas del módulo HC-SR04.

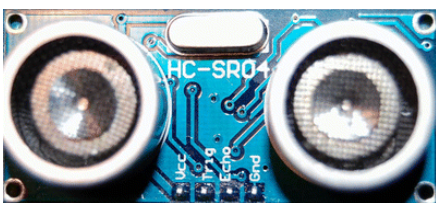


Figura 3.12: Front View

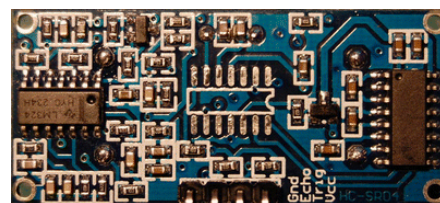


Figura 3.13: Back View

3.2.2. Funcionamiento, datos técnicos y especificaciones

En la figura 3.14 se muestra el esquema del módulo HC-SR04.

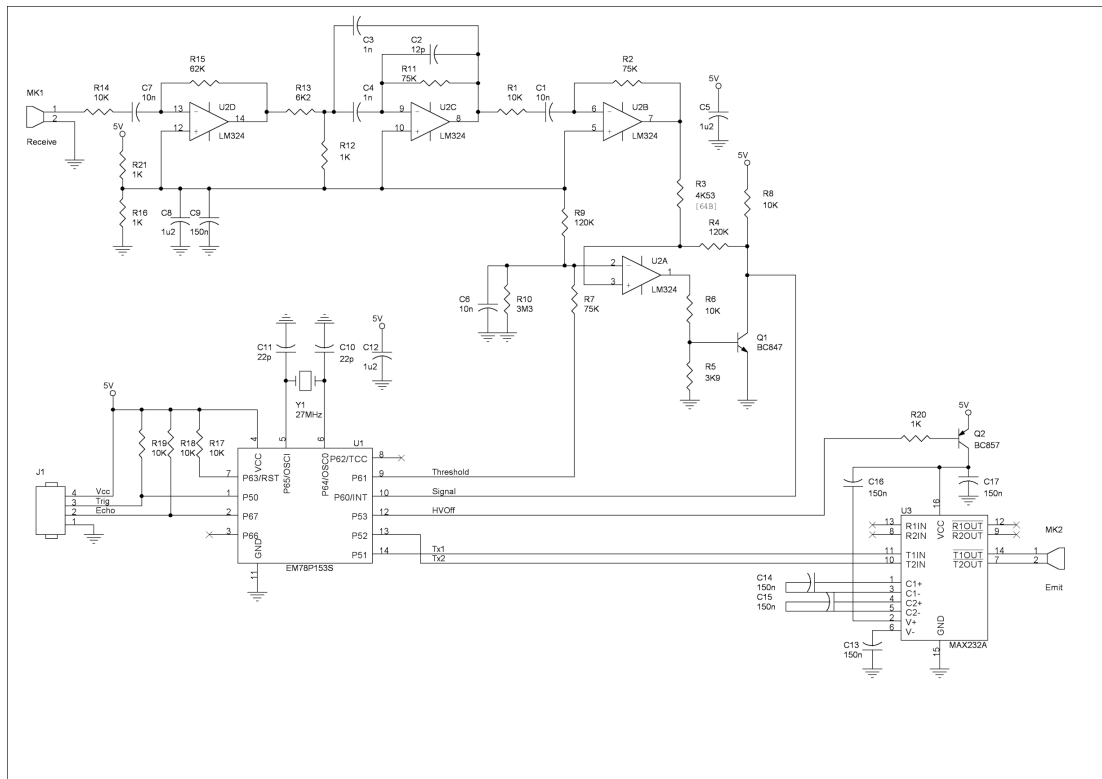


Figura 3.14: Módulo HC-SR04 Schematic

El circuito del esquema cuenta con dos transductores de ultrasonidos, uno para la emisión **MK2** y otro para la recepción **MK1**. Para generar los pulsos de ultrasonidos se necesita un voltaje relativamente alto. Para ello, se usa el integrado **MAX232A** [21] que puede generar $\pm 10\text{ V}$, a partir, de entradas de 5 V . El transductor emisor, **MK2**, se conecta entre dos salidas invertidas del **MAX232A** (pines 14 y 7) por lo que en realidad el transductor está alimentado a 20 V , voltaje necesario para funcionar correctamente.

Para la recepción de los ultrasonidos se usa el chip **LM324** [22], que contiene cuatro amplificadores operacionales (OPAMPs). El primer operacional **U2D** es un simple amplificador lineal que quita la señal continua (filtro paso alto debido a **C7**). El segundo **U2C**, es un filtro paso-banda. Modificando **R12** se puede cambiar, de forma fácil, la frecuencia de corte del filtro paso-banda. El siguiente operacional **U2B**, es nuevamente un amplificador que quita la señal continua, igual que el primero. El último **U2A**, es un comparador con histéresis, donde el umbral de conmutación se debe regular modificando la señal **Threshold** (proporcionada por el microcontrolador **EM78P153S** [23] por su pin 9), siendo la salida del transistor una señal de saturación, corte del mismo.

El funcionamiento secuencial y simplificado para entender este módulo es el siguiente:

1. Se envía una señal (pulso alto de 5 V) de al menos $10\text{ }\mu\text{s}$ por el pin **TRIGGER** del módulo HC-SR04 (conectado al pin 1 del **EM78P153S**) con esto se consigue activar el microcontrolador **EM78P153S**.

2. Una vez activado el microcontrolador **EM78P153S**, éste genera un conjunto de señales para poner en funcionamiento el módulo. En el apartado 4.2.1 se explica con detalle las señales que emite y la función principal del microcontrolador **EM78P153S**.
3. Cuando el integrado **MAX232A** reciba las señales **Tx1** y **Tx2**, excita el transductor emisor 8 veces, generando así un tren de 8 pulsos ultrasónicos a 40 KHz.
4. El microcontrolador **EM78P153S** establece el pin **ECHO** del módulo de ultrasonidos (pin 2 del **EM78P153S**) en un nivel alto (5V), se pondrá en un nivel bajo (0V) más tarde.
5. Cuando se detecte un obstáculo en el camino de los ultrasonidos se produce una reflexión y se recibe el rebote del tren de pulsos de 40 KHz por el transductor receptor.
6. Los amplificadores operacionales son usados para mejorar la señal de recepción. Esta señal analógica es importante para todo el proyecto y se lee por el microcontrolador **EM78P153S** por su pin 10.
7. El pin **ECHO** es un canal de salida del módulo y se mantiene en un nivel alto (5 V) hasta recibir el eco reflejado por el obstáculo, es decir, cuando se detecte la señal de recepción por el pin 10 del **EM78P153S**, el módulo pone su pin **ECHO** en un nivel bajo (0V). El tiempo que se mantiene en un nivel alto, es el tiempo que han tardado los ultrasonidos en regresar al transductor receptor desde el transductor de emisión.
8. La distancia a la que se encuentra el objeto detectado se puede averiguar gracias a la duración del pulso que se emite por el pin **ECHO**
9. Con un movimiento rectilíneo del tren de pulsos ultrasónicos, podemos calcular la distancia a la que se encuentra el objeto mediante la siguiente fórmula:

$$\text{EspacioReal} = \frac{\text{velocidad} \times \text{tiempo}}{2} \Rightarrow \text{EspacioReal} = \frac{343,2(m/s) \times \text{tiempo}}{2}$$

$$\text{EspacioReal} = 171,6(m/s) \times \text{tiempo} \Rightarrow \text{EspacioReal} = 0,01716(cm/us) \times \text{tiempo}$$

A una temperatura ambiente de 20°C la velocidad del aire es de 343,3 m/s. Se divide por dos, ya que el espacio que se calcula con $\text{espacio} = \text{velocidad} \times \text{tiempo}$ es el camino que realiza la onda en su camino de ida y vuelta.

3.2.3. Problemas y limitaciones.

3.2.3.1. Problema con la temperatura y condiciones de medida.

El módulo HC-SR04 presenta una resolución pobre en comparación con la que se espera conseguir, una resolución de como mínimo medio milímetro. Al analizar las características de este módulo en distancias cortas desde 7,00 cm hasta 20,00 cm se obtienen unos resultados que se alejan de los deseados. Su resolución, según el fabricante, es de 3 mm [18]. No obstante, la resolución medida experimentalmente es peor siendo aproximadamente de 8 mm.

Además, este módulo no tiene la capacidad de medir la temperatura, característica a tener en cuenta a la hora de medir la distancia a la que se encuentra un objeto. En condiciones de laboratorio a 20°C y un 50 % de humedad, la velocidad del sonido es de 343,2 m/s. Sin embargo, la velocidad del sonido incrementa, aproximadamente, 0,6(m/s) por cada grado centígrado que aumenta la temperatura ambiente.

3.2.3.2. Solución a la temperatura y condiciones de medida.

Este problema se puede solventar haciendo mediciones únicamente en un laboratorio a 20°C y con una humedad ambiente del 50 %.

Se podría haber optado por implementar en el proyecto un sensor de temperatura, pero uno que tenga una sensibilidad tan alta como la que se necesita, elevaría el precio del proyecto.

Siendo, la medición en laboratorio, la mejor opción, ya que se evitan más variables en el sistema y carga computacional al microcontrolador, que esto conlleva.

3.2.3.3. Problema con la señal de recepción del módulo

Para detectar uno de los problemas principales del módulo tenemos que analizar las señales que produce entre diferentes distancias cercanas. Se coloca una superficie plástica plana frente al módulo HC-SR04 a una distancia de 10,00 *cm*. La superficie plana está controlada por un tornillo micrométrico, por lo que puede acercarse o alejarse 0,25 *mm* del punto en el que se encuentre según se desee.

Como se observa en la figura 3.15 se conecta a un osciloscopio el transductor emisor, con esto se tiene la señal que lo excita y el momento en el que esto ocurre. Se llama a esta señal **emit**. La señal de emisión **emit**, tal y como el fabricante señala en el *datasheet*, es de 8 pulsos a una frecuencia de 40 *Khz* que dura 200 *us* microsegundos.

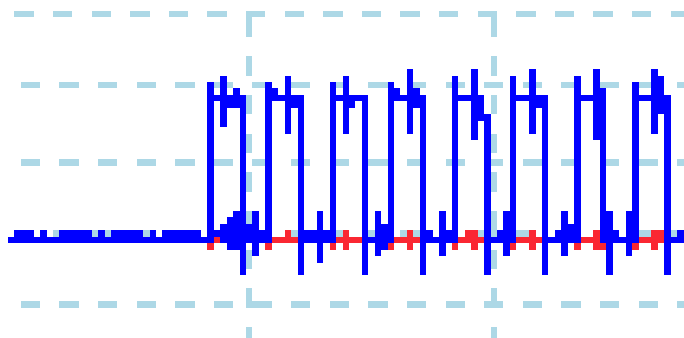


Figura 3.15: Señal emit

Del transductor receptor interesa al final, la señal amplificada y sin ruidos que se genera gracias a los cuatro amplificadores operacionales (OPAMPs) del LM324. Esta señal es **signal**, que se puede observar en la figura 3.7, que también se conecta al osciloscopio, para poder analizarla y estudiarla. La señal que genera el receptor y que seguidamente es procesada, por los amplificadores operacionales, **signal**, tiene una naturaleza analógica y no digital como se suponía teóricamente. No tiene una frecuencia de 40 *Khz*, sino una frecuencia ligeramente diferente.

En la figura 3.16, la señal roja, describe el comportamiento real de **signal** y la señal azul, describe el comportamiento teórico.

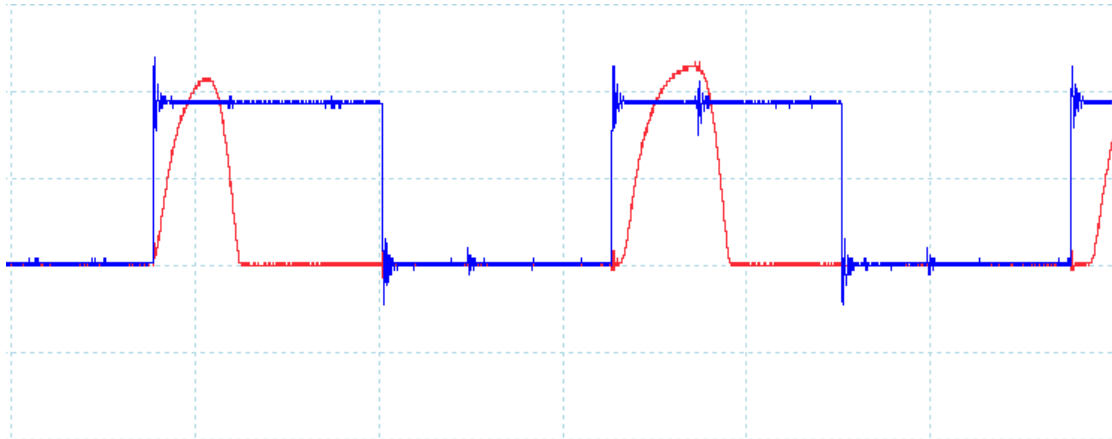


Figura 3.16: Señal teórica y real de **signal**

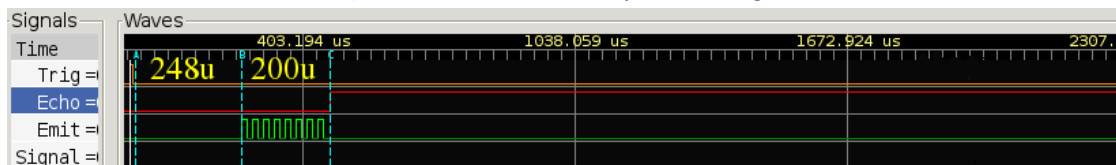


Figura 3.17: Señal para la emisión de ultrasonidos

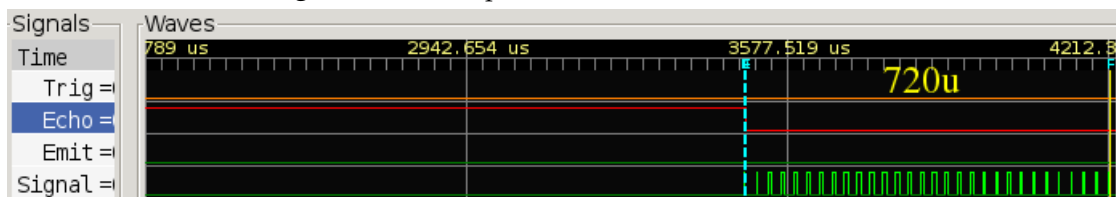


Figura 3.18: Señal de recepción de ultrasonidos

Al realizar varias mediciones en la misma posición no se produce ninguna variación en el ToF, ni en la señal recepción analógica que recibe el sensor de ultrasonidos. Si se acerca $0,25\text{ mm}$ la superficie al módulo de ultrasonidos se puede observar que la señal de emisión **emit** no varía (como era de esperar), pues es el origen. Pero si, se ve modificado el momento en el que se genera la señal de recepción. Esto ocurre, ya que si se acerca la superficie al módulo, el tiempo que tardan las ondas de ultrasonidos desde que se generan en el emisor hasta que son recogidas en el receptor, es menor. Este comportamiento es el esperado (totalmente normal), puesto que a menor distancia a recorrer menor es el tiempo que se necesita.

Una observación importante es que la señal de recepción **signal** es ligeramente diferente. Se detecta que la potencia a la que se recibe el primer y segundo pulso de ultrasonido es distinta. Para que se vea de una forma más gráfica y por tener una referencia temporal, se simula la señal de color azul (de las figuras 3.19 - 3.28) que es una onda cuadrada continua a 40 KHz con un 50% de *duty cycle*. Esta señal se genera en el mismo momento que **emit**, por lo que es el instante de la emisión de las ondas de ultrasonidos en el transductor emisor.

La recepción es capturada con el osciloscopio y se representa mediante la señal de color rojo.

Se coloca la superficie plana a una distancia de 10 cm del módulo y se va acercando la superficie $0,5\text{ mm}$ o $0,25\text{ mm}$ al módulo de ultrasonidos para poder estudiar el compor-

tamiento de la señal de recepción **signal**. En las siguiente figuras 3.19 - 3.28 se ve dicha evolución.

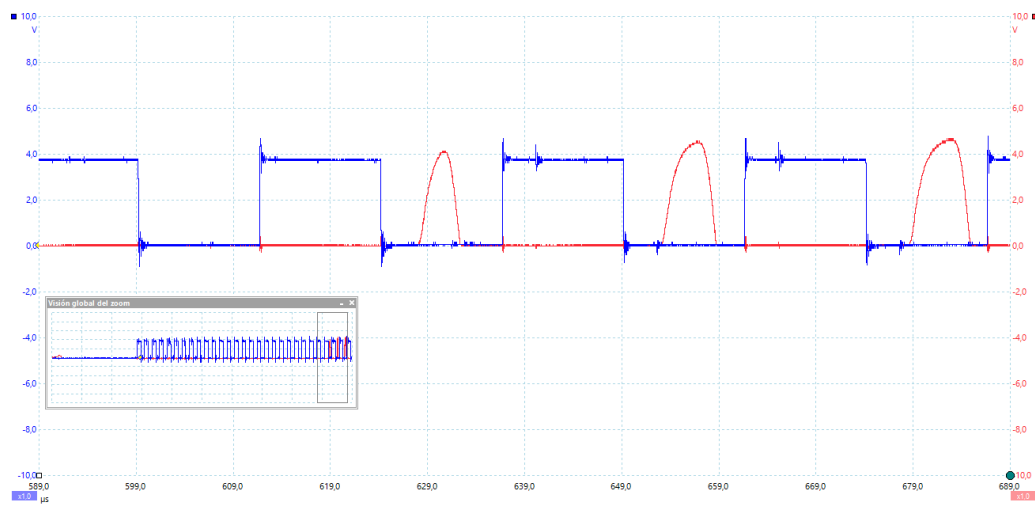


Figura 3.19: Distancia 100 mm

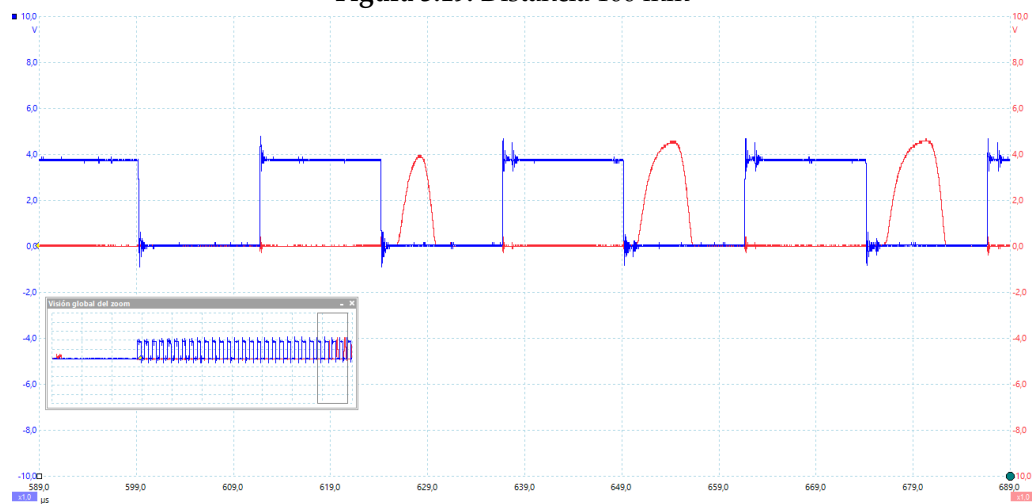


Figura 3.20: Distancia 100 mm - 0.5mm

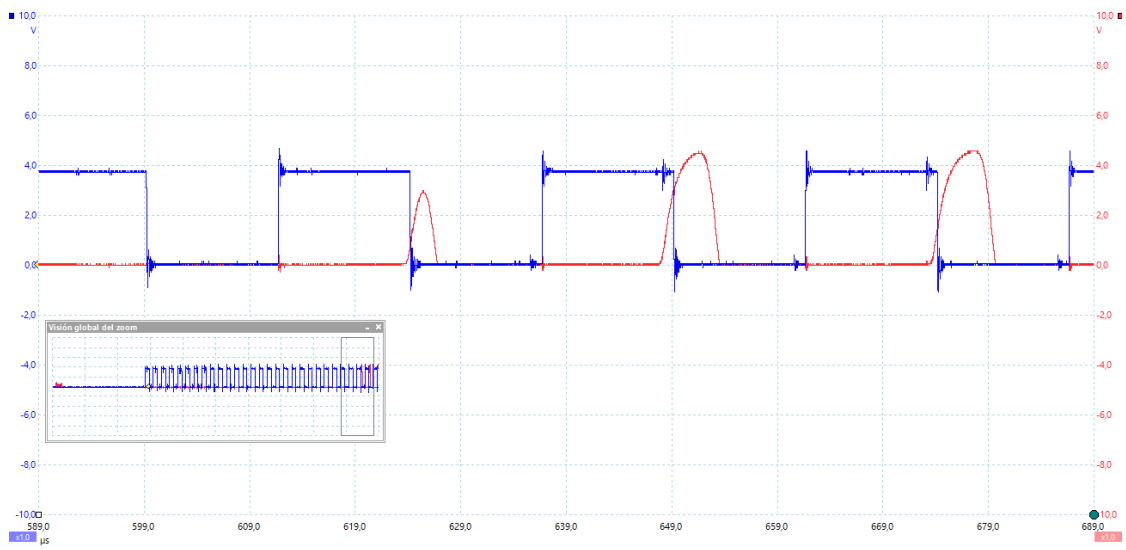


Figura 3.21: Distancia 100 mm - 1.0 mm

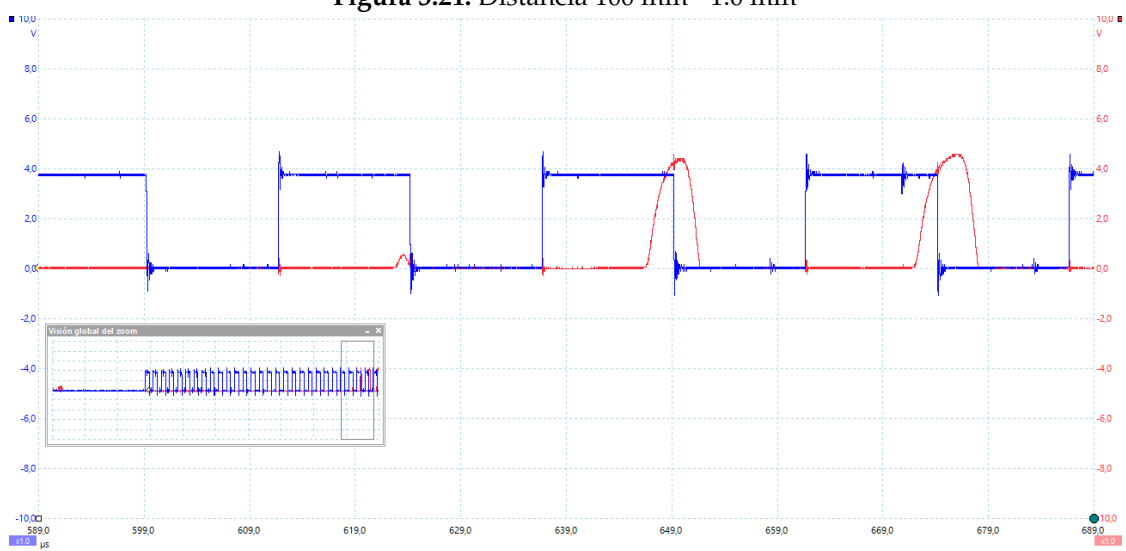


Figura 3.22: Distancia 100 mm - 1.25 mm

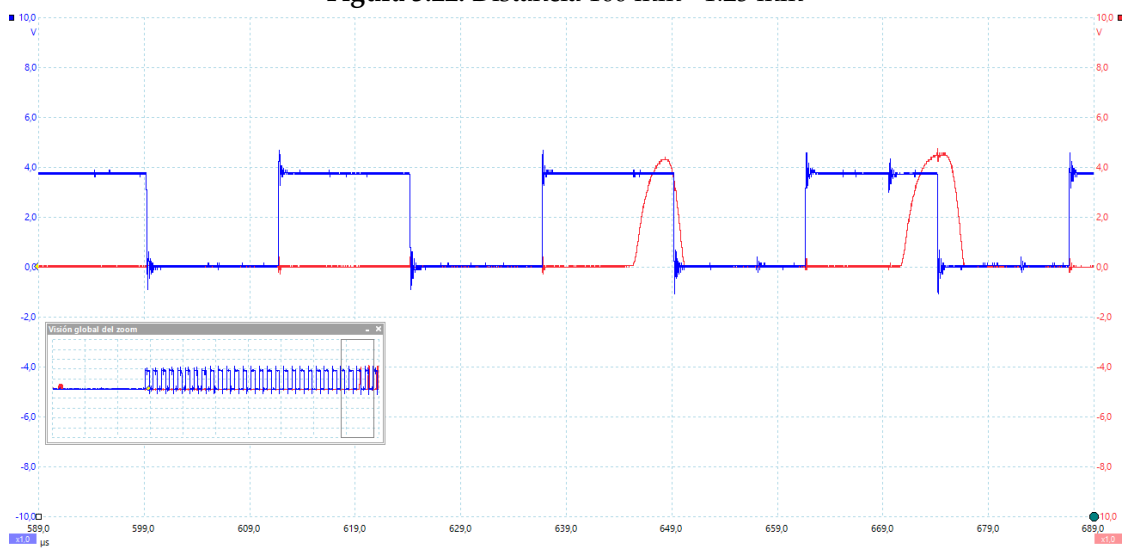


Figura 3.23: Distancia 100 mm - 1.50 mm

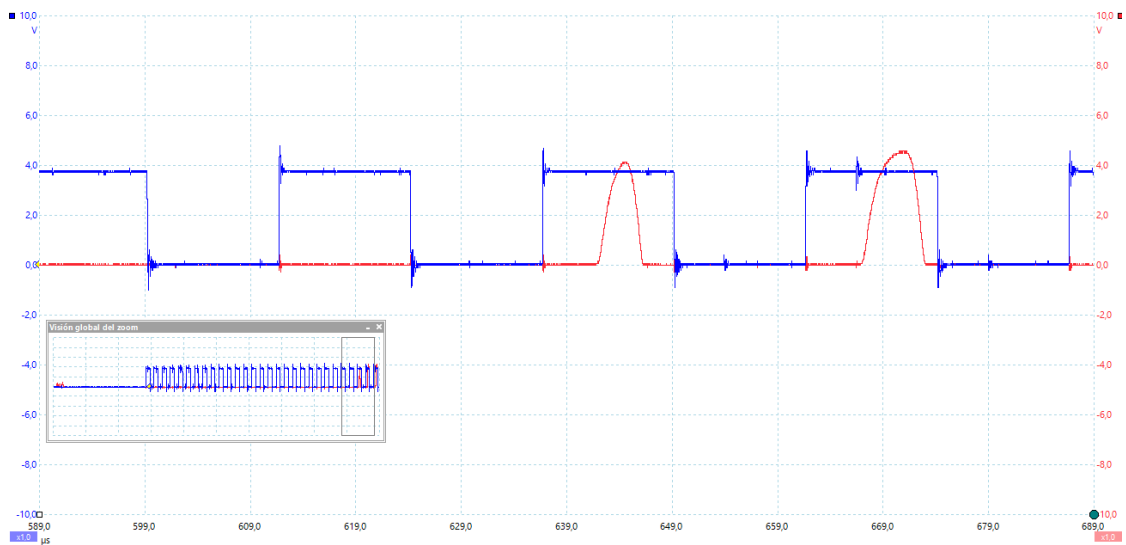


Figura 3.24: Distancia 100 mm - 2.0 mm

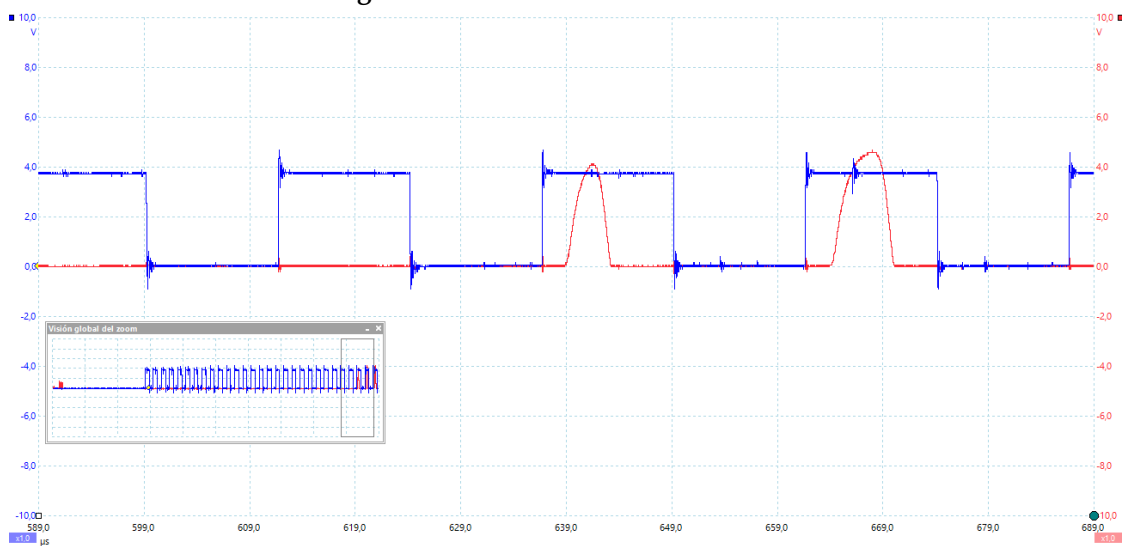


Figura 3.25: Distancia 100 mm - 2.5 mm

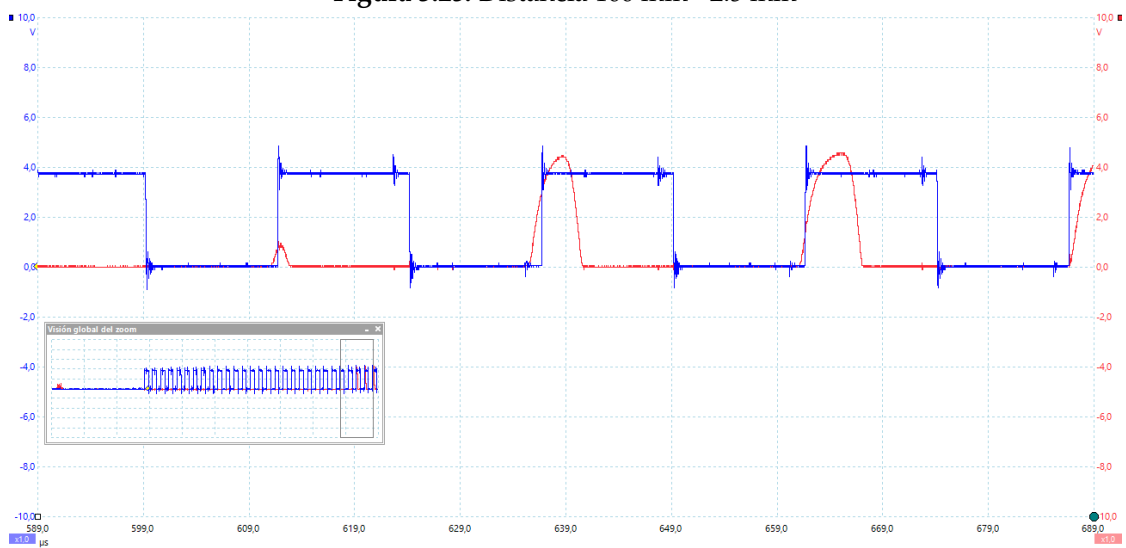


Figura 3.26: Distancia 100 mm - 3.0 mm

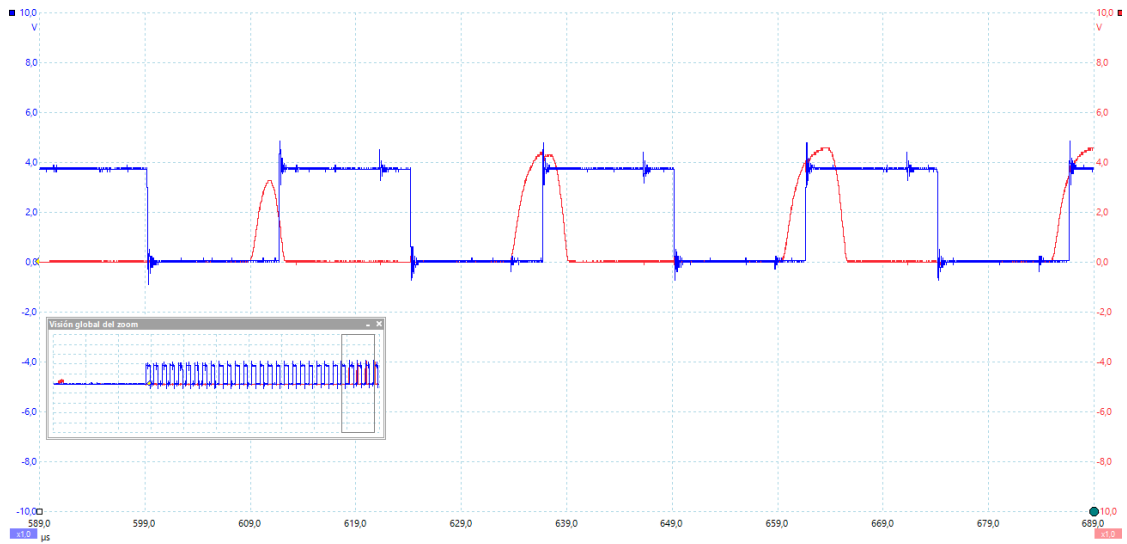


Figura 3.27: Distancia 100 mm - 3.25 mm

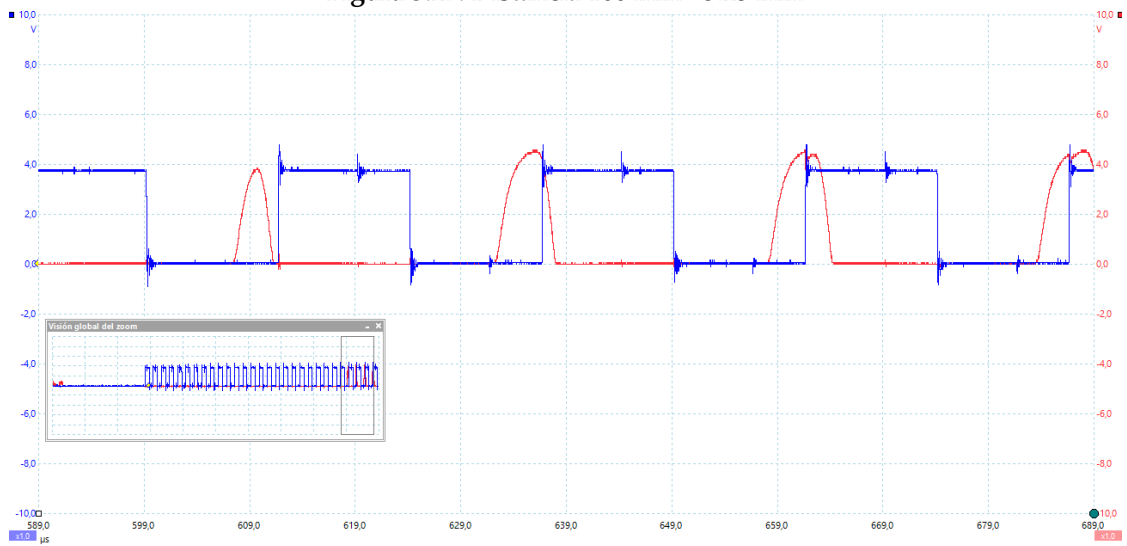


Figura 3.28: Distancia 100 mm - 3.50 mm

Se puede observar que cuando se acerca la superficie al módulo, los pulsos de ultrasonidos pierden potencia e incluso llegan a desaparecer (transición de las figuras distancia $100\text{ mm} - 0,5\text{ mm}$ a distancia $100\text{ mm} - 1,50\text{ mm}$). Si se sigue acercando la superficie al módulo el pulso que había desaparecido vuelve a aparecer (transición de las figuras Distancia $100\text{ mm} - 2,5\text{ mm}$ a Distancia $100\text{ mm} - 3,25\text{ mm}$).

Este fenómeno ocurre siempre, con independencia de la distancia a la que se realiza la medición (se ha comprobado en un rango de distancias de 7 cm a 20 cm). Por lo que el primer pulso recibido es irregular, es decir, desaparece o aparece dependiendo de la posición en la que se encuentre.

Por este fenómeno, el error del módulo incrementa y se ve afectado. Este será el problema principal, causante de la reducción de la precisión del módulo HC-SR04.

3.2.3.4. Atenuación del problema con la señal de recepción del modulo.

El módulo HC-SR04 se basa en la transmisión y recepción de una señal de ultrasonidos a 40 KHz . Los transductores emisor y receptor están alineados, además se mueven siempre paralelamente al plano sobre el que se realizan las mediciones, separados a una distancia d el plano. Para intentar atenuar el problema, anteriormente descrito, se usa una pequeña barrera absorbente entre el transductor emisor y el receptor. Su finalidad es reducir el ruido parcial que captura el receptor proveniente del emisor. Este fenómeno ha sido estudiado y evaluado en una publicación con unas características parecidas a este trabajo.[24]

La barrera absorbente está hecha de espuma de poliuretano con un grosor de un centímetro. Se le ha añadido una lámina rígida muy delgada de papel encarado al transductor receptor para ayudar a evitar este fenómeno.

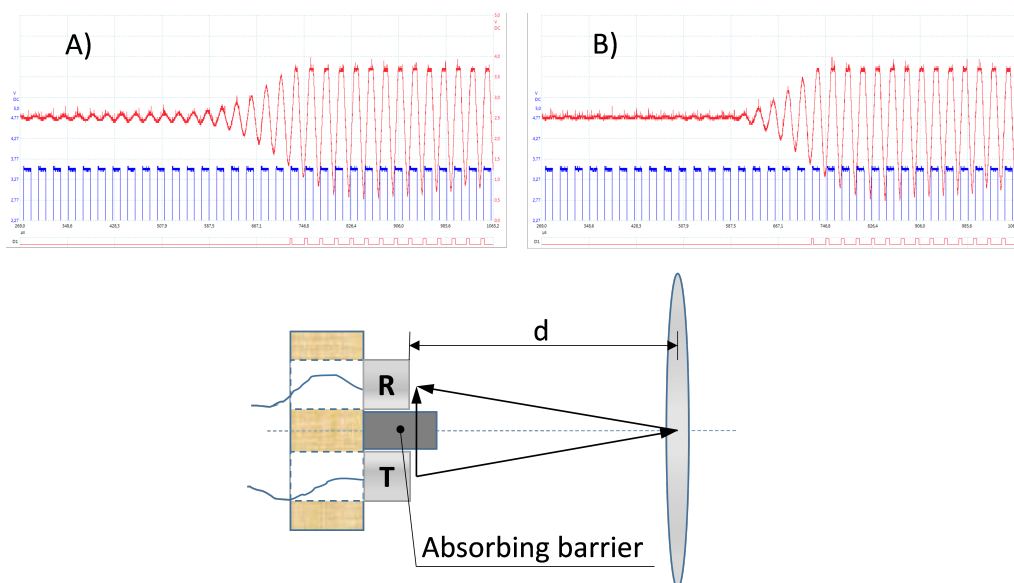


Figura 3.29: Atenuación del problema con la señal signal

En la parte inferior de la figura 3.29, se puede visualizar un *sketch* del sistema de medición y la localización de la barrera absorbente, que se sitúa entre el transductor transmisor (T) y receptor (R). En la parte superior derecha A), se muestra la señal recibida por el receptor cuando la barrera aislante no es usada. Mientras que, en la parte izquierda B), la señal cuando la barrera absorbente es posicionada entre los transductores transmisor y receptor.

Observando A) y B) se puede apreciar un decremento significativo del ruido cuando la

barrera aislante es usada, esto permite al sistema localizar mejor el primer pulso de ultrasonidos.

3.3 Robot cartesiano XY

Se utiliza un robot cartesiano XY con una zona de trabajo de $17\text{ cm} \times 20\text{ cm}$, que esta dotado de movimiento en el plano horizontal gracias a dos motores paso a paso² que se tiene en cada dirección X e Y.

3.3.1. Motores paso a paso.

El motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares. Esto permite controlar la posición de su eje, de esta forma excitando las bobinas adecuadas se moverá el eje con libertad.

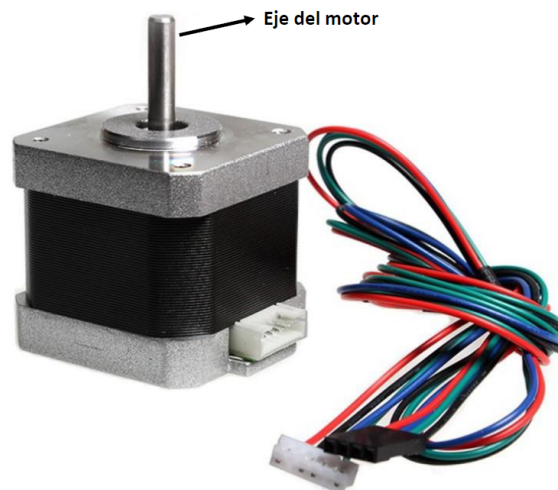


Figura 3.30: Motor paso a paso

En este robot se emplean dos motores paso a paso NEMA-17-17HS8401 [25] bipolares. Entre sus características se destaca que tienen un ángulo de paso $1,8^\circ$, es decir, se necesita dar 200 pasos para que el eje realice una vuelta completa. Tiene una tensión de alimentación de $12V$ y un torque de $4,8\text{Kg/cm}$.

3.3.2. Controladores de los motores paso a paso.

Para poder utilizar el motor paso a paso se necesita los controladores A4988. Su función es la de simplificar el manejo de motores paso a paso desde un procesador como el ATmega328 (Arduino Nano). Gracias al A4988, para controlar el motor, solamente son necesarias tres salidas digitales, una para indicar el sentido de giro (*DIR*), otra para comunicarle al motor que se desea avanzar un paso (*STEP*) y otra para activar o desactivar el controlador A4988 (*EN*).

²<https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/>

Entre las características del controlador A4988 se puede destacar lo siguiente:

Intensidad máxima	2 A
Tensión máxima	35 V
Microsteps	16

Tabla 3.6: Características del controlador A4988.

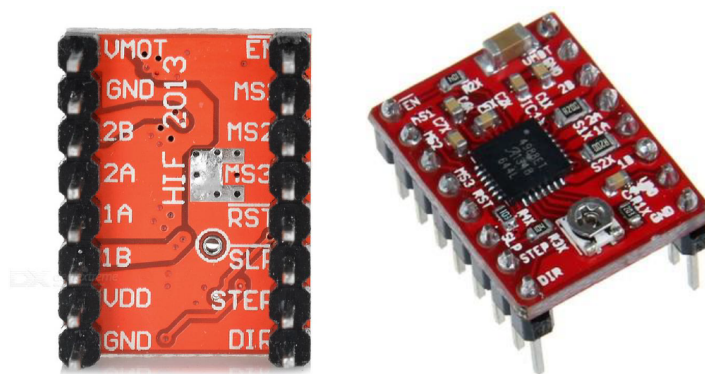


Figura 3.31: Controlador A4988

“Estos controladores pueden alcanzar altas temperaturas durante su funcionamiento y es necesario disipar el calor para que el dispositivo no se dañe. Para intensidades superiores 1A en el A4988 ... es necesario añadir un sistema de disipación de calor... Disponen de protecciones contra sobreintensidad, cortocircuito, sobretensión y sobrettemperatura. En general, son dispositivos robustos y fiables” [25]

Para conseguir una precisión superior al paso nominal del motor se aplica una técnica llamada *microstepping* [26].

Los motores paso a paso 17HS8401 tienen un ángulo de paso de $1,8^\circ$ o 200 pasos por revolución. Mientras que, un controlador de *microstepping* como el A4988 permite resoluciones más altas permitiendo ubicaciones de pasos intermedios, que se consiguen al energizar las bobinas internas del motor con niveles de corriente intermedios.

Por ejemplo, al establecer el motor a una resolución de un cuarto de paso, el motor pasa de 200 pasos por revolución a 800 micro pasos por revolución. Por lo que, ahora se necesitan dar 800 pasos para que el eje del motor realice una vuelta completa.

La configuración del *microstepping* se establece por los pines MS1, MS2 y MS3 a estos pines hay que aplicarles tensión 5V (*HIGH*), o no 0V (*LOW*) dependiendo de la configuración que se desee. Estos pines, a su vez, están conectados a tierra con resistencias de *pull-up* por lo que si no se les aplica tensión o no se conectan a nada están a 0V, es decir, *LOW*, tabla 3.7.

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
Hight	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Tabla 3.7: Configuraciones del controlador A4988.

Cada pulso transmitido al pin de entrada *STEP* corresponde a un paso o micropaso (depende de la configuración) del motor en la dirección (horaria o antihoraria) seleccionada por el pin *DIR*. Hay que tener en cuenta, que los pines *STEP* y *DIR* no están conectados internamente a ninguna resistencia de *pull-up* o a algún voltaje, por lo que estos pines deben estar siempre conectados en la aplicación. Si se quiere mantener la misma dirección todo el tiempo, el pin *DIR* se debe conectar a *GND* o *Vcc* dependiendo de lo que se desee.

El chip tiene tres entradas diferentes para controlar su estado: *RST*, *SLP* y *EN*. Para información más detallada de estos pines se ha de consultar su documentación.

Para el montaje del proyecto se deben tener algunas consideraciones en cuenta. Al no se emplearse los pines *RST* y *SLP* (*resetysleep*) se conectan entre ellos para que queden inutilizados. El pin *EN* sirve para la activación (*enable*) del chip, al ser una entrada negada e internamente conectada a una resistencia de *pull-up* podemos dejar este pin desconectado (*chipsiempreactivo*) o conectarlo a un pin del microcontrolador (ATmega328) para controlar cuando se quiere el chip activo.

3.3.3. Implementación final

En la figura 3.32, se muestra el esquema para dotar de movimiento al robot cartesiano XY.

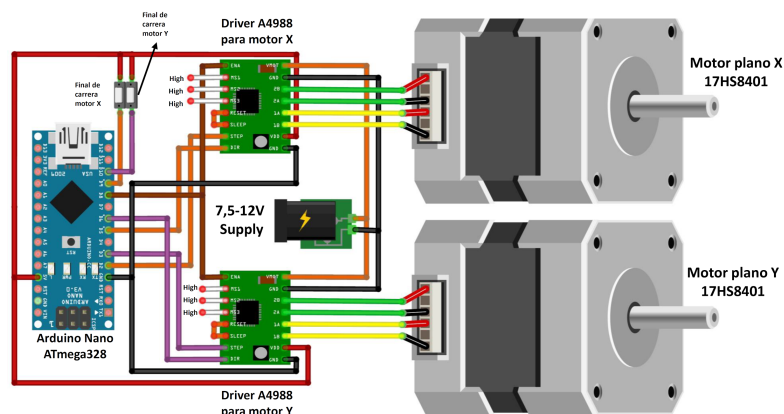


Figura 3.32: Esquema del robot cartesiano XY.

Con esta implementación los motores X e Y están listos para usarse.

En resumen, el motor del eje X es controlado por el Arduino Nano mediante los pines 2,5 y 8, conectados a *DIR*, *STEP* y *EN*, respectivamente, del controlador A4988 (driver)

del motor X. Mientras que el motor eje Y, es también controlado por el Arduino Nano mediante los pines 3,6 y 8, conectados a *DIR*, *STEP* y *EN*, respectivamente, del controlador A4988 (driver) del motor Y.

Los chip's A4988 son configurados para funcionar con *microstepping*, estableciendo una resolución de un dieciseisavo, esto quiere decir, que para que el eje del motor de una vuelta completa necesita dar 3200 micro pasos. Para conseguir esto se han puesto los pines *MS1*, *MS2* y *MS3* de ambos chips a un nivel alto (*5VHigh*).

Se puede observar que el pin 8 del Arduino Nano controla el pin *EN* (*enable*) de los controladores de los motores X e Y.

También, se conectan dos *switches* finales de carrera en los pines 9 y 10, que sirven para detectar el punto (0,0) del robot. Si el robot en pleno funcionamiento se apaga por error, al volverse a encender no sabe en qué posición del plano se encuentra. Por lo que, lo primero que hace es intentar regresar al punto ($x = 0, y = 0$) que es el origen. Cuando el robot alcanza el punto ($x = 0, ?$) el final de carrera de motor X se activa indicando que se ha llegado al origen en el eje X. De la misma forma, cuando el robot alcance el punto ($?, y = 0$) el final de carrera del motor Y se activa indicando que se ha llegado al origen en el eje Y.

Con la estructura del robot finalizada (los perfiles, las correas dentadas, los cables y las ruedas para el desplazamiento) y esquema anteriormente descrito se tiene un robot cartesiano capaz de desplazar cualquier herramienta que se le acople en su cabezal por el plano XY, con rango de trabajo efectivo de $17 \times 20 \text{ cm}$.

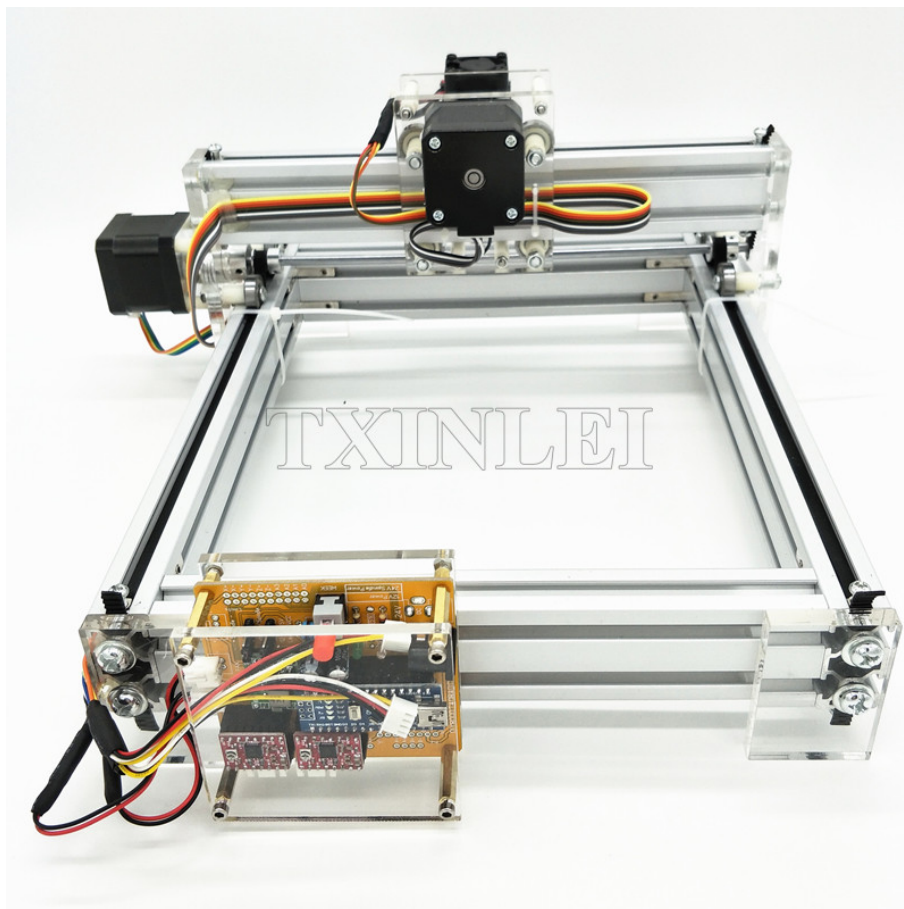


Figura 3.33: Robot cartesiano XY

CAPÍTULO 4

Software y adaptaciones

El proyecto consta de varias partes, las cuales, necesitan un código fuente para funcionar y comunicar correctamente las distintas partes que lo componen. En este apartado, se describen los programas generados. Se ha utilizado el IDE de Arduino para la programación de los microcontroladores, tanto del microcontrolador que controla el robot cartesiano XY, como del sensor de ultrasonidos.

La comunicación entre ambas partes del proyecto se realiza mediante el uso de un ordenador con un programa escrito en *PowerShell* de Windows 10. En un principio, se deseaba conectar ambos dispositivos directamente, para que pudiesen comunicarse mediante una conexión I^2C ¹. Existen librerías en Arduino para la implementación de este protocolo, pero hacen uso de interrupciones y otras series de características, como se van a modificar valores internos del microcontrolador (descritos en el anexo I) la conexión que se quiere realizar falla y se generan valores erróneos o ni siquiera funciona el sistema como se espera. Ésto se debe a que hay incompatibilidades con una conexión I^2C y las modificaciones que se realizan en el microcontrolador.

¹"I2CLicensingInformation" (PDF) .nxp.com. Archived(PDF)fromtheoriginalon2017-01-10. Retrieved2018-04-29.

El diagrama de flujo de la comunicación de los programas desarrollados para obtener la orografía se muestra en la figura 4.1.

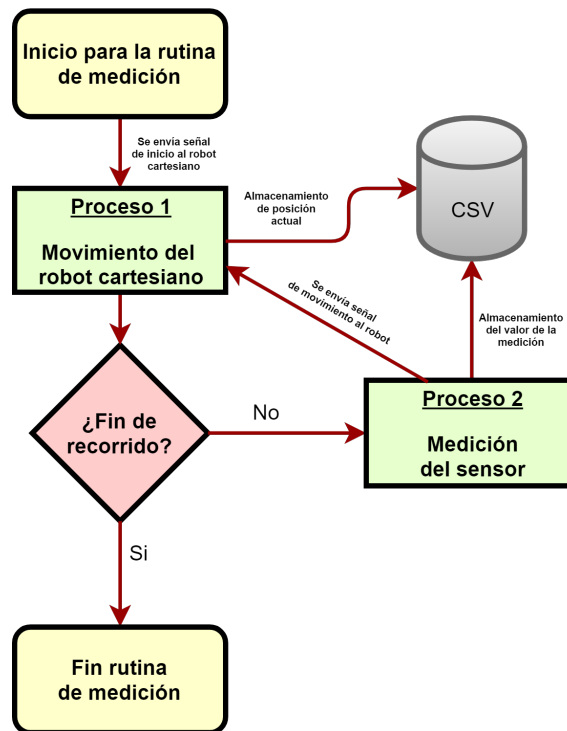


Figura 4.1: El diagrama de flujo de la comunicación de los programas desarrollados para obtener la orografía

Al iniciar el programa se envía una señal específica al microcontrolador del robot cartesiano, para que se mueva a la posición deseada. Al tratarse del inicio del programa, éste busca la posición de inicio, es decir, el origen de coordenadas ($x = 0, y = 0$).

Al llegar al destino se comprueba que el punto en el que se encuentra no sea el final del recorrido. Si no lo es, primero se guarda la posición actual del robot, después emite la señal de activación del sensor de ultrasonidos, que devuelve varias mediciones realizadas. Éstas se concatenan con la posición actual y se almacenan. Al no tratarse del final del recorrido se vuelve a enviar otra señal de movimiento al robot. Este proceso se repite hasta que se llegue a la posición final en la que no se realiza ninguna acción, solo se finaliza el programa en ejecución y se guarda en un documento CSV de los datos recogidos.

4.1 Implementación software del robot cartesiano

El programa para el funcionamiento del robot cartesiano se ha implementado con el IDE de Arduino. Los programas desarrollados para Arduino están, en general, estructurados de una forma determinada. El programa cuenta con tres secciones de código diferenciadas.

El *setup()*, esta sección del código se ejecuta una sola vez al iniciar el programa de Arduino. Cada vez que se reinicia el dispositivo esta sección de código se vuelve a cargar para que todo vuelva a inicializarse, ya que el modelo del Arduino escogido en este proyecto no tiene memoria persistente. Esta sección de código se utiliza para la configuración e inicialización de los pines y de las variables necesarias para el correcto funcionamiento del programa.

El *loop()*, es la sección del programa que sirve para establecer el flujo y su comportamiento. Su ejecución será secuencial y cíclica. Como el nombre de la sección indica, es decir, la ejecución de este código nunca finaliza, una vez se ejecute la última instrucción vuelve a empezar desde el inicio sin ejecutar el *setup()* de nuevo.

Podemos declarar fuera de estas dos secciones cualquier función y variable global necesaria, esta sección libre también sirve para cargar librerías adicionales que se puedan ser de utilidad. Cabe recordar, que el lenguaje de programación de Arduino es una adaptación de C++, es posible incluso utilizar comandos estándar del lenguaje.

En la figura 4.2. Se muestra el comportamiento del programa generado para el robot cartesiano.

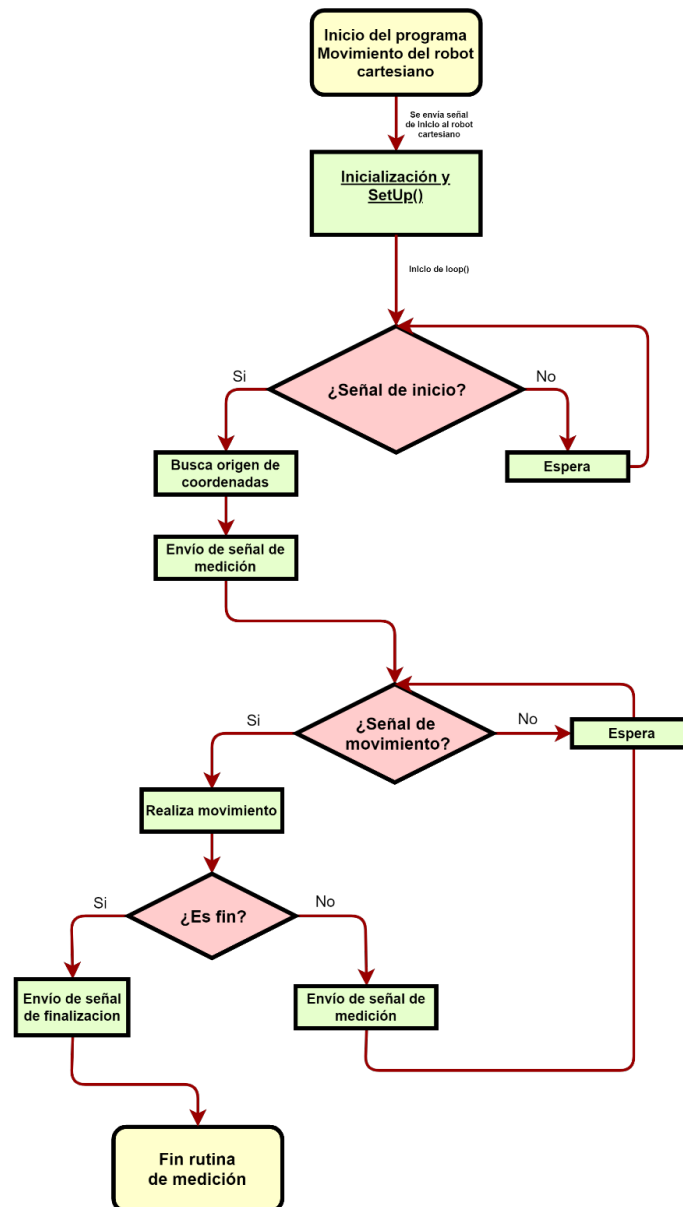


Figura 4.2: Diagrama de flujo de ejecución del robot cartesiano

Al conectar el robot a una fuente de alimentación se inicia el programa instalado en él. Inicializándose así, todas las variables globales, se cargan todas las librerías externas y se inicializan las funciones que se hayan definido en la sección libre de nuestro programa. Luego se ejecuta el *setup()*, descrito anteriormente, en este apartado se establecen características básicas de la configuración del robot tales como:

- Los pines que se usan.
- La configuración de estos pines para un funcionamiento analógico o digital.
- La configuración de los distintos pines de entrada (*inputs*) o pines de salida (*outputs*).
- Configuración de resistencias de *pull-up*.
- Configuración del canal de comunicación.
- Declaración de variables, como las dimensiones de la zona de trabajo útil sobre las que se quieren realizar las mediciones.

Después de que se termine de ejecutar el *setup()* del programa se inicia la sección *loop()*, descrita anteriormente. En este apartado se entra a un bucle infinito, que está constantemente a la espera de la señal de inicio para el comienzo del proceso de movimiento del robot.

Cuando se detecta la señal de inicio lo primero que se hace es ejecutar una función llamada *buscarCero()* que busca el origen de coordenadas haciendo uso los finales de carrera debidamente situados en el robot. Una vez se encuentre en el origen de coordenadas, el robot emite una señal de medición que indica que se encuentra en la posición adecuada para realizar las mediciones. Además, se emite la posición en el plano actual, que en este caso es ($x = 0, y = 0$).

En este punto, se vuelve a entrar en un bucle infinito de espera hasta que una nueva señal de movimiento llegue al robot cartesiano. Ésta se genera en el programa de comunicación entre el robot cartesiano y el sensor de ultrasonidos. Sirve para informar de que las mediciones se han realizado correctamente y se puede mover el cabezal del robot cartesiano a una nueva posición, para seguir realizando mediciones en otro punto del plano. Cuando el robot recibe la señal de movimiento sale de su espera infinita y procede a situarse en la siguiente posición deseada. Una vez allí, comprueba si se encuentra en la posición final, ($x = fin, y = fin$), si esto es así se envía una señal que indica que se encuentra en esta posición y sus coordenadas. Dando así lugar al final de la rutina.

Si no se encuentra en la posición final, envía una señal de medición y las coordenadas de la posición actual del robot. A continuación, se genera una pequeña espera estática y vuelve a entrar en un bucle infinito de espera, hasta que una nueva señal de movimiento llegue al robot cartesiano, volviendo así al estado anterior.

Con este proceso se genera una malla de puntos por los que el robot se mueve, como se puede ver en la figura 4.3.

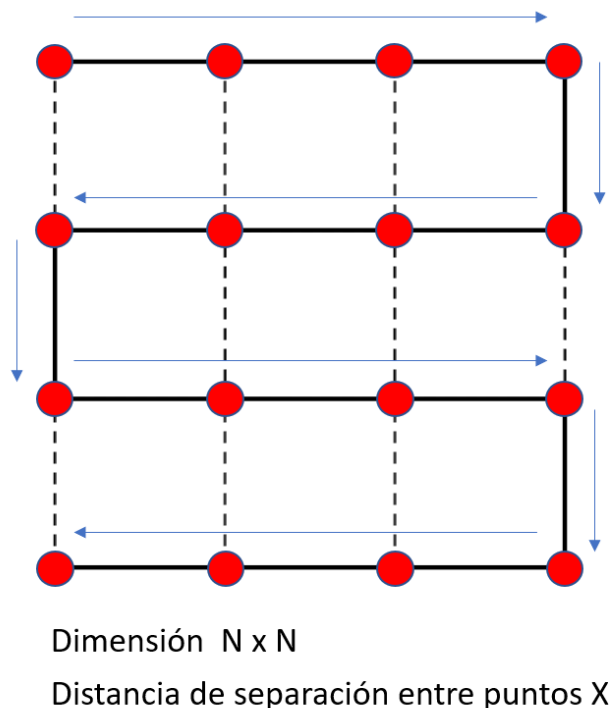


Figura 4.3: Malla de puntos

Los distintos movimientos que realiza el robot se producen debido a las variables globales que se definen en el *setup()* del programa. En el *setup()* se define inicialmente la posición del robot que por defecto es $(x = 0, y = 0)$, las dimensiones que tiene la zona de trabajo útil del robot y dos variables, N y X , que representan la dimensión de la malla de puntos generada por el movimiento del robot y la distancia de separación entre puntos.

El robot solo almacena la posición actual en la que se encuentra cada vez que avanza sobre el plano, con el conocimiento de la posición actual, el de las dimensiones de la malla que debe recorrer, y el de la distancia entre puntos de la malla. El robot decide su próximo movimiento cada vez que recibe una señal de movimiento.

4.2 Adaptación hardware del sensor de ultrasonidos e implementación software.

4.2.1. Adaptación hardware para la emisión del sensor de ultrasonidos.

Para aumentar la complejidad del proyecto se ha intentado simular y mejorar el comportamiento del controlador del sensor de ultrasonidos, haciendo uso del microcontrolador M1. La finalidad de este proceso es el poder prescindir del máximo número de componentes del módulo HC-SR04. Al tener ya un microcontrolador (M1), se busca poder asignarle todos los procesos necesarios para realizar una medición, pudiendo así prescindir del microcontrolador EM78P153S.

Como primer paso para prescindir del microcontrolador EM78P153S del módulo de ultrasonidos, se debe generar un proceso que sea capaz de activar el piezoeléctrico del

sensor de ultrasonidos, al voltaje necesario. De esta forma, se generan los pulsos de ultrasonidos deseados, es decir, se debe generar un proceso para la emisión del sensor de ultrasonidos sin hacer uso del microcontrolador de este módulo. En la figura 4.4, se ob-

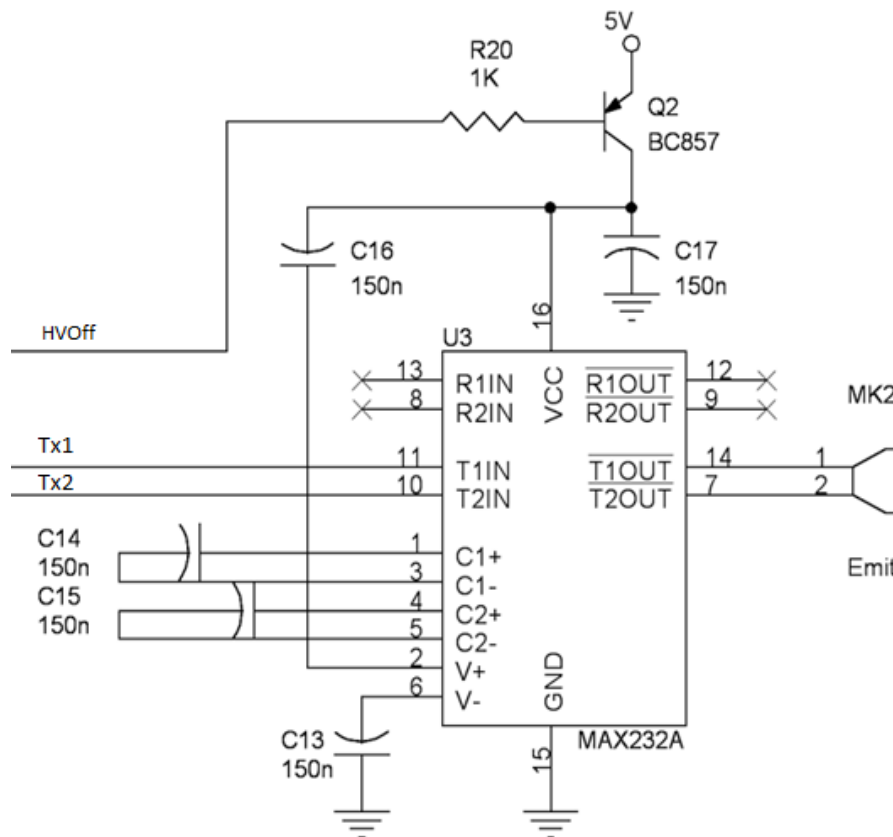


Figura 4.4: Esquema del circuito del módulo HC-SR04

serva una parte del esquema del circuito del módulo HC-SR04, que sirve específicamente para excitar el piezoeléctrico MK2, al ser excitado al voltaje necesario y como indica el fabricante, se genera una serie de ultrasonidos a una frecuencia determinada.

Haciendo uso de ingeniería inversa [27] se estudia el comportamiento de esta parte del dispositivo, con la finalidad de reproducir el comportamiento del microcontrolador del módulo de ultrasonidos.

En la figura 4.5 se pueden observar las señales de entrada necesarias para activar este proceso.

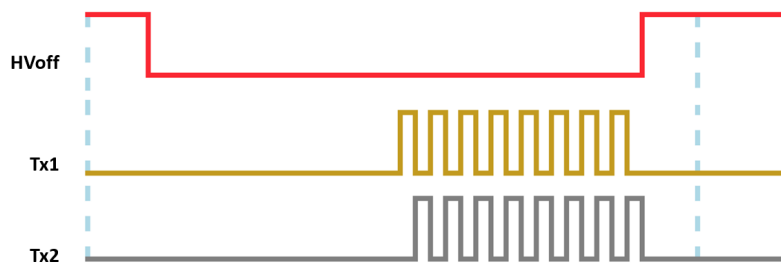


Figura 4.5: Señales para la inicialización del proceso de generación de ultrasonidos.

En este trabajo, no se describe el proceso que se ha seguido para conseguir todos los resultados finales del análisis mediante ingeniería inversa del dispositivo, ya que es un proceso largo y no es de relevancia en la explicación de este proyecto.

Los resultados obtenidos indican el tipo de señales que se deben generar para la activación del integrado MAX232A. Este componente conseguirá generar unos potenciales en dos de sus salidas de $+ / - 10 V$, que conectados de la forma correcta podrán alimentar al piezoeléctrico MK2 20 V, que es suficiente para generar los ultrasonidos.

Por lo tanto, mediante el IDE de Arduino y hardware específico se genera una función que al ser llamada, reproduce a las señales descritas en la imagen anterior. De esta forma, se simula el comportamiento exacto del microcontrolador del módulo HC-SR04.

A modo resumen y explicativo, cuando la señal **HVoff** tiene un potencial de 0 V activa el transistor $Q2$, que permite el paso de corriente eléctrica, desde el colector al emisor. De esta forma, se activa la fuente de alimentación principal del integrado MAX232A, para que pueda funcionar correctamente.

$T x 1$ debe ser una señal de 40 KHz con un *duty cycle* del 50 % y con una amplitud de 5 V. Esta señal es una de las entradas que necesita el integrado MAX232A para producir la salida deseada. $T x 2$ es una señal idéntica a $T x 1$, pero invertida. Para facilitar el trabajo al microcontrolador estas señales se generan, a partir de una señal principal, haciendo uso de puertas lógicas. A la señal fuente se la denomina REMIT.

En la figura 4.6 se muestra un esquema de las señales que participan en la generación de $T x 1$ y $T x 2$.

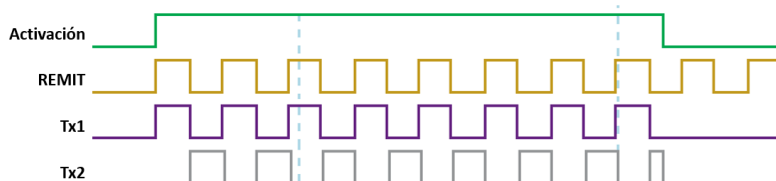


Figura 4.6: Esquema de señales generadas, activación y REMIT generadas por software $T x 1$ y $T x 2$ por hardware

$T x 1$ se genera, a partir, de las señales de REMIT y activación, ambas señales se combinan haciendo uso de puertas lógicas NAND. $T x 1$ es el resultado de la operación lógica:

$$T x 1 = Activacion \times REMIT$$

$T x 2$ se genera a partir de la señal $T x 1$ y Activación, ambas señales se combinan haciendo uso de puertas lógicas NAND. $T x 2$ es el resultado de la operación lógica:

$$T x 2 = T x 1 \times Activacin$$

En este apartado, hay que destacar la dificultad de conseguir generar señales con las características de REMIT. Debido a que son señales con frecuencias muy determinadas y deben describir unos *duty cycles* exactos. Generar este tipo de señales haciendo uso del microprocesador AT-Mega32U8, no es trivial, ya que su microcontrolador es mono hilo, solo puede ejecutar una tarea al mismo tiempo. Por lo que la generación de esta señal simultánea a la ejecución de otro código es imposible. El proceso para la generación de este tipo de señales con el microcontrolador de AT-Mega32U8 está descrito en el Anexo I.

4.2.2. Adaptación hardware para la recepción del sensor de ultrasonidos.

En el apartado anterior se describe como deber ser la adaptación de un proceso para la emisión del sensor de ultrasonidos sin hacer uso del microcontrolador de este. En este apartado se describe lo contrario, como hacer la adaptación de un proceso para la recepción de la señal de ultrasonidos sin hacer uso del microcontrolador del módulo de ultrasonidos.

En la siguiente figura 4.7 se observan los componentes hardware encargados de la recepción de la señal de ultrasonidos. Estos componentes y funcionalidades ya fueron descritos en la sección 3.2.2

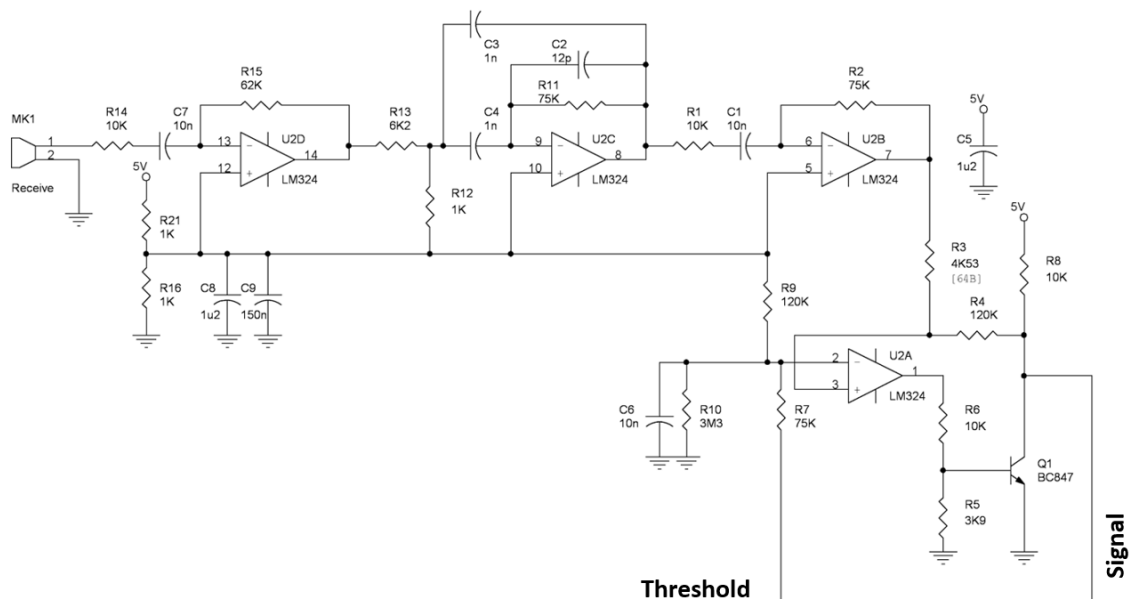


Figura 4.7: Componentes hardware

Una vez más se estudia el comportamiento de esta parte del dispositivo con la finalidad de reproducir el comportamiento del microcontrolador del módulo de ultrasonidos. Podemos observar que con el microcontrolador únicamente interactúan dos señales Threshold y Signal, y estas señales son una de salida y otra de entrada del microcontrolador, respectivamente.

Este proceso tiene únicamente una salida del microcontrolador que es la señal Threshold, esto quiere decir que solo se necesita producir una señal desde M1. Threshold, marca el umbral de conmutación para el comparador con histéresis U2A. Esta señal se

activa poco después de excitar el piezoeléctrico transmisor y se establece a un potencial de 1,7 V. Como Arduino no puede generar un potencial en específico, se consigue ese valor con un divisor resistivo.

En este proceso se tiene una única entrada al microcontrolador que es la señal Signal. Ésta se genera por los ultrasonidos recibidos por el piezoeléctrico MK1, que son interpretados como una señal digital y luego transformados por los amplificadores operacionales. En la figura 4.8 se observa como es signal.

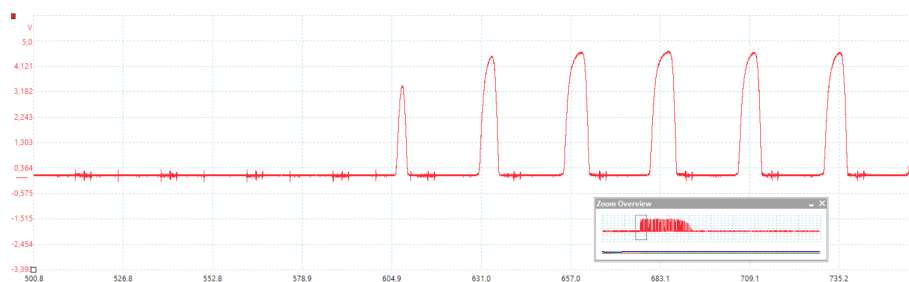


Figura 4.8: Proceso de captación de la señal de ultrasonidos

El microcontrolador del módulo devuelve el tiempo transcurrido desde que se han generado los ultrasonidos hasta que se detecta el primer flanco de subida en la señal Signal.

En este proyecto, se usa la señal Signal, pero con un enfoque distinto, se debe imitar el comportamiento del microprocesador para que el sistema de la recepción de ultra sonidos funcione y genere correctamente Signal. Por lo que solo es necesario generar con el micro controlador M1 la señal trheshold.

4.2.3. Adaptación hardware para la detección del phase shift.

En este proyecto la variación de fase (PSA) se determina vía hardware, el planteamiento es muy sencillo.

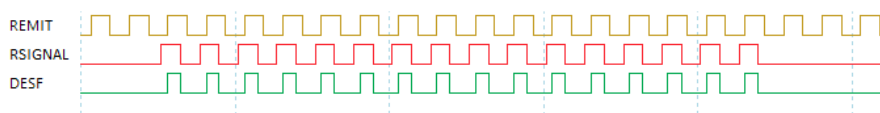


Figura 4.9: Señales usadas para el cálculo de PSA

Se desea averiguar el desfase de una onda con respecto a otra. En la figura 4.9 la onda de referencia es REMIT y la que tiene un desfase es R SIGNAL. Para calcular el desfase, las señales REMIT y R SIGNAL se combinan mediante puertas lógicas NAND, y su comportamiento sigue la siguiente fórmula booleana:

$$DESF = REMIT \times R SIGNAL$$

La duración de la señal DESF es de 400 ms con una frecuencia de 40 KHz, 8 voltios de amplitud y un *duty cycle* variable. Esta señal produce 16 pulsos cuya duración de pulso alto varía de 0 a 12,5 ms. Ésta, carga un condensador C de la figura 3.1, que está diseñado en combinación con R1 (resistencia de carga) y R2 (resistencia de descarga) de manera que:

Si el pulso alto de la señal DESF dura 12,5 ms, el condensador se carga a 5 V. De esta forma, el microcontrolador leyendo el voltaje del condensador, después de que este sea cargado por DESF, puede inferir el desfase de las ondas.

4.2.4. Consideraciones para la implementación software del sensor de ultrasonidos.

Antes de comenzar con la explicación de la implementación del programa para el sensor de ultrasonidos, se justifica el uso una librería externa no oficial de Arduino. Para cambiar el valor de salida de una señal digital de Arduino, la función a usar y la más común, es *digitalWrite* (*pin*, *value*). Ésta función es muy lenta, puesto que, cambiar el valor lógico de una señal digital, tarda con esta función 6280 *ns*. Para este proyecto se necesitan velocidades de actuación mucho mas rápidas, se opta por utilizar una librería externa llamada *digitalWriteFast*. Con ésta se pueden usar funciones como *digitalWriteFast* (*pin*, *value*) que puede aumentar la velocidad del cambio de valor de una señal digital a 125 *ns*, es decir, 50 veces más rápido que la función tradicional.

En este apartado se hace uso de interrupciones, básicamente, una interrupción es una señal que interrumpe la actividad normal del microcontrolador para ejecutar una tarea prioritaria, la cual en el momento de activación que es atendida y ejecutada. Cuando una señal activa una interrupción, la ejecución normal del microcontrolador se suspende, todos los registros en memoria y el estado actual del microcontrolador se almacenan para que se ejecuten unas funciones especiales programadas para estos casos.

Cuando la ejecución de esta función especial finaliza, se restablece el estado del microcontrolador en el que se activó la interrupción y se continua en el mismo punto de ejecución en el que se encontraba antes de que la señal activase la interrupción.

“El concepto de interrupción nace de la necesidad imperiosa de reaccionar de forma inmediata en respuesta a un acontecimiento electrónico fulgurante, que no admite demora. Bien sea por la urgencia del suceso o porque algo se podría perder de forma irre recuperable sino reaccionamos con suficiente presteza”.[28]

El activador de las interrupciones en este proyecto son solo de tipo hardware, es decir, los cambios de voltaje en alguno de los pines especificados pueden generar una interrupción en el microcontrolador.

En la siguiente tabla se muestran los distintos tipos de activadores para una interrupción hardware.

Tipo de interrupción	Condición de salto
LOW	La interrupción se activa cuando el pin es LOW.
CHANGE	La interrupción se activa cuando el pin cambia de valor de LOW a HIGH o al contrario
RISING	La interrupción se activa cuando el pin cambia de valor de LOW a HIGH
FALLING	La interrupción se activa cuando el pin cambia de valor de HIGH a LOW

Tabla 4.1: Tipos de activadores para una interrupción.

4.2.5. Implementación software del sensor de ultrasonidos.

La implementación software para realizar las mediciones con el sensor de ultrasonidos modificado se ha realizado con el IDE de Arduino. Las modificaciones son las que se han comentado en los apartados 4.2.1 y 4.2.2 . La especificación de las conexiones hechas en el proyecto se detalló en la figura 3.1.

El diagrama de flujo de ejecución para realizar la medición con el sensor de ultrasonidos se muestra en la figura 4.10.

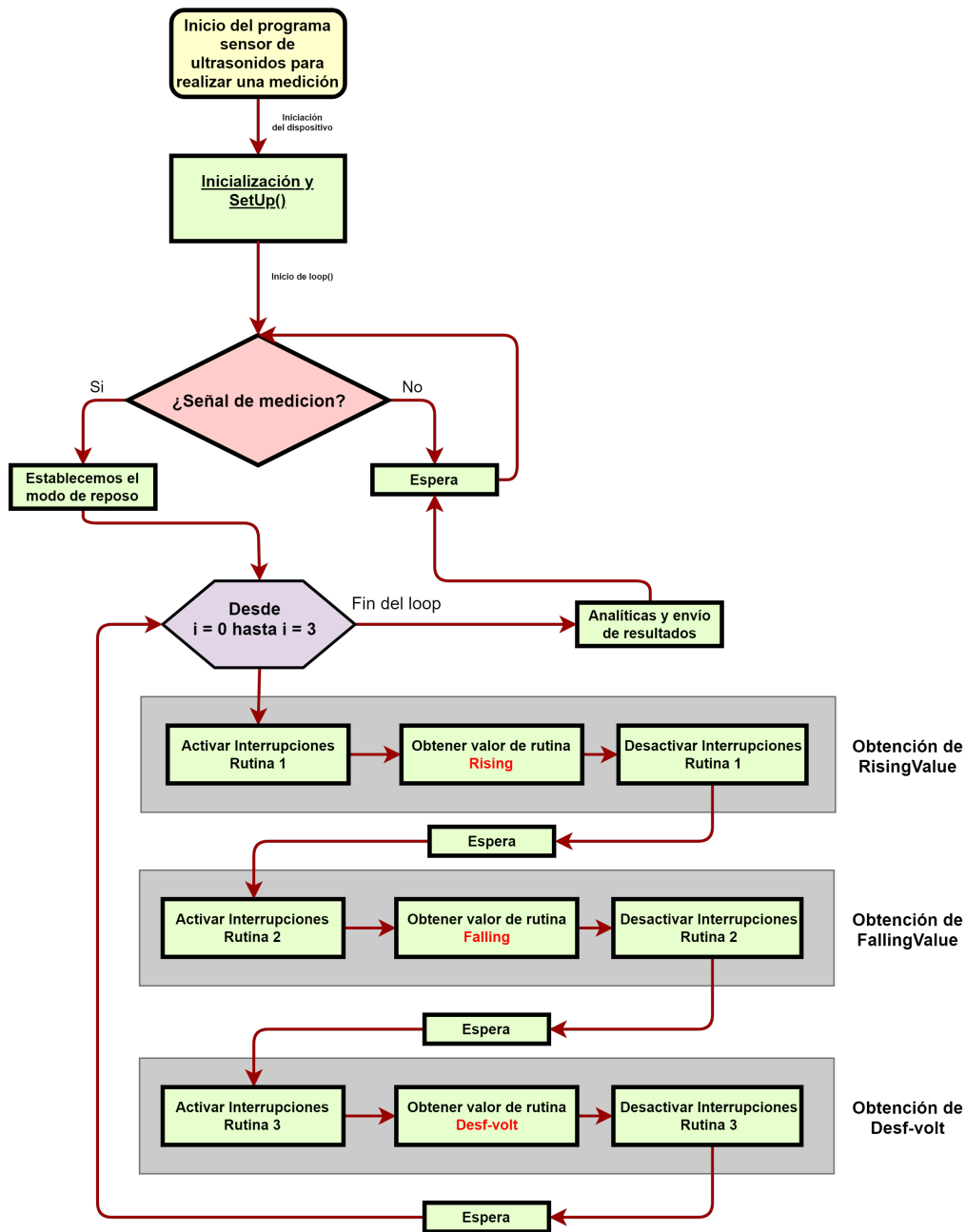


Figura 4.10: Diagrama de flujo de ejecución para realizar la medición con el sensor de ultrasonidos

Con el fin de poder realizar el proceso completo de emisión y recepción de ultrasonidos, se han desarrollado funciones específicas para cada proceso descrito en el diagrama de flujo anterior.

A continuación se explican, en orden los distintos procesos.

En la inicialización y *setup()* se definen variables globales necesarias para el proceso completo, se cargan las librerías externas necesarias, como *digitalWriteFast*, se establecen los pines que se van a usar y las funciones que tienen cada uno de ellos (pines de entrada, salida, interrupciones).

Después de la ejecución del *setup()* del programa se inicia la sección *loop()*. En este apartado se entra a un bucle infinito, el cual esta constantemente a la espera de la señal de inicio para el comienzo del proceso de medición del sensor de ultrasonidos.

Cuando se detecta la señal de inicio, lo primero que se hace es ejecutar una función para que el sistema entre en un estado de reposo. Así, los valores de las distintas señales que se generan por el microcontrolador e intervienen en el proceso de medición del sensor de ultrasonidos, deben ser los que se marcan en la siguiente tabla:

PIN/Señal	Tipo	Valor
HvOff	OUTPUT	HIGH
Threshold	OUTPUT	HIGH
Activación	OUTPUT	LOW
Remit	OUTPUT	LOW
RSignal	OUTPUT	LOW
Pin 0	Interrupción	DISABLE
Pin 2	Interrupción	DISABLE

Tabla 4.2: Valores de las señales para el estado de reposo.

Una vez que el sistema se encuentra en reposo y tras una pequeña espera, se entra a un bucle donde se obtienen distintos parámetros implicados en la medición de la distancia. Este bucle se repite cuatro veces. Los parámetros que se obtienen en una iteración son *RisingValue*, *FallingValue*, *DesfValue* y *VoltValue*.

Para obtener el valor de *RisingValue* primeramente se activan las interrupciones correspondientes llamando a la función *ActivarInterrupciones1()*. Con esta función se asocian a los pines 0 y 2 del microcontrolador M1 dos interrupciones distintas. Ambas se inician con un activador en la señal del tipo *RISING* (tabla 4.2), es decir, cuando en este pin detecte un flanco de subida en la señal que está controlando.

La función asociada a la interrupción del pin número 0 es sencilla. Como se muestra en la figura 3.1, a este pin está conectado la señal *SIGNAL* del módulo de ultrasonidos. Cuando la primera onda de la recepción de ultrasonidos se detecta se activa esta interrupción, que lo único que hace es detener la generación de la señal *REMIT* y marcar una variable global booleana como *FALSE*, indicando que se ha ejecutado esta interrupción. La función asociada a la interrupción del pin número 2, es únicamente un contador. Como se muestra en la figura 3.1, a este pin está conectado la señal *REMIT* que genera el microcontrolador M1. Esta interrupción se activa desde la generación de *REMIT*, se cuentan las veces que la señal *REMIT* cambia de un valor *LOW* a *HIGH*, siempre y cuando, no se haya ejecutado aún la interrupción asociada al pin número 0. En el programa de ejecución de esta interrupción existe una sección de código que detiene la activación del piezoeléctrico emisor de ultrasonidos cuando el contador es igual a 8.

Después de la activación de las interrupciones, se comienza un proceso para obtener el *RisingValue* de la medición.

Se llama a la función *activacionDelSensor()*, esta función tendrá como finalidad generar el pulso de ultrasonidos necesario para la medición.

Se muestra el código simplificado usado para esta función:

```
//RUTINA DE ACTIVACION PARA GENERACIÓN DE ULTRASONIDOS
void activacionDelSensor(){
  //Establecemos el modo de reposo
  digitalWriteFast(hvoff, HIGH);
  digitalWriteFast(threshold, HIGH);
  digitalWriteFast(activacion, LOW);
  //Reposo para REMIT
  TCCR3A = 0;
  TCCR3B = 0;
  TCNT3 = 0;
  ...
  ...
  //A los 500 microsegundos empezamos la rutina
  delayMicroseconds(500);
  //Empezamos la activación del modulo
  digitalWriteFast(hvoff, LOW);
  digitalWriteFast(threshold, LOW);
  delayMicroseconds(200);
  //A los 200 microsegundos activamos la señal activacion
  digitalWriteFast(activacion, HIGH);
  //En el mismo momento se activa la generación de REMIT (Tx1 , Tx2)
  TCCR3A = _BV (WGM31) | _BV (COM3A1);
  TCCR3B = _BV (WGM32)|_BV (WGM33)| _BV (CS31);
  ICR3 = ((F_CPU / 8) / frequency) - 1;
  OCR3A = ((ICR3 + 1) / 2) - 1;
  //Las señales hvoff, threshold y activacion se detendrán gracias a una interrupción
  asociada al pin 2
}
```

Figura 4.11: Código de la activación de ultrasonidos.

Como se puede observar en el código lo primero que se hace al iniciar esta función, es establecer las señales, threshold, HVoff, activación y REMIT (Tx1, Tx2) en su estado de reposo o estado natural.

Se hace una espera de 500 microsegundos para asegurarnos de que todo el sistema está en reposo, después de los 500 microsegundos empieza la activación del módulo. Las señales HVoff y threshold se establecen a 0V, después de otros 200 microsegundos se establece la señal de activación a 5V. Cuando la instrucción termina de ejecutarse se modifican los registros internos del microcontrolador, al modificarse éstos de forma correcta se consigue REMIT, que es una señal de 40KHz con un duty cycle del 50 % en una salida analógica del Arduino (se explica esta generación en el Anexo I). Gracias a la interrupción asociada al pin número 2 en el momento que se detecten 8 pulsos de la señal REMIT las señales HVoff, Activación y threshold volverán a su estado de reposo.

Una vez ejecutada la función `activacionDelSensor()` se entra a un bucle infinito que no se detiene hasta que no se haya ejecutado la interrupción asociada al pin número 0. Una vez se detenga este bucle infinito el valor de RisingValue es el valor del contador que se ha generado la interrupción asociada al pin número 0, seguidamente se desactivan las interrupciones asociadas a los pines 0 y 2.

En resumen, desde el momento en el que termina de ejecutarse el `setup()`:

1. Se recibe una señal de inicio.
2. Se reinician las variables para que el sistema este en reposo.

3. Se activan las interrupciones. Una interrupción, cuenta las veces que la señal REMIT cambia de un valor LOW a HIGH. Esta señal se genera desde el momento en el que el piezoeléctrico emisor es excitado para la generación de los ultrasonidos. Una interrupción, para detectar la recepción de la señal de ultrasonidos por el piezoeléctrico receptor.
4. Se genera el pulso de ultrasonidos, con el microcontrolador M1.
5. Cuando se detecta que se han recibido el eco de la señal de ultrasonidos, se para el sistema a causa de la segunda interrupción. Obteniéndose el valor del contador que genera la primera interrupción.

El proceso para la obtención del valor de FallingValue es prácticamente igual al proceso para la obtención del RisingValue, únicamente cambia el activador de la interrupción asociada al pin número 2. Se activan las interrupciones correspondientes llamando a la función ActivarInterrupciones2(), con esta función se asocian a los pines 0 y 2 del microcontrolador M1 dos interrupciones distintas. La interrupción asociada al pin número 2 se inicia con un activador en la señal del tipo FALLIING (tabla 4.1), es decir, se activa la interrupción cuando en este pin detecte un flanco de bajada en la señal que está controlando. La interrupción asociada al pin número 0 no es igual que en el proceso para la obtención del valor del RisingValue.

El comportamiento de las funciones asociadas a cada interrupción no varía nada más que en los cambios descritos anteriormente. El resto del código es igual que en el apartado anterior.

Por lo que al final el valor FallingValue es el número de veces que la señal REMIT cambia de un valor HIGH a LOW, desde que comienza el proceso hasta que se recibe la primera señal de ultrasonidos en la recepción.

El proceso para la obtención del valor de DesfValue y VoltValue es parecido a los anteriores. Primero, se activan las interrupciones correspondientes llamando a la función ActivarInterrupciones3(). Con ésta, se asocian a los pines 0 y 2 del microcontrolador M1 dos interrupciones distintas. Ambas se inician con un activador en la señal del tipo RISING. La función asociada a la interrupción del pin número 0 es distinta a las anteriores, cuando la primera onda de la recepción de ultrasonidos se detecta se activa esta interrupción, en este momento se guarda el valor del registro TC3R1B del timer 1 en una variable global. De esta forma, se puede saber el valor del timer asociado a la señal REMIT, cuando se detecta el flanco de subida de la señal SIGNAL (Anexo I). Instantáneamente se genera la señal R SIGNAL, que es una réplica de la señal SIGNAL, por el microcontrolador M1. Se trata de una señal cuadrada de 40KHz con un 50 % de duty cycle, la cual solamente está activa durante 200 microsegundos. Luego R SIGNAL cambia a un valor digital LOW, ejecutándose a continuación una lectura analógica del pin X que está conectado al condensador C, de la figura 3.1, y finalmente se marca una variable booleana como FALSE, que sirve para indicar que se ha ejecutado esta interrupción.

La función asociada a la interrupción del pin número 2 es únicamente un contador. Se activa desde la generación de REMIT, y se cuentan las veces que la señal REMIT cambia de un valor LOW a HIGH hasta que el contador tenga un valor igual a 8, entonces se para la activación del sensor de ultrasonidos.

Después de la activación de las interrupciones, se comienza el proceso para obtener el DesfValue y VoltValue. Se llama a la función activacionDelSensor(), que tiene como finalidad generar el pulso de ultrasonidos necesario para la medición.

Una vez generado el pulso de ultrasonidos se entra en un bucle infinito que no se detiene hasta que no se ejecute la interrupción asociada al pin número 0. Una vez dete-

nido el bucle se obtienen los valores necesarios. VoltValue se obtiene de la interrupción asociada al pin número 0, cuando realiza la lectura analógica al condensador. DesfValue corresponde con el valor del timer, que se guardó en la interrupción asociada al pin número 0. Finalmente se desactivan las interrupciones asociadas a los pines 0 y 2, terminando así el proceso.

El bucle captura cuatro veces estos valores, por lo que calcula la media y varianza de todos y cada uno de los parámetros. Ésto es el resultado que devuelve el proceso de medición, quedando el sistema a la espera, otra vez, de una nueva señal de medición.

4.3 Implementación software del calibrado

Se implementa también, un programa con el IDE de Arduino para el calibrado del sensor de ultrasonidos.

Este programa es sencillo, usando las bases del programa generado en el apartado anterior 4.2, se realizan diez mediciones en un mismo punto. En cada una de las mediciones se obtienen los parámetros RisingValue, FallingValue, DesfValue y VoltValue, éstos están descritos en el apartado 4.2.5.

De estas diez mediciones se calcula la media y la varianza.

El proceso descrito se repite otras diez veces, es decir, calcularemos diez veces la media y la varianza de las mediciones. De estos diez resultados se seleccionan y devuelven, solo con aquellos que tengan una varianza menor que un margen determinado. Con esto se obtienen los parámetros que definen las características reales de un punto en concreto.

Este proceso se repite cada vez que se envía una señal de medición al sensor, y devuelve un resultado final por el puerto serie de comunicación.

El resultado que se genera está estructurado de la siguiente forma:

MEDIA_RisingValue; VARIANZA_RisingValue; MEDIA_FallingValue; ...

CAPÍTULO 5

Calibrado

Para el calibrado se usa el software descrito en el apartado 4.3.

Como se puede observar en la figura 5.1, una superficie circular (A) se coloca perpendicular al módulo de ultrasonidos (B). Esta superficie es móvil gracias a un tornillo micrométrico que controla su posición, figura 5.2. tiene un error de 0.01 mm y tiene un rango de recorrido de 25 mm. tabla

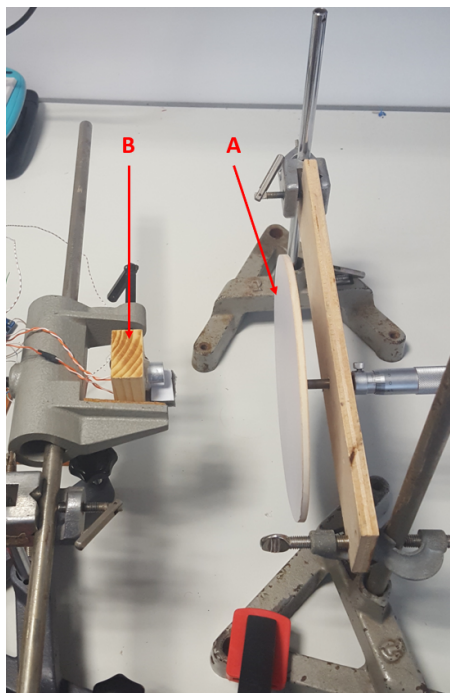


Figura 5.1: Dispositivo de calibración

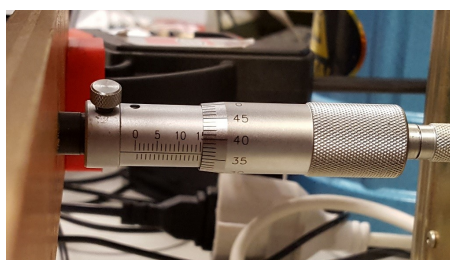


Figura 5.2: Tornillo micrométrico para el control del calibrador.

Para la calibración, los pasos utilizados son 0.10 ± 0.01 mm. Se recolectan 151 muestras, representando cada muestra el recorrido de distancias de 0 a 15.1 milímetros. Como la longitud de onda de la señal de ultrasonidos es de 4.8 mm, se han analizado más de tres longitudes de onda completas.

Se establece el rango exacto en el que se quiere que funcione el dispositivo, es decir establecer una distancia mínima y una distancia máxima. Como ya se indicó anteriormente el interés de este proyecto se centra en distancias cortas, pero buscando una precisión elevada.

Se establece la superficie de calibrado a 100 milímetros del sensor de ultrasonidos. Por lo que se hace un recorrido desde la posición 100 hasta la posición 115.1 milímetros.

Cuando se hacen las mediciones se guarda, la distancia real a la que se encuentra la superficie, la media de RisingValue, FallingValue, DesfValue y VoltValue. Estos datos se guardan en una matriz en una misma fila. Una vez guardados estos datos, la superficie se desplaza 0.01 milímetros y se vuelve a repetir el proceso. Al final se obtiene una matriz de valores de este estilo (se han recortado filas para que se aprecien los cambios):

Contador A	Contador B	Valor en el desfase	Punto en milímetros
24	24	4.2	109.5
...
24	24	4.44	110
...
24	24	3.96	110.5
...
26	25	3.42	111
...
26	25	2.34	111.5
...
26	25	0.34	112
...
26	25	1.12	112.5
...
26	26	2.54	113
...
26	26	3.52	113.5
...
25	25	4.39	114
...
27	26	3.32	115.5
...
25	25	3.91	115
...
27	26	3.32	115.5
...
27	26	1.95	116
...
27	27	0.05	116.5
...
27	26	1.42	117
...
27	27	2.69	117.5
...
27	27	3.36	118
...
26	26	4.35	118.5
...
26	26	4.49	119
...
27	27	3.98	119.5
...
28	27	2.88	120

Tabla 5.1: Tabla de valores del calibrado.

CAPÍTULO 6

Resultados y discusiones

6.1 Detección de la tendencia del potencial a subida o bajada.

De los datos obtenidos en la calibración se tienen distintas mediciones y sus valores característicos, con éstos se busca determinar la distancia de la medición. Analizando los resultados, se observa que el desfase de onda (PSA) cambia al variar la distancia del objeto, ver figura 6.1. Por lo que, el *duty cycle* de la señal que representa desfase (DESF) también cambia, haciendo que se cargue más o menos el condensador C (Figura 3.1), representado por barras azules de figura 3. Cuando la señal tiene un *duty cycle* máximo (50%), el condensador se carga al máximo (5 V que es el valor de 1023 en el convertidor A / D) El gráfico azul representa el aumento o la disminución del voltaje en el condensador, en un rango de distancias consecutivas. La detección de si se encuentra aumentando o disminuyendo el voltaje, se encuentra representada por las barras rojas. Cuando el valor de éstas es 1000 se esta ante una bajada del voltaje y cuando es 100, una subida.

El sistema de medición que se ha desarrollado y programado es capaz de detectar la tendencia a aumento o disminución del voltaje del condensador, gracias al valor en los contadores de *RisingValue* y *FallingValue* explicados en la apartado 4.2.

Si el valor de los contadores son iguales, REMIT está a voltaje bajo y si el contador de *RisingValue* es mayor que el contador de *FallingValue*, está en un voltaje alto. Si REMIT está en un voltaje alto, el desfase está disminuyendo conforme se aumenta la distancia , por lo tanto, el potencial de carga del condensador también disminuye. Lo contrario ocurre si REMIT está en un voltaje bajo. Como se puede ver en la figura 3, a medida que aumenta la distancia, el potencial aumenta y disminuye cada media longitud de onda (4,25 mm). Esto es importante porque se realizan dos curvas de calibración, una para el potencial ascendente y otra para el potencial descendente.

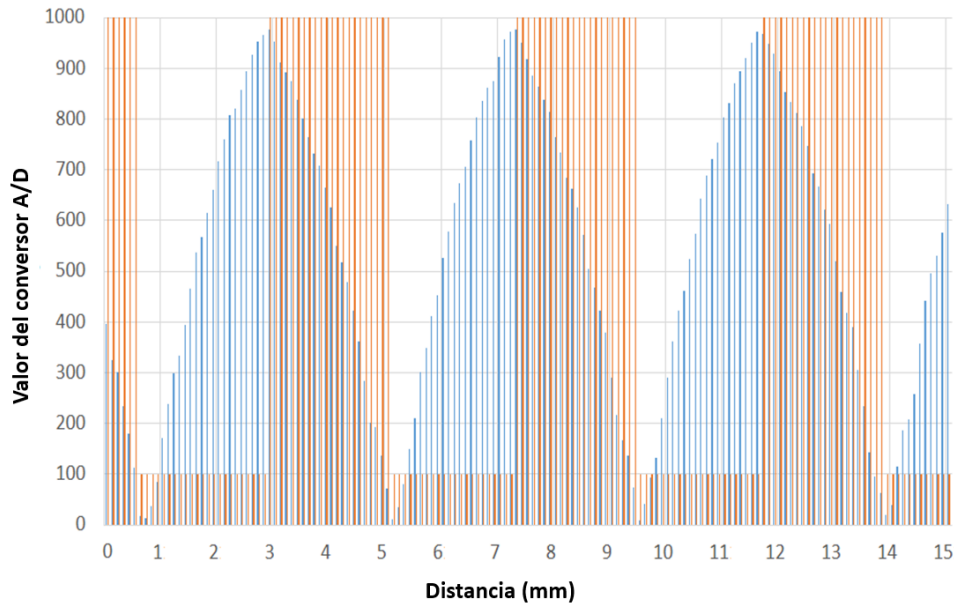


Figura 6.1: En azul el valor del conversor A/D (1024 = 5 Voltios) y en rojo, la detección de potencial de subida (eje vertical = 100) o de bajada (eje vertical = 1000).

6.2 Calibración de los datos.

En la figura 6.2 se muestran los dos ajustes de las regresiones realizadas para los potenciales de subida y bajada. El ajuste resultado debería ser una regresión exponencial. Ésta correspondería a la ecuación del condensador de carga y descarga, no es así ya que, no solo hay un condensador. Existen resistencias en el circuito de detección del PSA, las cual pueden tener algún tipo de auto inducción, el sistema de adquisición de datos (A/D) del microcontrolador también puede introducir cambios.

Sin embargo, la regresión con una ecuación de segundo orden es muy buena ($R^2 > 0,99$).

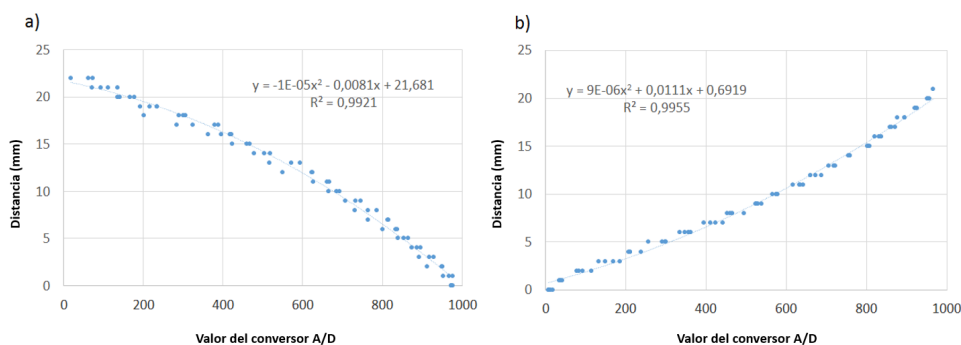


Figura 6.2: Valor del conversor A/D (1024 = 5 Voltios) por (a) potencial de subida, (b) potencial de bajada

6.3 Distancia real y distancia medida.

Para comparar ambos conjuntos de datos (real contra medido), se realiza una regresión lineal de la distancia real y la medida (figura 6.3). En esta regresión, los términos

dependientes deberían tener un valor cercano a uno y el independiente a cero, es decir, mientras mejor haya sido la medida mejor se adaptara la regresión lineal. El coeficiente de determinación de la regresión lineal es de 0,99 (Figura 6.3). Como se puede ver en la figura 6.1, cuando ocurre un cambio en la tendencia del voltaje ascendente o descendente, el valor medido real contra el valor medido, no se ajusta a la recta. Esto lleva a la conclusión de que en estos puntos hay un error mayor, el cual se discute en la siguiente sección.

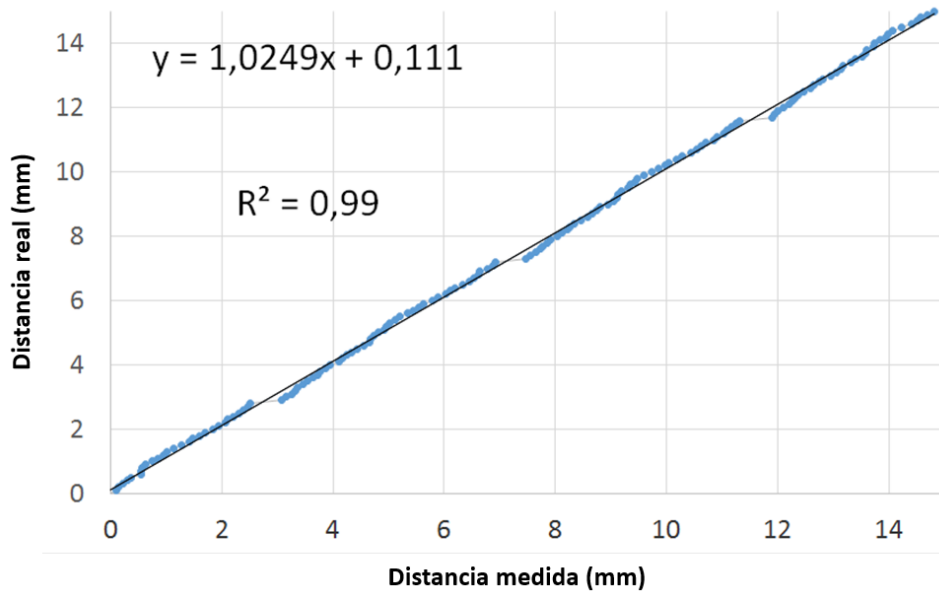


Figura 6.3: Distancia real y medida

6.4 Error absoluto.

En la figura 6.4 se muestran los errores absolutos de las mediciones. Como se puede ver, hay un patrón en cada cambio de fase. Si se compara con la figura 6.4 el error absoluto es mayor en las zonas donde el desfase es máximo o mínimo (en los extremos). Este error podría mejorarse evitando medir a estas distancias, mediante algún mecanismo que acerque o aleje los piezoeléctricos emisores y receptores, una pequeña distancia conocida. Finalmente, se obtiene en este sistema una precisión de $\pm 0,3 \text{ mm}$.

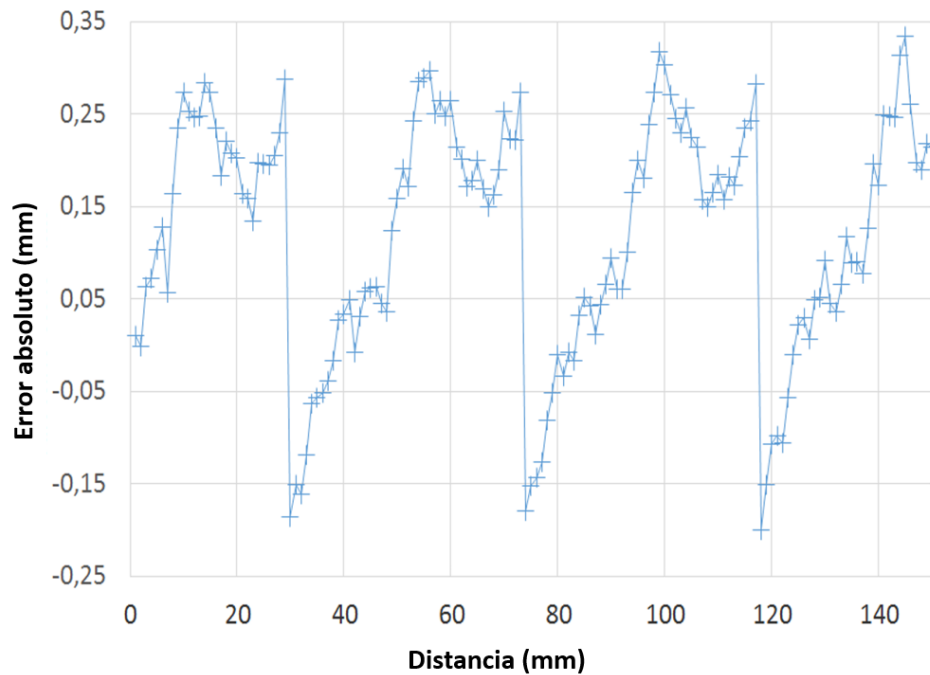


Figura 6.4: Error absoluto

CAPÍTULO 7

Conclusiones

A partir de un módulo de ultrasonidos de bajo coste, se ha conseguido desarrollar un sistema de medición de distancia basado en la evaluación de dos parámetros, el tiempo de vuelo (ToF) y el desfase de señales (PSA). El sistema del módulo de ultrasonidos estaba diseñado en un principio para medir solo el tiempo de vuelo. Para la detección del desfase el hardware se modifica utilizando un condensador. Tomando la información del voltaje de carga del condensador, es posible medir un nuevo parámetro, el desfase de la señal.

Se ha demostrado que la modificación realizada en el hardware del módulo comercial mejora la precisión por debajo del milímetro en comparación con los 3 *mm* originales.

Se ha conseguido desarrollar un robot cartesiano que se desplaza por el plano horizontal XY, el cual sirve para colocar el sensor ultrasónico sobre una superficie de estudio. El sistema desarrollado podrá usarse para el estudio y control de obras de arte antiguas, estudiando la orografía del documento y buscando relaciones con los cambios del entorno.

El proyecto realizado ha sido mucho más complejo de lo que se esperaba en un principio. El estudio de los distintos fenómenos que se generaban en las señales de ultrasonidos era complejo. Se intentaron muchas otras técnicas para mejorar la precisión de las mediciones, pero o no funcionaban o elevaban demasiado el coste del proyecto.

Para alcanzar los objetivos establecidos, se ha estudiado el estado del arte de los dispositivos de medición de distancias. Se llegó a la conclusión de que la mejor metodología y dispositivo para realizar mediciones, eran los ultrasonidos y el módulo HC-SR04, respectivamente. Gracias a las modificaciones hardware y a la metodología propuesta para la medición del desfase se ha conseguido, también, mejorar la precisión de las mediciones. Se han reemplazado y agregado componentes sencillos y baratos al sistema, consiguiendo así un dispositivo low cost. Estudiando y analizando el microcontrolador del módulo de ultrasonidos se ha conseguido sustituir por un microcontrolador específico y programable para el desarrollo del proyecto.

Con respecto al dispositivo robótico, se consigue implementar un robot cartesiano con un cabezal, al que puede acoplarse el sensor de ultrasonidos. El esqueleto del robot son perfiles bosch, correas dentadas y cables. Con solo aumentar la longitud de los perfiles se obtiene un robot con un área de trabajo útil mayor, es decir, se consigue obtener un robot cartesiano escalable.

7.1 Relación del trabajo desarrollado con los estudios cursados.

Se ha cursado el grado de Ingeniería Informática, con especialidad en computación, de la Universidad Politécnica de Valencia.

Los conocimientos adquiridos durante la carrera han servido de mucha ayuda a la hora de la documentación y autoformación en distintas tecnologías y métodos expuestos en el proyecto. Sin embargo, en ningún momento se desarrolla un campo en concreto estudiado en la carrera.

Muchas de las asignaturas cursadas han servido de apoyo al desarrollo del proyecto, como pueden ser:

- Fundamentos físicos de la Informática. Para el análisis físico de todas las señales de ultrasonidos, por la lectura y comprensión de circuitos electrónicos y componentes electrónicos.)
- Arquitectura e ingeniería de computadores. Para la descripción de microcontroladores y funcionamiento.
- Sistemas robotizados. Para la implementación del robot cartesiano XY.
- Programación. Para la extensa programación realizada en todo el proyecto.
- Estadística Para el análisis de los distintos resultados obtenidos.

Bibliografía

- [1] E. R. Moutaye and H. Tap-Beteille. Design of a CMOS APD array for a 3-D camera based on the time of flight distance measurement, *IEEE Instrumentation & Measurement Technology Conference Proceedings*, Austin, TX, 2010, pp. 443-446.
- [2] ToF, Consultado el 20 de mayo de 2017 en: https://en.wikipedia.org/wiki/Time_of_flight
- [3] PSA, Consultado el 20 de mayo de 2017 en: <https://es.wikipedia.org/wiki/Desfase>
- [4] Avanzini, G.B; Ceriani, N.M.; Zanchettin, A.M.; Rocco, P.; Bascetta, L. Safety Control of Industrial Robots Based on a Distributed Distance Sensor, *IEEE Trans. on Control Systems Technology*, 2014, 22(6), 2127 - 2140.
- [5] Zhang, J.; Yang, x.; Song, G.M.; Chen, T.Y.; Zhang, Y. Relative Orientation and Position Detections Based on an RGB-D Sensor and Dynamic Cooperation Strategies for Jumping Sensor Nodes Recycling, *Sensors*, 2015, 15(9), 23618-23639.
- [6] Payá, L.; Amorós, F.; Fernández, L.; Reinoso, O. Performance of Global-Appearance Descriptors in Map Building and Localization Using Omnidirectional Vision, *Sensors*, 2014, 14(2), 3033-3064.
- [7] Lebosse, C.; Renaud, P.; Bayle, B.; de Mathelin. PModeling and Evaluation of Low-Cost Force Sensors, *IEEE Transactions on Robotics*, 2011, 27(4), 815 – 822.
- [8] Baumann, R.; Wilmhurst, D.A. Vision System Sorts Castings at General Motors Canada. Ed. Robot Vision. Bedford, *IFS Ltd*, 1983 , pp. 255 - 266.
- [9] B Le-Anh, T.; De Koster, M.B.M. A review of design and control of automated guided vehicle systems, *European J. of Operational Research*, 2006, 171, 1-23.
- [10] Shan, J; Wang, X. Experimental study on mobile robot navigation using stereo vision, *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, 1958 – 1965.
- [11] Gorostiza, E.M.; Lázaro, J.L.; Meca F.J.; Salido, D., Espinosa, F.; Pallarés, L. Infrared sensor system for mobile-robot positioning in intelligent spaces, *Sensors*, 2011, 11, 5416-5438, DOI: 10.3390/s110505416.
- [12] Rekimoto, J. SmartSkin. An infrastructure for freehand manipulation on interactive surfaces, *Proceedings of the CHI 2002*, ACM Conference, Minneapolis, Minnesota, USA. 2002.
- [13] Lee, H.K.; Chang, S.I.; Yoon, E. E. Dual-mode capacitive proximity sensor for robot application: implementation of tactile and proximity sensing capability on a single polymer platform using shared electrodes, *IEEE Sens. J.*, 2010, 9, 1748–1755.

- [14] Fericean, S.; Droxler, R. New noncontacting inductive analog proximity and inductive linear displacement sensors for industrial automation, *IEEE Sens. J.*, 2007, 7, 1538-1545, DOI 10.1109/JSEN.2007.908232.
- [15] Majchrzak, J.; Michalski, M.; Wiczynski, G. Distance estimation with a long-range ultrasonic sensor system, *IEEE Sens. J.*, 2009, 9, 767 – 773.
- [16] Zhang, H.; Wang, Y.; Zhang Y.; Wang, D. WJin, B. Design and performance analysis of an intrinsically safe ultrasonic ranging sensor, *Sensors*, 1997, 46, 1236-1240.
- [17] Gueuning, F.E.; Varlan, M.; Eugène, C.E.; Dupuis, P. Accurate distance measurement by an autonomous ultrasonic system combining time-of-flight and phase-shift methods *IEEE Trans. on Instrum*, 2016, 16, 867, DOI 10.3390/s16060867.
- [18] HC-SR04 User's Manual, Consultado el 19 de septiembre de 2016 en: https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit
- [19] Open source, Consultado el 20 de mayo de 2017 en: https://es.wikipedia.org/wiki/C%C3%B3digo_abierto
- [20] Hardware libre. Consultado el 20 de mayo de 2017 en: https://es.wikipedia.org/wiki/Hardware_libre
- [21] Max220-Max249. Consultado el 20 de mayo de 2017 en: <https://datasheets.maximintegrated.com/en/ds/MAX220-MAX249.pdf>
- [22] LM324 quad operational amplifier. Consultado el 20 de mayo de 2017 en: <https://www.fairchildsemi.com/datasheets/LM/LM324.pdf>
- [23] EM78P153S. Consultado el 20 de mayo de 2017 en: <http://pdf1.alldatasheet.es/datasheet-pdf/view/118205/EMC/EM78P153S.html>
- [24] Bravo, J.M.; Sánchez-Pérez, J.V.; Ferri, M.; Redondo, J.; Picó, R. Application of Ultrasound Phase-Shift Analysis to Authenticate Wooden Panel Paintings, *Sensors*, 2014, 14, 7992-8002.
- [25] Driver A4988 y motores paso a paso. Consultado el 20 de mayo de 2017 en: <https://www.luisllamas.es/motores-paso-paso-arduino-driver-a4988-drv8825/>
- [26] Micro stepping. Consultado el 20 de mayo de 2017 en: <http://www.nmbtc.com/step-motors/engineering/full-half-and-microstepping/>
- [27] Ingeniería inversa. Consultado el 20 de mayo de 2017 en: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_inversa
- [28] Interrupciones. Consultado el 20 de mayo de 2017 en: <https://www.prometec.net/interrupciones/>

APÉNDICE A

Glosario

En este glosario unicamente se dará una breve descripción de los términos más repetidos y complejos en el proyecto.

- **ToF** Es el tiempo que necesita una onda de ultrasonidos (en este caso), en recorrer una distancia a través de un medio.
- **PSA** El desfase entre dos ondas, es la diferencia de fase de estas. Habitualmente, esta diferencia de fases, se mide en un instante en concreto. El desfase indica como de desplazada está la función horizontalmente a la derecha de la posición que debería tener.

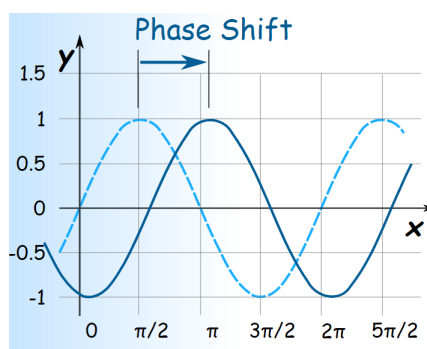


Figura A.1: Desfase de ondas, PSA.

- **Piezoeléctrico** Los piezoeléctricos convierten la tensión mecánica en electricidad, y la electricidad en vibraciones mecánicas. Es una propiedad de determinados cristales. Su masa adquiere una polarización eléctrica y aparece una diferencia de potencial y cargas eléctricas en su superficie.
- **Señal analógica** Una señal analógica es un tipo de señal generada por un fenómeno electromagnético. Se puede representar por una función matemática continua en la que es variable su amplitud y periodo.
- **Señal digital** Son aquellas señales que tienen una variación discontinua en el tiempo, y que pueden representar solamente un número limitado de valores discretos, en lugar de valores dentro de un cierto rango.
- **Amplificador operacional** Es un dispositivo amplificador electrónico de alta ganancia acoplado en corriente continua que tiene dos entradas y una salida.

- **Duty cycle** O ciclo de trabajo, es la relación que existe entre el tiempo en que la señal se encuentra en estado activo y el periodo de la misma. Su valor se encuentra comprendido entre 0 y 1, en porcentajes.
- **Diagrama de flujo** Es la representación gráfica de un algoritmo o proceso. Estos diagramas muestran los pasos y el comportamiento seguido por algún algoritmo o proceso.
- **Tornillo micrómetro** En una herramienta que sirve para valorar el tamaño de un objeto con gran precisión, en un rango del orden de centésimas o de milésimas de milímetro.

APÉNDICE B

Timers del microcontrolador
ATmega32u4

12. Timer/Counter0, Timer/Counter1, and Timer/Counter3 Prescalers

Timer/Counter0, 1, and 3 share the same prescaler module, but the Timer/Counters can have different prescaler settings. The description below applies to all Timer/Counters. Tn is used as a general name, n = 0, 1, or 3.

12.1 Internal Clock Source

The Timer/Counter can be clocked directly by the system clock (by setting the CSn2:0 = 1). This provides the fastest operation, with a maximum Timer/Counter clock frequency equal to system clock frequency ($f_{CLK_I/O}$). Alternatively, one of four taps from the prescaler can be used as a clock source. The prescaled clock has a frequency of either $f_{CLK_I/O}/8$, $f_{CLK_I/O}/64$, $f_{CLK_I/O}/256$, or $f_{CLK_I/O}/1024$.

12.2 Prescaler Reset

The prescaler is free running, i.e., operates independently of the Clock Select logic of the Timer/Counter, and it is shared by the Timer/Counter Tn. Since the prescaler is not affected by the Timer/Counter's clock select, the state of the prescaler will have implications for situations where a prescaled clock is used. One example of prescaling artifacts occurs when the timer is enabled and clocked by the prescaler ($6 > CSn2:0 > 1$). The number of system clock cycles from when the timer is enabled to the first count occurs can be from 1 to N+1 system clock cycles, where N equals the prescaler divisor (8, 64, 256, or 1024).

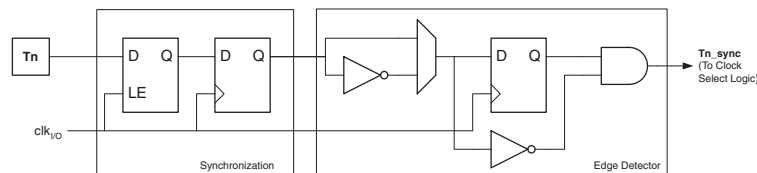
It is possible to use the prescaler reset for synchronizing the Timer/Counter to program execution. However, care must be taken if the other Timer/Counter that shares the same prescaler also uses prescaling. A prescaler reset will affect the prescaler period for all Timer/Counters it is connected to.

12.3 External Clock Source

An external clock source applied to the Tn pin can be used as Timer/Counter clock (clk_{Tn}). The Tn pin is sampled once every system clock cycle by the pin synchronization logic. The synchronized (sampled) signal is then passed through the edge detector. Figure 12-1 shows a functional equivalent block diagram of the Tn synchronization and edge detector logic. The registers are clocked at the positive edge of the internal system clock ($clk_{I/O}$). The latch is transparent in the high period of the internal system clock.

The edge detector generates one clk_{Tn} pulse for each positive ($CSn2:0 = 7$) or negative ($CSn2:0 = 6$) edge it detects.

Figure 12-1. Tn/T0 Pin Sampling



The synchronization and edge detector logic introduces a delay of 2.5 to 3.5 system clock cycles from an edge has been applied to the Tn pin to the counter is updated.

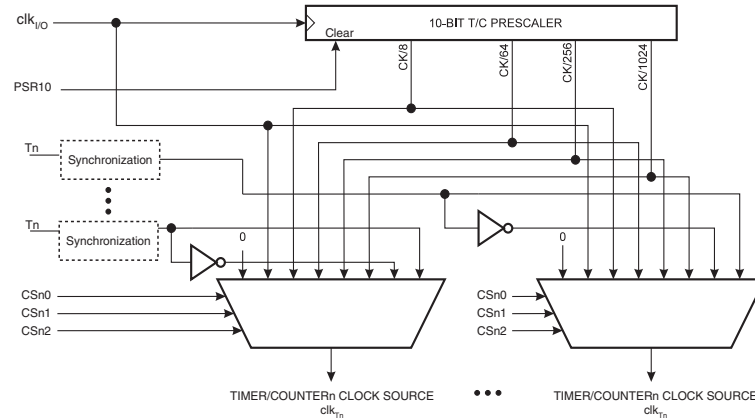
Enabling and disabling of the clock input must be done when Tn has been stable for at least one system clock cycle, otherwise it is a risk that a false Timer/Counter clock pulse is generated.

Each half period of the external clock applied must be longer than one system clock cycle to ensure correct sampling. The external clock must be guaranteed to have less than half the system clock frequency ($f_{ExtClk} < f_{clk_I/O}/2$) given a 50/50% duty cycle. Since the edge detector uses sampling, the maximum frequency of an external clock it can detect is half the sampling frequency (Nyquist sampling theorem). However, due to variation of the system clock frequency and duty cycle caused by Oscillator source (crystal, resonator, and

capacitors) tolerances, it is recommended that maximum frequency of an external clock source is less than $f_{clk_I/O}/2.5$.

An external clock source can not be prescaled.

Figure 12-2. Prescaler for Synchronous Timer/Counters



Note: T3 input is not available on the [ATmega16U4/ATmega32U4](#) products. “Tn” only refers to either T0 or T1 inputs.

12.4 Register Description

12.4.1 General Timer/Counter Control Register – GTCCR

Bit	7	6	5	4	3	2	1	0	
	TSM	-	-	-	-	-	PSRASY	PSRSYNC	GTCCR
Read/Write	R/W	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – TSM: Timer/Counter Synchronization Mode**

Writing the TSM bit to one activates the Timer/Counter Synchronization mode. In this mode, the value that is written to the PSRASY and PSRSYNC bits is kept, hence keeping the corresponding prescaler reset signals asserted. This ensures that the corresponding Timer/Counters are halted and can be configured to the same value without the risk of one of them advancing during configuration. When the TSM bit is written to zero, the PSRASY and PSRSYNC bits are cleared by hardware, and the Timer/Counters start counting simultaneously.

- Bit 0 – PSRSYNC: Prescaler Reset for Synchronous Timer/Counters**

When this bit is one, Timer/Counter0 and Timer/Counter1 and Timer/Counter3 prescaler will be Reset. This bit is normally cleared immediately by hardware, except if the TSM bit is set. Note that Timer/Counter0, Timer/Counter1 and Timer/Counter3 share the same prescaler and a reset of this prescaler will affect all timers.

14. 16-bit Timers/Counters (Timer/Counter1 and Timer/Counter3)

The 16-bit Timer/Counter unit allows accurate program execution timing (event management), wave generation, and signal timing measurement. The main features are:

- True 16-bit Design (i.e., Allows 16-bit PWM)
- Three independent Output Compare Units
- Double Buffered Output Compare Registers
- One Input Capture Unit
- Input Capture Noise Canceler
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- External Event Counter
- Ten independent interrupt sources (TOV1, OCF1A, OCF1B, OCF1C, ICF1, TOV3, OCF3A, OCF3B, OCF3C, and ICF3)

14.1 Overview

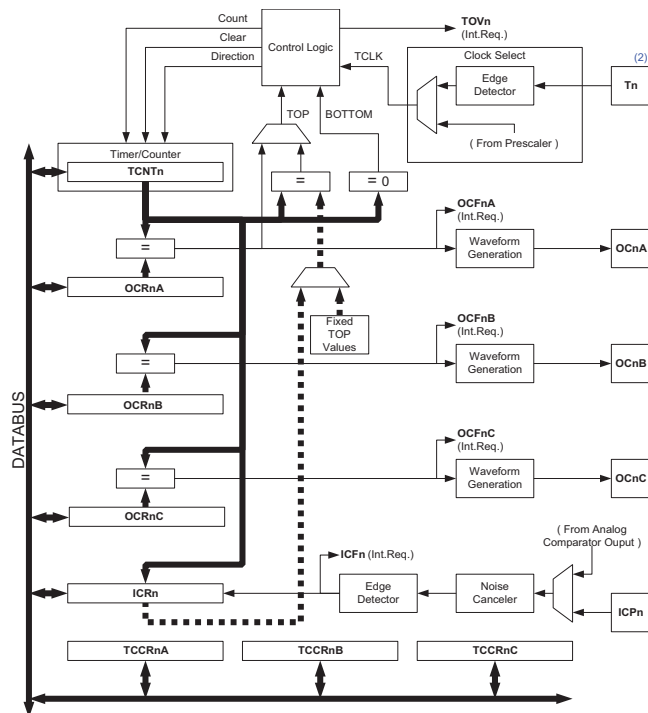
Most register and bit references in this section are written in general form. A lower case “n” replaces the Timer/Counter number, and a lower case “x” replaces the Output Compare unit channel. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT1 for accessing Timer/Counter1 counter value and so on.

A simplified block diagram of the 16-bit Timer/Counter is shown in [Figure 14-1 on page 112](#). For the actual placement of I/O pins, see [“Pinout” on page 3](#). CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the [“16-bit Timers/Counters \(Timer/Counter1 and Timer/Counter3\)” on page 111](#).

The Power Reduction Timer/Counter1 bit, PRTIM1, in [“Power Reduction Register 0 - PRR0” on page 47](#) must be written to zero to enable Timer/Counter1 module.

The Power Reduction Timer/Counter3 bit, PRTIM3, in [“Power Reduction Register 1 - PRR1” on page 48](#) must be written to zero to enable Timer/Counter3 module.

Figure 14-1. 16-bit Timer/Counter Block Diagram⁽¹⁾



- Note:
1. Refer to "Pinout" on page 3, Table 10-3 on page 74, and Table 10-6 on page 77 for Timer/Counter1 and 3 and 3 pin placement and description.
 2. Tn only refers to T1 since T3 input is not available on the product.

14.1.1 Registers

The Timer/Counter (TCNTn), Output Compare Registers (OCRnA/B/C), and Input Capture Register (ICRn) are all 16-bit registers. Special procedures must be followed when accessing the 16-bit registers. These procedures are described in the section "Accessing 16-bit Registers" on page 113. The Timer/Counter Control Registers (TCCRnA/B/C) are 8-bit registers and have no CPU access restrictions. Interrupt requests (shorten as Int.Req.) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn). TIFRn and TIMSKn are not shown in the figure since these registers are shared by other timer units.

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the Tn pin. The Clock Select logic block controls which clock source and edge the Timer/Counter uses to increment (or decrement) its value. The Timer/Counter is inactive when no clock source is selected. The output from the clock select logic is referred to as the timer clock (clk_{Tn}).

The double buffered Output Compare Registers (OCRnA/B/C) are compared with the Timer/Counter value at all time. The result of the compare can be used by the Waveform Generator to generate a PWM or variable frequency output on the Output Compare pin (OCnA/B/C). See "Output Compare Units" on page 119. The compare match event will also set the Compare Match Flag (OCFnA/B/C) which can be used to generate an Output Compare interrupt request.

The Input Capture Register can capture the Timer/Counter value at a given external (edge triggered) event on either the Input Capture pin (ICPn) or on the Analog Comparator pins (See “Analog Comparator” on page 293.) The Input Capture unit includes a digital filtering unit (Noise Canceler) for reducing the chance of capturing noise spikes.

The TOP value, or maximum Timer/Counter value, can in some modes of operation be defined by either the OCRnA Register, the ICRn Register, or by a set of fixed values. When using OCRnA as TOP value in a PWM mode, the OCRnA Register can not be used for generating a PWM output. However, the TOP value will in this case be double buffered allowing the TOP value to be changed in run time. If a fixed TOP value is required, the ICRn Register can be used as an alternative, freeing the OCRnA to be used as PWM output.

14.1.2 Definitions

The following definitions are used extensively throughout the document:

BOTTOM	The counter reaches the <i>BOTTOM</i> when it becomes 0x0000.
MAX	The counter reaches its <i>MAX</i> imum when it becomes 0xFFFF (decimal 65535).
TOP	The counter reaches the <i>TOP</i> when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be one of the fixed values: 0x00FF, 0x01FF, or 0x03FF, or to the value stored in the OCRnA or ICRn Register. The assignment is dependent of the mode of operation.

14.2 Accessing 16-bit Registers

The TCNTn, OCRnA/B/C, and ICRn are 16-bit registers that can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. The same Temporary Register is shared between all 16-bit registers within each 16-bit timer. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

Not all 16-bit accesses uses the Temporary Register for the high byte. Reading the OCRnA/B/C 16-bit registers does not involve using the Temporary Register.

To do a 16-bit write, the high byte must be written before the low byte. For a 16-bit read, the low byte must be read before the high byte.

The following code examples show how to access the 16-bit timer registers assuming that no interrupts updates the temporary register. The same principle can be used directly for accessing the OCRnA/B/C and ICRn Registers. Note that when using “C”, the compiler handles the 16-bit access.

Assembly Code Examples⁽¹⁾

```
...
; Set TCNTn to 0x01FF
ldi    r17,0x01
ldi    r16,0xFF
out    TCNTnH,r17
out    TCNTnL,r16
; Read TCNTn into r17:r16
in     r16,TCNTnL
in     r17,TCNTnH
...
```

C Code Examples⁽¹⁾

```
unsigned int i;
...
/* Set TCNTn to 0x01FF */
TCNTn = 0x1FF;
/* Read TCNTn into i */
i = TCNTn;
...
```

Note: 1. See [“Code Examples” on page 8](#).

The assembly code example returns the TCNTn value in the r17:r16 register pair.

It is important to notice that accessing 16-bit registers are atomic operations. If an interrupt occurs between the two instructions accessing the 16-bit register, and the interrupt code updates the temporary register by accessing the same or any other of the 16-bit Timer Registers, then the result of the access outside the interrupt will be corrupted. Therefore, when both the main code and the interrupt code update the temporary register, the main code must disable the interrupts during the 16-bit access.

The following code examples show how to do an atomic read of the TCNTn Register contents. Reading any of the OCRnA/B/C or ICRn Registers can be done by using the same principle.

Assembly Code Example⁽¹⁾

```
TIM16_ReadTCNTn:
; Save global interrupt flag
in      r18,SREG
; Disable interrupts
cli
; Read TCNTn into r17:r16
in      r16,TCNTnL
in      r17,TCNTnH
; Restore global interrupt flag
out     SREG,r18
ret
```

C Code Example⁽¹⁾

```
unsigned int TIM16_ReadTCNTn( void )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    __disable_interrupt();
    /* Read TCNTn into i */
    i = TCNTn;
    /* Restore global interrupt flag */
    SREG = sreg;
    return i;
}
```

Note: 1. See "Code Examples" on page 8.

The assembly code example returns the TCNTn value in the r17:r16 register pair.

The following code examples show how to do an atomic write of the TCNTn Register contents. Writing any of the OCRnA/B/C or ICRn Registers can be done by using the same principle.

Assembly Code Example⁽¹⁾

```
TIM16_WriteTCNTn:
; Save global interrupt flag
in      r18,SREG
; Disable interrupts
cli
; Set TCNTn to r17:r16
out     TCNTnH,r17
out     TCNTnL,r16
; Restore global interrupt flag
out     SREG,r18
ret
```

C Code Example⁽¹⁾

```
void TIM16_WriteTCNTn( unsigned int i )
{
    unsigned char sreg;
    unsigned int i;
    /* Save global interrupt flag */
    sreg = SREG;
    /* Disable interrupts */
    __disable_interrupt();
    /* Set TCNTn to i */
    TCNTn = i;
    /* Restore global interrupt flag */
    SREG = sreg;
}
```

Note: 1. See “Code Examples” on page 8.

The assembly code example requires that the r17:r16 register pair contains the value to be written to TCNTn.

14.2.1 Reusing the Temporary High Byte Register

If writing to more than one 16-bit register where the high byte is the same for all registers written, then the high byte only needs to be written once. However, note that the same rule of atomic operation described previously also applies in this case.

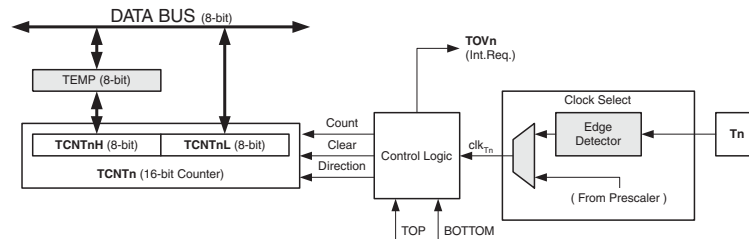
14.3 Timer/Counter Clock Sources

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the *Clock Select* (CSn2:0) bits located in the *Timer/Counter control Register B* (TCCRnB). For details on clock sources and prescaler, see “[Timer/Counter0](#), [Timer/Counter1](#), and [Timer/Counter3 Prescalers](#)” on page 92.

14.4 Counter Unit

The main part of the 16-bit Timer/Counter is the programmable 16-bit bi-directional counter unit. [Figure 14-2](#) shows a block diagram of the counter and its surroundings.

Figure 14-2. Counter Unit Block Diagram



Signal description (internal signals):

Count: Increment or decrement TCNTn by 1.

Direction: Select between increment and decrement.

Clear: Clear TCNTn (set all bits to zero).

clk_{Tn}: Timer/Counter clock.

TOP: Signalize that TCNTn has reached maximum value.

BOTTOM: Signalize that TCNTn has reached minimum value (zero).

The 16-bit counter is mapped into two 8-bit I/O memory locations: *Counter High* (TCNTnH) containing the upper eight bits of the counter, and *Counter Low* (TCNTnL) containing the lower eight bits. The TCNTnH Register can only be indirectly accessed by the CPU. When the CPU does an access to the TCNTnH I/O location, the CPU accesses the high byte temporary register (TEMP). The temporary register is updated with the TCNTnH value when the TCNTnL is read, and TCNTnH is updated with the temporary register value when TCNTnL is written. This allows the CPU to read or write the entire 16-bit counter value within one clock cycle via the 8-bit data bus. It is important to notice that there are special cases of writing to the TCNTn Register when the counter is counting that will give unpredictable results. The special cases are described in the sections where they are of importance.

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each *timer clock* (clk_{Tn}). The clk_{Tn} can be generated from an external or internal clock source, selected by the *Clock Select* bits (CSn2:0). When no clock source is selected (CSn2:0 = 0) the timer is stopped. However, the TCNTn value can be accessed by the CPU, independent of whether clk_{Tn} is present or not. A CPU write overrides (has priority over) all counter clear or count operations.

The counting sequence is determined by the setting of the *Waveform Generation mode* bits (WGMn3:0) located in the *Timer/Counter Control Registers A and B* (TCCRnA and TCCRnB). There are close connections between how the counter behaves (counts) and how waveforms are generated on the Output Compare outputs OCnx. For more details about advanced counting sequences and waveform generation, see ["Modes of Operation" on page 122](#).

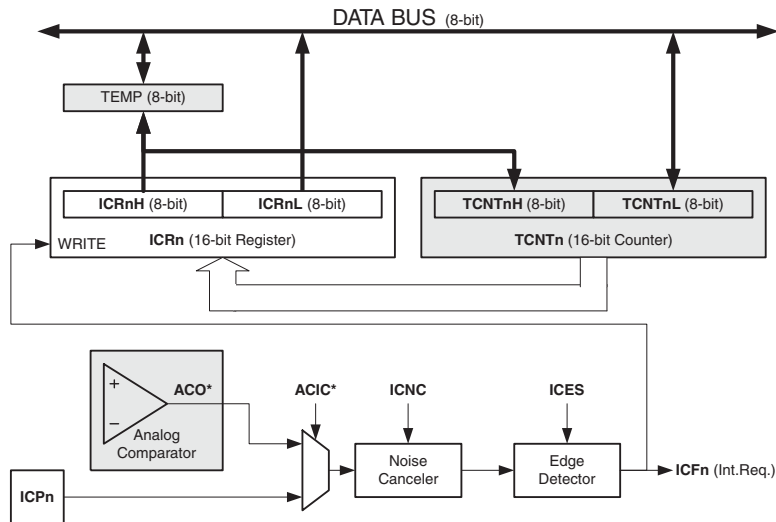
The Timer/Counter Overflow Flag (TOVn) is set according to the mode of operation selected by the WGMn3:0 bits. TOVn can be used for generating a CPU interrupt.

14.5 Input Capture Unit

The Timer/Counter incorporates an input capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICPn pin or alternatively, for the Timer/Counter1 only, via the Analog Comparator unit. The time-stamps can then be used to calculate frequency, duty-cycle, and other features of the signal applied. Alternatively the time-stamps can be used for creating a log of the events.

The Input Capture unit is illustrated by the block diagram shown in Figure 14-3. The elements of the block diagram that are not directly a part of the input capture unit are gray shaded. The small “n” in register and bit names indicates the Timer/Counter number.

Figure 14-3. Input Capture Unit Block Diagram



Note: The Analog Comparator Output (ACO) can only trigger the Timer/Counter1 ICP – not Timer/Counter3, 4, or 5. When a change of the logic level (an event) occurs on the *Input Capture Pin* (ICPn), alternatively on the *analog Comparator output* (ACO), and this change confirms to the setting of the edge detector, a capture will be triggered. When a capture is triggered, the 16-bit value of the counter (TCNTn) is written to the *Input Capture Register* (ICRn). The *Input Capture Flag* (ICFn) is set at the same system clock as the TCNTn value is copied into ICRn Register. If enabled (TICIE_n = 1), the input capture flag generates an input capture interrupt. The ICFn flag is automatically cleared when the interrupt is executed. Alternatively the ICFn flag can be cleared by software by writing a logical one to its I/O bit location.


Reading the 16-bit value in the *Input Capture Register* (ICRn) is done by first reading the low byte (ICRnL) and then the high byte (ICRnH). When the low byte is read the high byte is copied into the high byte Temporary Register (TEMP). When the CPU reads the ICRnH I/O location it will access the TEMP Register.

The ICRn Register can only be written when using a Waveform Generation mode that utilizes the ICRn Register for defining the counter’s TOP value. In these cases the *Waveform Generation mode* (WGMn3:0) bits must be set before the TOP value can be written to the ICRn Register. When writing the ICRn Register the high byte must be written to the ICRnH I/O location before the low byte is written to ICRnL.

For more information on how to access the 16-bit registers refer to “[Accessing 16-bit Registers](#)” on page 113.

14.5.1 Input Capture Trigger Source

The main trigger source for the input capture unit is the *Input Capture Pin* (ICPn). Timer/Counter1 can alternatively use the analog comparator output as trigger source for the input capture unit. The Analog Comparator is selected as trigger source by setting the *analog Comparator Input Capture* (ACIC) bit in the *Analog Comparator Control and Status Register* (ACSR). Be aware that changing trigger source can trigger a capture. The input capture flag must therefore be cleared after the change.



Both the *Input Capture Pin* (ICPn) and the *Analog Comparator output* (ACO) inputs are sampled using the same technique as for the Tn pin (Figure 12-1 on page 92). The edge detector is also identical. However, when the noise canceler is enabled, additional logic is inserted before the edge detector, which increases the delay by four system clock cycles. Note that the input of the noise canceler and edge detector is always enabled unless the Timer/Counter is set in a Waveform Generation mode that uses ICRn to define TOP.

An input capture can be triggered by software by controlling the port of the ICPn pin.

14.5.2 Noise Canceler

The noise canceler improves noise immunity by using a simple digital filtering scheme. The noise canceler input is monitored over four samples, and all four must be equal for changing the output that in turn is used by the edge detector.

The noise canceler is enabled by setting the *Input Capture Noise Canceler* (ICNCn) bit in *Timer/Counter Control Register B* (TCCRnB). When enabled the noise canceler introduces additional four system clock cycles of delay from a change applied to the input, to the update of the ICRn Register. The noise canceler uses the system clock and is therefore not affected by the prescaler.

14.5.3 Using the Input Capture Unit

The main challenge when using the Input Capture unit is to assign enough processor capacity for handling the incoming events. The time between two events is critical. If the processor has not read the captured value in the ICRn Register before the next event occurs, the ICRn will be overwritten with a new value. In this case the result of the capture will be incorrect.

When using the Input Capture interrupt, the ICRn Register should be read as early in the interrupt handler routine as possible. Even though the Input Capture interrupt has relatively high priority, the maximum interrupt response time is dependent on the maximum number of clock cycles it takes to handle any of the other interrupt requests.

Using the Input Capture unit in any mode of operation when the TOP value (resolution) is actively changed during operation, is not recommended.

Measurement of an external signal's duty cycle requires that the trigger edge is changed after each capture. Changing the edge sensing must be done as early as possible after the ICRn Register has been read. After a change of the edge, the Input Capture Flag (ICFn) must be cleared by software (writing a logical one to the I/O bit location). For measuring frequency only, the clearing of the ICFn Flag is not required (if an interrupt handler is used).

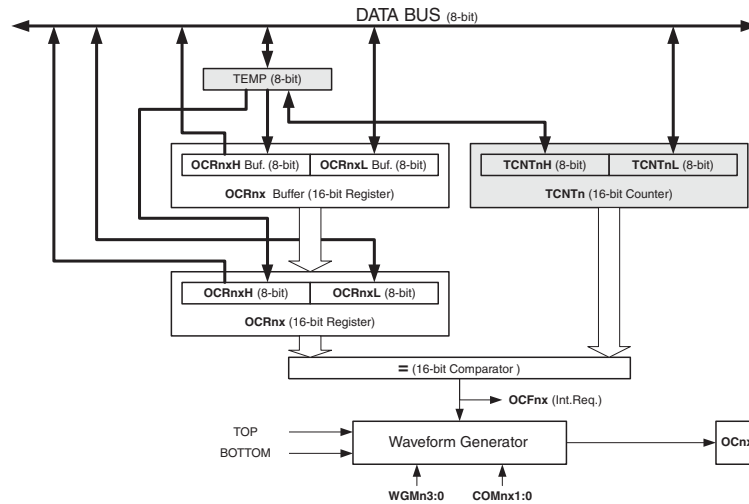
14.6 Output Compare Units

The 16-bit comparator continuously compares TCNTn with the *Output Compare Register* (OCRnx). If TCNT equals OCRnx the comparator signals a match. A match will set the *Output Compare Flag* (OCFnx) at the next timer clock cycle. If enabled (OCIEnx = 1), the Output Compare Flag generates an Output Compare interrupt. The OCFnx Flag is automatically cleared when the interrupt is executed. Alternatively the OCFnx Flag can be cleared by software by writing a logical one to its I/O bit location. The Waveform Generator uses the match signal to generate an output according to operating mode set by the *Waveform Generation mode* (WGMn3:0) bits and *Compare Output mode* (COMnx1:0) bits. The TOP and BOTTOM signals are used by the Waveform Generator for handling the special cases of the extreme values in some modes of operation (See “Modes of Operation” on page 122.)

A special feature of Output Compare unit A allows it to define the Timer/Counter TOP value (i.e., counter resolution). In addition to the counter resolution, the TOP value defines the period time for waveforms generated by the Waveform Generator.

Figure 14-4 shows a block diagram of the Output Compare unit. The small “n” in the register and bit names indicates the device number (n = n for Timer/Counter n), and the “x” indicates Output Compare unit (A/B/C). The elements of the block diagram that are not directly a part of the Output Compare unit are gray shaded.

Figure 14-4. Output Compare Unit, Block Diagram



The OCRnx Register is double buffered when using any of the twelve *Pulse Width Modulation* (PWM) modes. For the Normal and *Clear Timer on Compare* (CTC) modes of operation, the double buffering is disabled. The double buffering synchronizes the update of the OCRnx Compare Register to either TOP or BOTTOM of the counting sequence. The synchronization prevents the occurrence of odd-length, non-symmetrical PWM pulses, thereby making the output glitch-free.

The OCRnx Register access may seem complex, but this is not case. When the double buffering is enabled, the CPU has access to the OCRnx Buffer Register, and if double buffering is disabled the CPU will access the OCRnx directly. The content of the OCR1x (Buffer or Compare) Register is only changed by a write operation (the Timer/Counter does not update this register automatically as the TCNT1 and ICR1 Register). Therefore OCR1x is not read via the high byte temporary register (TEMP). However, it is a good practice to read the low byte first as when accessing other 16-bit registers. Writing the OCRnx Registers must be done via the TEMP Register since the compare of all 16 bits is done continuously. The high byte (OCRnxH) has to be written first. When the high byte I/O location is written by the CPU, the TEMP Register will be updated by the value written. Then when the low byte (OCRnxL) is written to the lower eight bits, the high byte will be copied into the upper 8-bits of either the OCRnx buffer or OCRnx Compare Register in the same system clock cycle.

For more information of how to access the 16-bit registers refer to [“Accessing 16-bit Registers” on page 113](#).

14.6.1 Force Output Compare

In non-PWM Waveform Generation modes, the match output of the comparator can be forced by writing a one to the *Force Output Compare* (FOCnx) bit. Forcing compare match will not set the OCFnx Flag or reload/clear the timer, but the OCnx pin will be updated as if a real compare match had occurred (the COMn1:0 bits settings define whether the OCnx pin is set, cleared or toggled).

14.6.2 Compare Match Blocking by TCNTn Write

All CPU writes to the TCNTn Register will block any compare match that occurs in the next timer clock cycle, even when the timer is stopped. This feature allows OCRnx to be initialized to the same value as TCNTn without triggering an interrupt when the Timer/Counter clock is enabled.

14.6.3 Using the Output Compare Unit

Since writing TCNTn in any mode of operation will block all compare matches for one timer clock cycle, there are risks involved when changing TCNTn when using any of the Output Compare channels, independent of whether the Timer/Counter is running or not. If the value written to TCNTn equals the OCRnx value, the compare match will be missed, resulting in incorrect waveform generation. Do not write the TCNTn equal to TOP in PWM modes with variable TOP values. The compare match for the TOP will be ignored and the counter will continue to 0xFFFF. Similarly, do not write the TCNTn value equal to BOTTOM when the counter is down-counting.

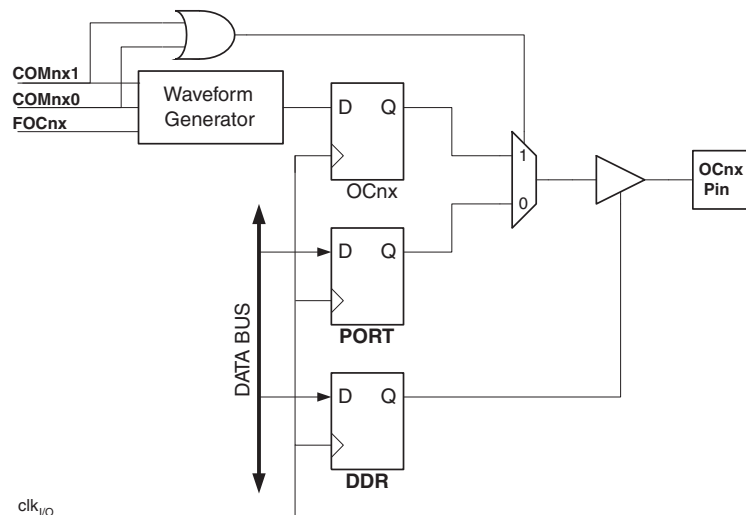
The setup of the OCnx should be performed before setting the Data Direction Register for the port pin to output. The easiest way of setting the OCnx value is to use the Force Output Compare (FOCnx) strobe bits in Normal mode. The OCnx Register keeps its value even when changing between Waveform Generation modes.

Be aware that the COMnx1:0 bits are not double buffered together with the compare value. Changing the COMnx1:0 bits will take effect immediately.

14.7 Compare Match Output Unit

The Compare Output mode (COMnx1:0) bits have two functions. The Waveform Generator uses the COMnx1:0 bits for defining the Output Compare (OCnx) state at the next compare match. Secondly the COMnx1:0 bits control the OCnx pin output source. Figure 14-5 shows a simplified schematic of the logic affected by the COMnx1:0 bit setting. The I/O Registers, I/O bits, and I/O pins in the figure are shown in bold. Only the parts of the general I/O Port Control Registers (DDR and PORT) that are affected by the COMnx1:0 bits are shown. When referring to the OCnx state, the reference is for the internal OCnx Register, not the OCnx pin. If a system reset occur, the OCnx Register is reset to "0".

Figure 14-5. Compare Match Output Unit, Schematic



The general I/O port function is overridden by the Output Compare (OCnx) from the Waveform Generator if either of the COMnx1:0 bits are set. However, the OCnx pin direction (input or output) is still controlled by the *Data Direction Register* (DDR) for the port pin. The Data Direction Register bit for the OCnx pin (DDR_OCnx) must be set as output before the OCnx value is visible on the pin. The port override function is generally independent of the Waveform Generation mode, but there are some exceptions. Refer to [Table 14-1 on page 131](#), [Table 14-2 on page 132](#), and [Table 14-3 on page 132](#) for details.

The design of the Output Compare pin logic allows initialization of the OCnx state before the output is enabled. Note that some COMnx1:0 bit settings are reserved for certain modes of operation. See [“16-bit Timers/Counters \(Timer/Counter1 and Timer/Counter3\)” on page 111](#).

The COMnx1:0 bits have no effect on the Input Capture unit.

14.7.1 Compare Output Mode and Waveform Generation

The Waveform Generator uses the COMnx1:0 bits differently in normal, CTC, and PWM modes. For all modes, setting the COMnx1:0 = 0 tells the Waveform Generator that no action on the OCnx Register is to be performed on the next compare match. For compare output actions in the non-PWM modes refer to [Table 14-1 on page 131](#). For fast PWM mode refer to [Table 14-2 on page 132](#), and for phase correct and phase and frequency correct PWM refer to [Table 14-3 on page 132](#).

A change of the COMnx1:0 bits state will have effect at the first compare match after the bits are written. For non-PWM modes, the action can be forced to have immediate effect by using the FOCnx strobe bits.

14.8 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the *Waveform Generation mode* (WGMn3:0) and *Compare Output mode* (COMnx1:0) bits. The Compare Output mode bits do not affect the counting sequence, while the Waveform Generation mode bits do. The COMnx1:0 bits control whether the PWM output generated should be inverted or not (inverted or non-inverted PWM). For non-PWM modes the COMnx1:0 bits control whether the output should be set, cleared or toggle at a compare match (See [“Compare Match Output Unit” on page 121](#).)

For detailed timing information refer to [“Timer/Counter Timing Diagrams” on page 129](#).

14.8.1 Normal Mode

The simplest mode of operation is the *Normal mode* (WGMn3:0 = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 16-bit value (MAX = 0xFFFF) and then restarts from the BOTTOM (0x0000). In normal operation the *Timer/Counter Overflow Flag* (TOVn) will be set in the same timer clock cycle as the TCNTn becomes zero. The TOVn Flag in this case behaves like a 17th bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOVn Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

The Input Capture unit is easy to use in Normal mode. However, observe that the maximum interval between the external events must not exceed the resolution of the counter. If the interval between events are too long, the timer overflow interrupt or the prescaler must be used to extend the resolution for the capture unit.

The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

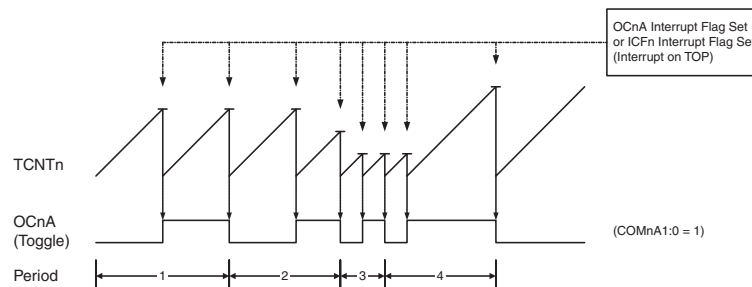
14.8.2 Clear Timer on Compare Match (CTC) Mode

In *Clear Timer on Compare* or CTC mode (WGMn3:0 = 4 or 12), the OCRnA or ICRn Register are used to manipulate the counter resolution. In CTC mode the counter is cleared to zero when the counter value (TCNTn)

matches either the OCRnA (WGMn3:0 = 4) or the ICRn (WGMn3:0 = 12). The OCRnA or ICRn define the top value for the counter, hence also its resolution. This mode allows greater control of the compare match output frequency. It also simplifies the operation of counting external events.

The timing diagram for the CTC mode is shown in Figure 14-6. The counter value (TCNTn) increases until a compare match occurs with either OCRnA or ICRn, and then counter (TCNTn) is cleared.

Figure 14-6. CTC Mode, Timing Diagram



An interrupt can be generated at each time the counter value reaches the TOP value by either using the OCFnA or ICFn Flag according to the register used to define the TOP value. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value. However, changing the TOP to a value close to BOTTOM when the counter is running with none or a low prescaler value must be done with care since the CTC mode does not have the double buffering feature. If the new value written to OCRnA or ICRn is lower than the current value of TCNTn, the counter will miss the compare match. The counter will then have to count to its maximum value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. In many cases this feature is not desirable. An alternative will then be to use the fast PWM mode using OCRnA for defining TOP (WGMn3:0 = 15) since the OCRnA then will be double buffered.

For generating a waveform output in CTC mode, the OCnA output can be set to toggle its logical level on each compare match by setting the Compare Output mode bits to toggle mode (COMnA1:0 = 1). The OCnA value will not be visible on the port pin unless the data direction for the pin is set to output (DDR_OCnA = 1). The waveform generated will have a maximum frequency of $f_{OCnA} = f_{clk_I/O} / 2$ when OCRnA is set to zero (0x0000). The waveform frequency is defined by the following equation:

$$f_{OCnA} = \frac{f_{clk_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

The N variable represents the prescaler factor (1, 8, 64, 256, or 1024).

As for the Normal mode of operation, the TOVn Flag is set in the same timer clock cycle that the counter counts from MAX to 0x0000.

14.8.3 Fast PWM Mode

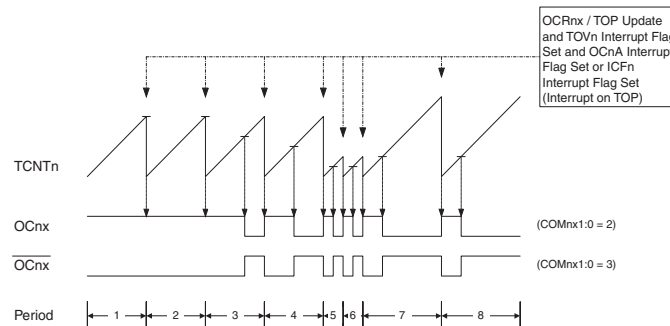
The *fast Pulse Width Modulation* or fast PWM mode (WGMn3:0 = 5, 6, 7, 14, or 15) provides a high frequency PWM waveform generation option. The fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is set on the compare match between TCNTn and OCRnx, and cleared at TOP. In inverting Compare Output mode output is cleared on compare match and set at TOP. Due to the single-slope operation, the operating frequency of the fast PWM mode can be twice as high as the phase correct and phase and frequency correct PWM modes that use dual-slope operation. This high frequency makes the fast PWM mode well suited for power regulation, rectification, and DAC applications. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost.

The PWM resolution for fast PWM can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 5, 6, or 7), the value in ICRn (WGMn3:0 = 14), or the value in OCRnA (WGMn3:0 = 15). The counter is then cleared at the following timer clock cycle. The timing diagram for the fast PWM mode is shown in Figure 14-7. The figure shows fast PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the single-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx Interrupt Flag will be set when a compare match occurs.

Figure 14-7. Fast PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches TOP. In addition the OCnA or ICFn Flag is set at the same timer clock cycle as TOVn is set when either OCRnA or ICRn is used for defining the TOP value. If one of the interrupts are enabled, the interrupt handler routine can be used for updating the TOP and compare values.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values the unused bits are masked to zero when any of the OCRnx Registers are written.

The procedure for updating ICRn differs from updating OCRnA when used for defining the TOP value. The ICRn Register is not double buffered. This means that if ICRn is changed to a low value when the counter is running with none or a low prescaler value, there is a risk that the new ICRn value written is lower than the current value of TCNTn. The result will then be that the counter will miss the compare match at the TOP value. The counter will then have to count to the MAX value (0xFFFF) and wrap around starting at 0x0000 before the compare match can occur. The OCRnA Register however, is double buffered. This feature allows the OCRnA I/O location to be written anytime. When the OCRnA I/O location is written the value written will be put into the OCRnA Buffer Register. The OCRnA Compare Register will then be updated with the value in the Buffer Register at the next timer clock cycle the TCNTn matches TOP. The update is done at the same timer clock cycle as the TCNTn is cleared and the TOVn Flag is set.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is

actively changed (by changing the TOP value), using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In fast PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three.

Refer to [Table 14-1 on page 131](#), [Table 14-2 on page 132](#), and [Table 14-3 on page 132](#).

The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn, and clearing (or setting) the OCnx Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N \cdot (1 + TOP)}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the fast PWM mode. If the OCRnx is set equal to BOTTOM (0x0000) the output will be a narrow spike for each TOP+1 timer clock cycle. Setting the OCRnx equal to TOP will result in a constant high or low output (depending on the polarity of the output set by the COMnx1:0 bits).

A frequency (with 50% duty cycle) waveform output in fast PWM mode can be achieved by setting OCnA to toggle its logical level on each compare match (COMnA1:0 = 1). This applies only if OCR1A is used to define the TOP value (WGM13:0 = 15). The waveform generated will have a maximum frequency of $f_{OCnA} = f_{clk_I/O}/2$ when OCRnA is set to zero (0x0000). This feature is similar to the OCnA toggle in CTC mode, except the double buffer feature of the Output Compare unit is enabled in the fast PWM mode.

14.8.4 Phase Correct PWM Mode

The *phase correct Pulse Width Modulation* or phase correct PWM mode (WGMn3:0 = 1, 2, 3, 10, or 11) provides a high resolution phase correct PWM waveform generation option. The phase correct PWM mode is, like the phase and frequency correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while upcounting, and set on the compare match while downcounting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

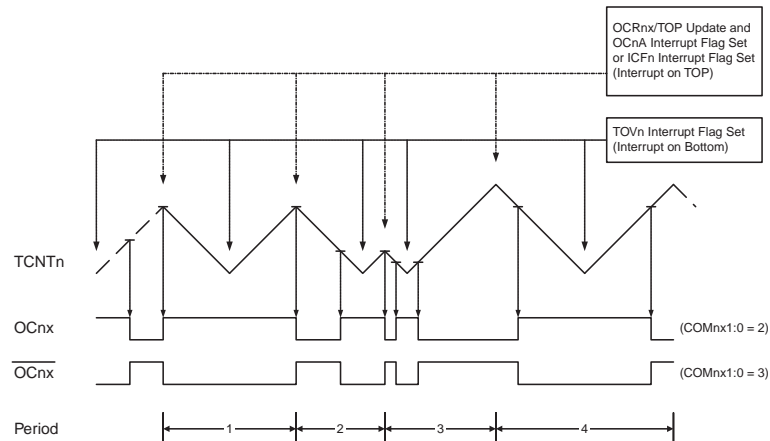
The PWM resolution for the phase correct PWM mode can be fixed to 8-, 9-, or 10-bit, or defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated by using the following equation:

$$R_{PCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase correct PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 1, 2, or 3), the value in ICRn (WGMn3:0 = 10), or the value in OCRnA (WGMn3:0 = 11). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct PWM mode is shown on [Figure 14-8 on page 126](#). The figure shows phase correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope

operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx Interrupt Flag will be set when a compare match occurs.

Figure 14-8. Phase Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOVn) is set each time the counter reaches BOTTOM. When either OCRnA or ICRn is used for defining the TOP value, the OCnA or ICFn Flag is set accordingly at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at TOP). The Interrupt Flags can be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx. Note that when using fixed TOP values, the unused bits are masked to zero when any of the OCRnx Registers are written. As the third period shown in [Figure 14-8 on page 126](#) illustrates, changing the TOP actively while the Timer/Counter is running in the phase correct mode can result in an unsymmetrical output. The reason for this can be found in the time of update of the OCRnx Register. Since the OCRnx update occurs at TOP, the PWM period starts and ends at TOP. This implies that the length of the falling slope is determined by the previous TOP value, while the length of the rising slope is determined by the new TOP value. When these two values differ the two slopes of the period will differ in length. The difference in length gives the unsymmetrical result on the output.

It is recommended to use the phase and frequency correct mode instead of the phase correct mode when changing the TOP value while the Timer/Counter is running. When using a static TOP value there are practically no differences between the two modes of operation.

In phase correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three.

Refer to [Table 14-1 on page 131](#), [Table 14-2 on page 132](#), and [Table 14-3 on page 132](#)

The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at

compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase correct PWM can be calculated by the following equation:

$$f_{OCRnPCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represent special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be continuously high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 11) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

14.8.5 Phase and Frequency Correct PWM Mode

The Phase and Frequency Correct PWM Mode (PWM4x = 1 and WGM40 = 1) provides a high resolution Phase and Frequency Correct PWM waveform generation option. The Phase and Frequency Correct PWM mode is based on a dual-slope operation. The counter counts repeatedly from BOTTOM to TOP (defined as OCR4C) and then from TOP to BOTTOM. In noninverting Compare Output Mode, and in complimentary Compare Output Mode, the Waveform Output (OCW4x) is cleared on the Compare Match between TCNT4 and OCR4x while upcounting, and set on the Compare Match while down-counting. In inverting Output Compare mode, the operation is inverted. The dual-slope operation has lower maximum operation frequency than single slope operation. However, due to the symmetric feature of the dual-slope PWM modes, these modes are preferred for motor control applications.

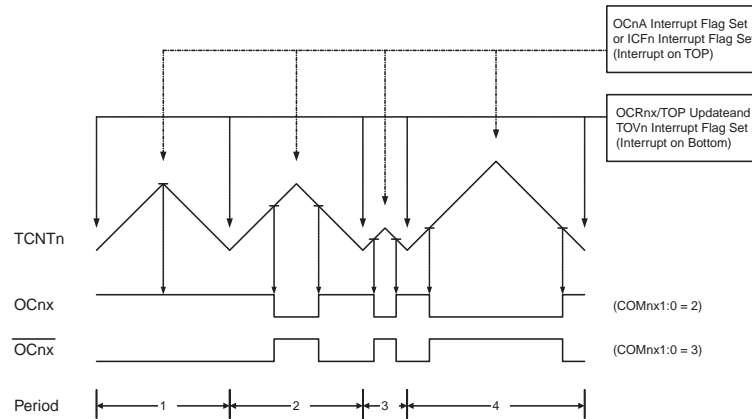
The main difference between the phase correct, and the phase and frequency correct PWM mode is the time the OCRnx Register is updated by the OCRnx Buffer Register, (see [Figure 14-8 on page 126](#) and [Figure 14-9 on page 128](#)).

The PWM resolution for the phase and frequency correct PWM mode can be defined by either ICRn or OCRnA. The minimum resolution allowed is 2-bit (ICRn or OCRnA set to 0x0003), and the maximum resolution is 16-bit (ICRn or OCRnA set to MAX). The PWM resolution in bits can be calculated using the following equation:

$$R_{PFCPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

In phase and frequency correct PWM mode the counter is incremented until the counter value matches either the value in ICRn (WGMn3:0 = 8), or the value in OCRnA (WGMn3:0 = 9). The counter has then reached the TOP and changes the count direction. The TCNTn value will be equal to TOP for one timer clock cycle. The timing diagram for the phase correct and frequency correct PWM mode is shown on [Figure 14-9](#). The figure shows phase and frequency correct PWM mode when OCRnA or ICRn is used to define TOP. The TCNTn value is in the timing diagram shown as a histogram for illustrating the dual-slope operation. The diagram includes non-inverted and inverted PWM outputs. The small horizontal line marks on the TCNTn slopes represent compare matches between OCRnx and TCNTn. The OCnx Interrupt Flag will be set when a compare match occurs.

Figure 14-9. Phase and Frequency Correct PWM Mode, Timing Diagram



The Timer/Counter Overflow Flag (TOVn) is set at the same timer clock cycle as the OCRnx Registers are updated with the double buffer value (at BOTTOM). When either OCRnA or ICFn is used for defining the TOP value, the OCnA or ICFn Flag set when TCNTn has reached TOP. The Interrupt Flags can then be used to generate an interrupt each time the counter reaches the TOP or BOTTOM value.

When changing the TOP value the program must ensure that the new TOP value is higher or equal to the value of all of the Compare Registers. If the TOP value is lower than any of the Compare Registers, a compare match will never occur between the TCNTn and the OCRnx.

As Figure 14-9 on page 128 shows the output generated is, in contrast to the phase correct mode, symmetrical in all periods. Since the OCRnx Registers are updated at BOTTOM, the length of the rising and the falling slopes will always be equal. This gives symmetrical output pulses and is therefore frequency correct.

Using the ICFn Register for defining TOP works well when using fixed TOP values. By using ICFn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

In phase and frequency correct PWM mode, the compare units allow generation of PWM waveforms on the OCnx pins. Setting the COMnx1:0 bits to two will produce a non-inverted PWM and an inverted PWM output can be generated by setting the COMnx1:0 to three.

Refer to Table 14-1 on page 131, Table 14-2 on page 132, and Table 14-3 on page 132.

The actual OCnx value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OCnx). The PWM waveform is generated by setting (or clearing) the OCnx Register at the compare match between OCRnx and TCNTn when the counter increments, and clearing (or setting) the OCnx Register at compare match between OCRnx and TCNTn when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

$$f_{OCnxPFCPWM} = \frac{f_{clk_I/O}}{2 \cdot N \cdot TOP}$$

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the

output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.

14.9 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{Tn}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set, and when the OCRnx Register is updated with the OCRnx buffer value (only for modes utilizing double buffering). Figure 14-10 shows a timing diagram for the setting of OCFnx.

Figure 14-10. Timer/Counter Timing Diagram, Setting of OCFnx, no Prescaling

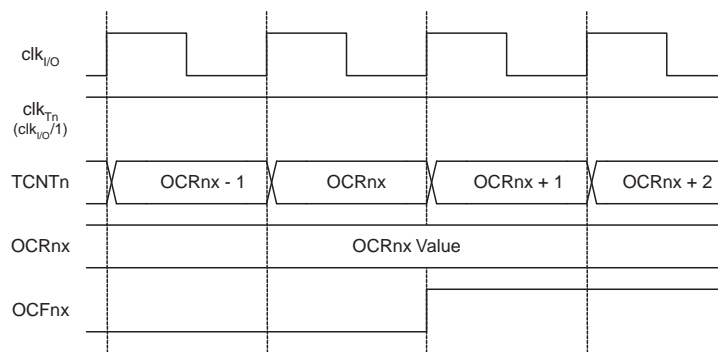


Figure 14-11 shows the same timing data, but with the prescaler enabled.

Figure 14-11. Timer/Counter Timing Diagram, Setting of OCFnx, with Prescaler ($f_{clk_{IO}}/8$)

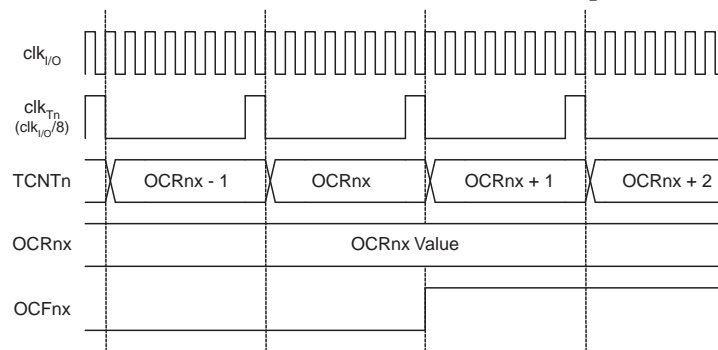


Figure 14-12 shows the count sequence close to TOP in various modes. When using phase and frequency correct PWM mode the OCRnx Register is updated at BOTTOM. The timing diagrams will be the same, but TOP should be replaced by BOTTOM, TOP-1 by BOTTOM+1 and so on. The same renaming applies for modes that set the TOVn Flag at BOTTOM.



Figure 14-12. Timer/Counter Timing Diagram, no Prescaling

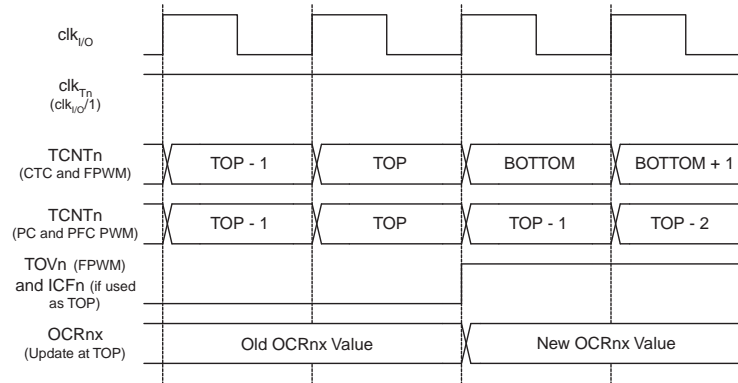
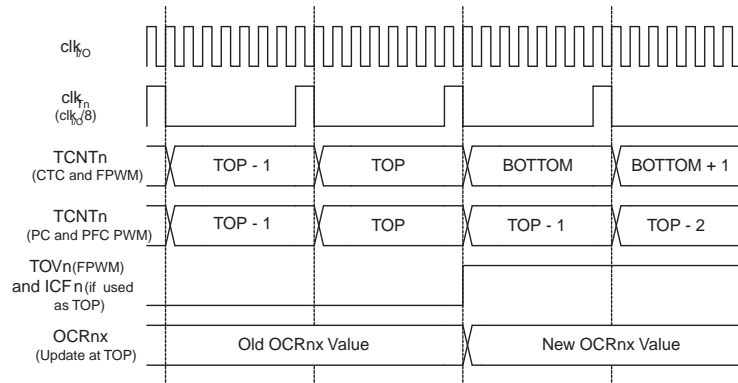


Figure 14-13 shows the same timing data, but with the prescaler enabled.

Figure 14-13. Timer/Counter Timing Diagram, with Prescaler ($f_{clk_IO}/8$)



14.10 16-bit Timer/Counter Register Description

14.10.1 Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.10.2 Timer/Counter3 Control Register A – TCCR3A

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

The COMnA1:0, COMnB1:0, and COMnC1:0 control the output compare pins (OCnA, OCnB, and OCnC respectively) behavior. If one or both of the COMnA1:0 bits are written to one, the OCnA output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnB1:0 bits are written to one, the OCnB output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COMnC1:0 bits are written to one, the OCnC output overrides the normal port functionality of the I/O pin it is connected to. However, note that the Data Direction Register (DDR) bit corresponding to the OCnA, OCnB or OCnC pin must be set in order to enable the output driver.

When the OCnA, OCnB or OCnC is connected to the pin, the function of the COMnx1:0 bits is dependent of the WGMn3:0 bits setting. The table shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to a normal or a CTC mode (non-PWM).

Table 14-1. Compare Output Mode, non-PWM

COMnA1/COMnB1/COMnC1	COMnA0/COMnB0/COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

The table shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the fast PWM mode.

Table 14-2. Compare Output Mode, Fast PWM⁽¹⁾

COMnA1/COMnB1/COMnC0	COMnA0/COMnB0/COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at TOP
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at TOP

Note: 1. A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See [“Fast PWM Mode” on page 100](#) for more details.

The table shows the COMnx1:0 bit functionality when the WGMn3:0 bits are set to the phase correct and frequency correct PWM mode.

Table 14-3. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM⁽¹⁾

COMnA1/COMnB1/COMnC1	COMnA0/COMnB0/COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGM13:0 = 8, 9, 10, or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when down-counting.
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when down-counting.

Note: 1. A special case occurs when OCRnA/OCRnB/OCRnC equals TOP and COMnA1/COMnB1/COMnC1 is set. See [“Phase Correct PWM Mode” on page 101](#) for more details.

• **Bit 1:0 – WGMn1:0: Waveform Generation Mode**

Combined with the WGMn3:2 bits found in the TCCRnB Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used, see the table below. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes ([“Modes of Operation” on page 98](#)).

Table 14-4. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	TOP	TOP
15	1	1	1	1	Fast PWM	OCRnA	TOP	TOP

Note: 1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

14.10.3 Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.10.4 Timer/Counter3 Control Register B – TCCR3B

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNCn: Input Capture Noise Canceler**

Setting this bit (to one) activates the Input Capture Noise Canceler. When the Noise Canceler is activated, the input from the Input Capture Pin (ICPn) is filtered. The filter function requires four successive equal valued samples of the ICPn pin for changing its output. The input capture is therefore delayed by four Oscillator cycles when the noise canceler is enabled.

- **Bit 6 – ICESn: Input Capture Edge Select**

This bit selects which edge on the Input Capture Pin (ICPn) that is used to trigger a capture event. When the ICESn bit is written to zero, a falling (negative) edge is used as trigger, and when the ICESn bit is written to one, a rising (positive) edge will trigger the capture.

When a capture is triggered according to the ICESn setting, the counter value is copied into the Input Capture Register (ICRn). The event will also set the Input Capture Flag (ICFn), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled.

When the ICRn is used as TOP value (see description of the WGMn3:0 bits located in the TCCRnA and the TCCRnB Register), the ICPn is disconnected and consequently the input capture function is disabled.

- **Bit 5 – Reserved Bit**

This bit is reserved for future use. For ensuring compatibility with future devices, this bit must be written to zero when TCCRnB is written.

- **Bit 4:3 – WGMn3:2: Waveform Generation Mode**

See TCCRnA Register description.

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see [Figure 13-8 on page 102](#) and [Figure 13-9 on page 103](#).

Table 14-5. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	clk _{IO} /1 (No prescaling)
0	1	0	clk _{IO} /8 (From prescaler)
0	1	1	clk _{IO} /64 (From prescaler)
1	0	0	clk _{IO} /256 (From prescaler)
1	0	1	clk _{IO} /1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Counter, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

14.10.5 Timer/Counter1 Control Register C – TCCR1C

Bit	7	6	5	4	3	2	1	0	
	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

14.10.6 Timer/Counter3 Control Register C – TCCR3C

Bit	7	6	5	4	3	2	1	0	
	FOC3A	–	–	–	–	–	–	–	TCCR3C
Read/Write	W	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – FOCnA: Force Output Compare for Channel A**

The FOCnA/FOCnB/FOCnC bits are only active when the WGMn3:0 bits specifies a non-PWM mode. When writing a logical one to the FOCnA/FOCnB/FOCnC bit, an immediate compare match is forced on the waveform generation unit. The OCnA/OCnB/OCnC output is changed according to its COMnx1:0 bits setting. Note that the FOCnA/FOCnB/FOCnC bits are implemented as strobes. Therefore it is the value present in the COMnx1:0 bits that determine the effect of the forced compare.

A FOCnA/FOCnB/FOCnC strobe will not generate any interrupt nor will it clear the timer in Clear Timer on Compare Match (CTC) mode using OCRnA as TOP.

The FOCnA/FOCnB/FOCnC bits are always read as zero.

- **Bit 4:0 – Reserved Bits**

These bits are reserved for future use. For ensuring compatibility with future devices, these bits must be written to zero when TCCRnC is written.

14.10.7 Timer/Counter1 – TCNT1H and TCNT1L

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.10.8 Timer/Counter3 – TCNT3H and TCNT3L

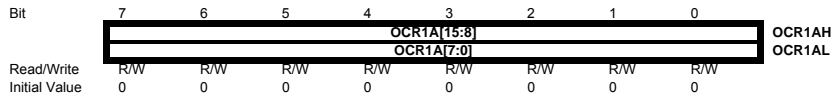
Bit	7	6	5	4	3	2	1	0	
	TCNT3[15:8]								TCNT3H
	TCNT3[7:0]								TCNT3L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The two *Timer/Counter* I/O locations (TCNTnH and TCNTnL, combined TCNTn) give direct access, both for read and for write operations, to the *Timer/Counter* unit 16-bit counter. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “[Accessing 16-bit Registers](#)” on page 113.

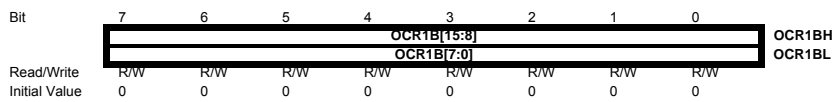
Modifying the counter (TCNTn) while the counter is running introduces a risk of missing a compare match between TCNTn and one of the OCRnx Registers.

Writing to the TCNTn Register blocks (removes) the compare match on the following timer clock for all compare units.

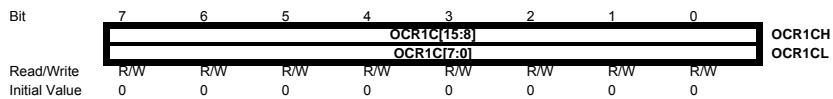
14.10.9 Output Compare Register 1 A – OCR1AH and OCR1AL



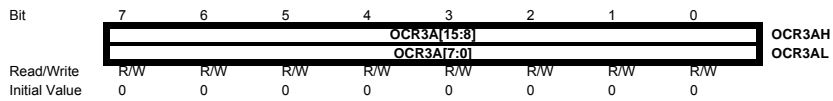
14.10.10 Output Compare Register 1 B – OCR1BH and OCR1BL



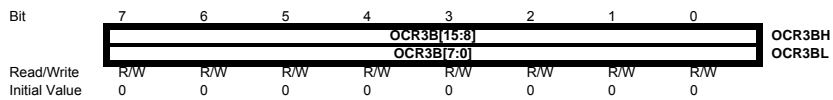
14.10.11 Output Compare Register 1 C – OCR1CH and OCR1CL



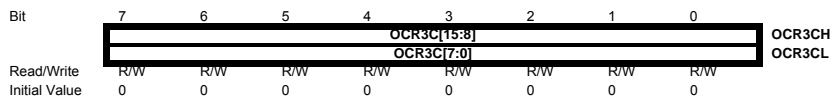
14.10.12 Output Compare Register 3 A – OCR3AH and OCR3AL



14.10.13 Output Compare Register 3 B – OCR3BH and OCR3BL



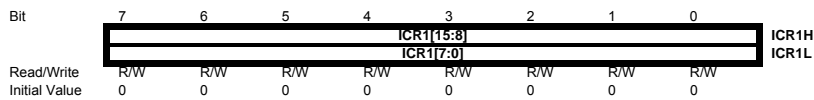
14.10.14 Output Compare Register 3 C – OCR3CH and OCR3CL



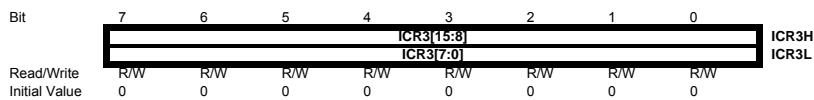
The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNTn). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OCnx pin.

The Output Compare Registers are 16-bit in size. To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 113.

14.10.15 Input Capture Register 1 – ICR1H and ICR1L



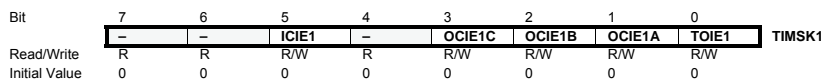
14.10.16 Input Capture Register 3 – ICR3H and ICR3L



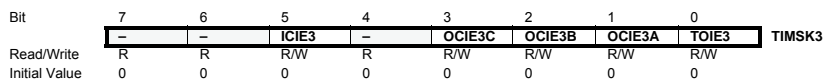
The Input Capture is updated with the counter (TCNTn) value each time an event occurs on the ICPn pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

The Input Capture Register is 16-bit in size. To ensure that both the high and low bytes are read simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers. See “Accessing 16-bit Registers” on page 113.

14.10.17 Timer/Counter1 Interrupt Mask Register – TIMSK1



14.10.18 Timer/Counter3 Interrupt Mask Register – TIMSK3



- **Bit 5 – ICIE_n: Timer/Counter_n, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter_n Input Capture interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 63.) is executed when the ICF_n Flag, located in TIFR_n, is set.

- **Bit 3 – OCIEnC: Timer/Counter_n, Output Compare C Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter_n Output Compare C Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 63.) is executed when the OCF_nC Flag, located in TIFR_n, is set.

- **Bit 2 – OCIE_nB: Timer/Counter_n, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter_n Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 63.) is executed when the OCF_nB Flag, located in TIFR_n, is set.

- **Bit 1 – OCIE_nA: Timer/Counter_n, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter_n Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 63.) is executed when the OCF_nA Flag, located in TIFR_n, is set.

- **Bit 0 – TOIE_n: Timer/Counter_n, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter_n Overflow interrupt is enabled. The corresponding Interrupt Vector (See “Interrupts” on page 63.) is executed when the TOV_n Flag, located in TIFR_n, is set.

14.10.19 Timer/Counter1 Interrupt Flag Register – TIFR1

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.10.20 Timer/Counter3 Interrupt Flag Register – TIFR3

Bit	7	6	5	4	3	2	1	0	
	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3	TIFR3
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICF_n: Timer/Counter_n, Input Capture Flag**

This flag is set when a capture event occurs on the ICP_n pin. When the Input Capture Register (ICR_n) is set by the WGM_n3:0 to be used as the TOP value, the ICF_n flag is set when the counter reaches the TOP value.


ICF_n is automatically cleared when the Input Capture Interrupt Vector is executed. Alternatively, ICF_n can be cleared by writing a logic one to its bit location.

- **Bit 3 – OCF_nC: Timer/Counter_n, Output Compare C Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT_n) value matches the Output Compare Register C (OCR_nC).

Note that a Forced Output Compare (FOC_nC) strobe will not set the OCF_nC Flag.

OCF_nC is automatically cleared when the Output Compare Match C Interrupt Vector is executed. Alternatively, OCF_nC can be cleared by writing a logic one to its bit location.



- **Bit 2 – OCFnB: Timer/Counter1, Output Compare B Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register B (OCRnB).

Note that a Forced Output Compare (FOCnB) strobe will not set the OCFnB Flag.

OCFnB is automatically cleared when the Output Compare Match B Interrupt Vector is executed. Alternatively, OCFnB can be cleared by writing a logic one to its bit location.

- **Bit 1 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNTn) value matches the Output Compare Register A (OCRnA).

Note that a Forced Output Compare (FOCnA) strobe will not set the OCFnA Flag.

OCFnA is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCFnA can be cleared by writing a logic one to its bit location.

- **Bit 0 – TOVn: Timer/Counter, Overflow Flag**

The setting of this flag is dependent of the WGMn3:0 bits setting. In Normal and CTC modes, the TOVn Flag is set when the timer overflows. Refer to [Table 14-4 on page 133](#) for the TOVn Flag behavior when using another WGMn3:0 bit setting.

TOVn is automatically cleared when the Timer/Counter Overflow Interrupt Vector is executed. Alternatively, TOVn can be cleared by writing a logic one to its bit location.