TRABAJO FIN DE GRADO

INEGENIERÍA ELECTRÓNICA INDUSTRIAL AUTOMÁTICA

SISTEMA ODOMÉTRICO BASADO EN EFECTO DOPPLER

ODOMETRIC SYSTEM BASED ON DOPPLER EFFECT

AUTOR: ALEXIS RODRÍGUEZ JORDÁN
TUTOR: JONAY TOMÁS TOLEDO CARRILLO

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA
LA LAGUNA, JULIO 2021

Agradecimientos

Quiero agradecer en primer lugar a mi tutor, Jonay Toledo, que me ha dado la idea y la oportunidad para la realización de este proyecto, y me ha estado guiando a lo largo de las distintas etapas, ayudándome en los momentos en los que surgían dudas y problemas.

También quiero agradecer al profesorado del grado de ingeniería electrónica que me ha enseñado los conocimientos de las distintas disciplinas, las cuales me han servido y he aplicado en gran parte de este trabajo, y que me servirán para la ejecución de futuros trabajos.

Por otro lado, quiero agradecer a mis padres y hermanas, que siempre han estado apoyándome a lo largo de estos 4 años de estudio, animándome en los momentos más difíciles, donde aplicar un esfuerzo extra te llevará a una recompensa mayor y a cumplir con tus objetivos. Es gracias a ellos, por los que he podido optar a estudiar este grado y llegar hasta aquí.

Por último, no me puedo olvidar de mis compañeros y a mi grupo de amigos, con los cuales he compartido los momentos más difíciles, pero también los mejores momentos acompañados por los logros de nuestros esfuerzos. Me han ayudado a sacar adelante las asignaturas con su colaboración y el esfuerzo en grupo en prácticas y trabajos, donde compartíamos nuestras ideas.

En general, todos han colaborado para que hoy pueda graduarme y salir como ingeniero técnico industrial de la rama de electrónica y automática.

Resumen

Los sistemas robotizados han sido una de las ramas de estudio con mayor interés dentro del marco de la tecnología. Estos sistemas robotizados se llevan implementando desde hace tiempo, con robots manipuladores en los procesos automáticos de las plantas industriales, pero este tipo de robots no son capaces de operar fuera de su entorno y es aquí donde entran los robots móviles.

La idea de un robot móvil es que pueda desplazarse en su entorno a través de un medio, ya sea aéreo, terrestre o marítimo. Todos estos robots tienen la necesidad de conocer su entorno para poder actuar ante sus diversidades. Esta cantidad de información es suministrada a través de los sensores que convierten alguna magnitud física en impulsos eléctricos. La capacidad que tiene un robot de actuar depende por lo tanto no solo de sus actuadores, sino también de la precisión con la que obtiene las magnitudes que rigen su sistema. Es en este punto cuando cobra el sentido de la elaboración de este proyecto.

Concretamente estamos ante un problema clásico de la odometría de los robots móviles terrestres, que consiste en la determinación de la posición. La posición relativa de un robot se puede determinar si se conoce las velocidades de su sistema de referencia en base a su posición inicial. La idea es aumentar la precisión con la que se obtienen estas velocidades, reduciendo la incertidumbre mediante la implementación de un sensor que pueda actuar en aquellos casos en los que otros tipos de sensores fallan.

En los robots terrestre, una forma típica de obtener la velocidad es con un *encoder*, un sensor giratorio que mide la velocidad de las ruedas. Este sensor, aunque posee bastante resolución atrae varios problemas como la introducción de datos erróneos al medir en situaciones en el que las ruedas del robot sufren algún deslizamiento.

La forma que hemos elegido para resolver este problema es diseñando e implementando un sensor de ultrasonidos, que aprovechándose del efecto *Doppler*, será capaz de medir la velocidad sin necesidad de tener un contacto físico con el suelo.

Para ello, hemos organizado este proyecto en etapas a distinto niveles. Primero la fase de documentación, donde se establecerá todo el marco teórico y que será necesario para entender y diseñar el sensor. Después, la implementación práctica del proyecto, donde se estudiarán las características del sensor de ultrasonido utilizado, para encontrar la mejor forma de aprovechar el efecto *Doppler,* diseñar el circuito y programar el microcontrolador Arduino que calculará la velocidad y gestionará la comunicación con el robot. Luego, tenemos la fase de medida y comprobación de la eficacia del sensor y de los métodos usados. Nos apoyaremos en el uso de simuladores, instrumentos de medida, un banco de ensayos donde poder someter al sensor a distintas velocidades, e incluso se ha realizado la captura de datos con otro microcontrolador, el STM32, que nos añadirá información extra, con la aplicación de varios programas de análisis de datos como Python. La ordenación de estas fases serán acorde a él orden llevado en la ejecución de este proyecto, en el tiempo en el que fue realizado.

Tras analizar los resultados y en la fase final del proyecto, expondremos nuestra conclusión sobre la eficacia del sensor, viendo la viabilidad de utilizarlo en un robot real de interior con su instalación en una silla de ruedas motorizada, y exponiendo sus ventajas, desventajas y posibles acciones de mejora.

Abstract

Robotic systems have been one of the branches of study with the greatest interest within the framework of technology. These robotic systems have been implemented for a long time, with manipulator robots in the automatic processes of industrial plants, but these types of robots are not capable of operating outside their environment, and this is where mobile robots come in.

The idea of a mobile robot is that it can move in its environment by means of a medium, be it air, land, or sea. All these robots have the need to know their environment to be able to act on its diversities. This amount of information is provided through sensors that convert some physical magnitude into electrical impulses. The ability of a robot to act therefore depends not only on its actuators, but also on the precision with which it obtains the magnitudes that govern its system. It is at this point that this project makes sense.

Specifically, we are dealing with a classic problem in the odometry of terrestrial mobile robots, which consists of determining their position. The relative position of a robot can be determined if the velocities of its reference system are known based on its initial position. The idea is to increase the accuracy with which these velocities are obtained, reducing uncertainty by implementing a sensor that can act in cases where other types of sensors fail.

In terrestrial robots, a typical way to obtain the speed is with an encoder, a rotating sensor that measures the speed of the wheels. This sensor, although it has a high resolution, attracts several problems such as the introduction of erroneous data when measuring in situations where the wheels of the robot suffer some slippage.

The way we have chosen to solve this problem is by designing and implementing an ultrasonic sensor, which, taking advantage of the Doppler effect, will be able to measure the speed without the need for physical contact with the ground.

To do this, we have organized this project in stages at different levels. First, the documentation phase, where all the theoretical frameworks will be established, and which will be necessary to understand and design the sensor. Then, the practical implementation of the project, where we will study the characteristics of the ultrasound sensor used, to find the best way to take advantage of the Doppler effect, design the circuit and program the Arduino microcontroller that will calculate the speed and manage the communication with the robot. Then we have the phase of measuring and testing the efficiency of the sensor and the methods used. We will rely on the use of simulators, measuring instruments, a test bench where the sensor can be subjected to different speeds, and we have even captured data with another microcontroller, the STM32, which will add extra information, with the application of various data analysis programs such as Python. The order of these phases will be in accordance with the order in which this project was carried out during the time in which it was carried out.

After analysing the results and in the final phase of the project, we will present our conclusion on the effectiveness of the sensor, looking at the feasibility of using it in a real indoor robot with its installation in a motorised wheelchair, and explaining its advantages, disadvantages, and possible actions for improvement.

Índice

1. Introducción

Una de las tecnologías que más impacto ha tenido en los últimos años, ha sido la automatización y la robótica. La robótica es la encargada del diseño y construcción de robots, en la que intervienen multitud de disciplinas como la electrónica, la mecánica y la informática.

La robótica ha logrado grandes avances tecnológicos que han logrado mejorar el estilo de vida de las personas, por ejemplo, la incorporación de robots a la industria ha logrado producir en mayores cantidades y con menores costos, pudiendo llevar más tecnología a más partes del mundo.

Pero primero abarcaremos sobre que es un robot. Un robot es una máquina que ha sido programada para realizar acciones que antes solo podían ser realizadas por humano. Existen multitud de robots en función del tipo de constitución y aplicación, por ejemplo, tenemos los robots de manipulación como los brazos robóticos, usados en la industria. También tenemos robots móviles que pueden realizar multitud de acciones como exploración, transporte, manipulación… Los robots móviles han sido un foco importante de investigación por la capacidad que tienen de poder trabajar en diferentes entornos, sin embargo, su modelización es mucho más compleja, pues no existe un punto de referencia fija.

Los robots responden a su entorno mediante el manejo de los datos proporcionados por los sensores que posee. Según la clase de entorno serán necesario ciertos tipos de sensores para medir distintas magnitudes físicas que permitan calcular la posición y orientación.

1.1 Antecedentes

Este proyecto nace de las inquietudes que existen en la determinación de la posición de un robot móvil terrestre, bajo ciertas condiciones.

El tipo de robot al que será enfocado este proyecto es un robot compuesto por dos ruedas móviles. Existen multitud de configuraciones en base al número de ruedas motrices y tipo de dirección (Figura 1.1), en este caso trataremos un robot móvil de tracción diferencial.

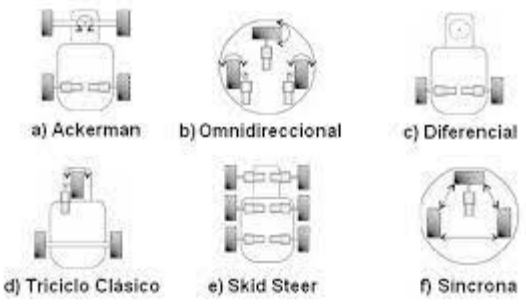


Figura 1.1. Configuraciones robot móvil [21]

La tracción diferencial debe su nombre precisamente porque el control de la velocidad y dirección es gobernado por la diferencia de velocidades de las dos ruedas motrices [1]. Esta configuración permite realizar giros cerrados incluso sobre su propio eje cuando las ruedas giran en sentido opuesto.

El robot móvil que trataremos en el proyecto será una silla de ruedas motorizada con el tipo de dirección descrita, la cual posee multitud de sensores y dispone de un ordenador con la

suficiente capacidad, como para permitir realizar distintas funciones como el control manual de forma remota e incluso conducción autónoma.

Es por lo tanto importante poder determinar la posición con la menor incertidumbre posible. El método más utilizado es medir la velocidad a la que se mueve cada rueda y luego aplicar un modelo de cinemática directa. Al estudio y estimación de la posición de vehículos a ruedas, es lo que se denomina odometría.

La velocidad se obtiene haciendo uso de un sensor, típicamente mediante un *encoder,* que mide el número de revoluciones de las ruedas. Los *encoders* tienen bastante resolución sin embargo su implementación conlleva unos cuantos inconvenientes:

- Los *encoders* requieren de una conexión mecánica a la rueda, lo que implica tener que realizar una instalación adecuada y sin holguras.
- La velocidad calculada dependerá de otros parámetros como el radio de la rueda o las características del *encoder*, por lo que no puede ser cambiado de un dispositivo a otro de forma directa.
- Cuando una rueda está en movimiento no implica siempre que se produzca un cambio de posición u orientación. En los casos en los que la resistividad del terreno es baja o el robot se encuentre bloqueado por un objeto, la rueda deslizará sobre la superficie e introducirá un error.

Este último caso es el que nos interesa, pues un deslizamiento aumentará la incertidumbre de la muestra y no solo eso, si no que el robot no será consciente de ello.

Para minimizar esta incertidumbre, lo que se lleva haciendo en los últimos años es disponer de otros sensores que puedan proporcionar más datos. Sin embargo, los métodos usados requieren en ocasiones modelos más complejos, y una inversión económica más costosa.

1.2 Objetivo

El objetivo de este proyecto en encontrar y plantear una solución al problema anterior descrito. Para ello se implementará un sensor que sea capaz de medir la velocidad de las ruedas aprovechándose de un efecto físico de las ondas, el efecto *Doppler*.

Se realizará un montaje experimental con un sensor de ultrasonidos, esto permitirá medir la velocidad del robot sin necesidad de tener un contacto físico con el suelo, y a además solo medirá si existe de verdad un desplazamiento.

2. Marco Teórico.

Pasamos a detallar el marco teórico en el cual se apoya este trabajo. En primer lugar, explicaremos como el efecto *Doppler* nos permite obtener la velocidad, y posteriormente como a partir de la velocidad obtenida calcular la posición aplicando un modelo de cinemática directa (Odometría).

2.1 Odometría.

Es posible calcular la posición relativa del robot, si se conoce o establece un punto inicial y si se conocen las velocidades del robot. La odometría es la que se encarga de este aspecto, en los vehículos con ruedas.

Para obtener la posición actual, se calcula mediante un modelo de cinemática directa, y obtendremos un sistema con tres grados de libertad, es decir, dos para la posición en el plano y una para la orientación (Figura 2.1.1). Estas coordenadas serán relativas respecto a un punto de partida.

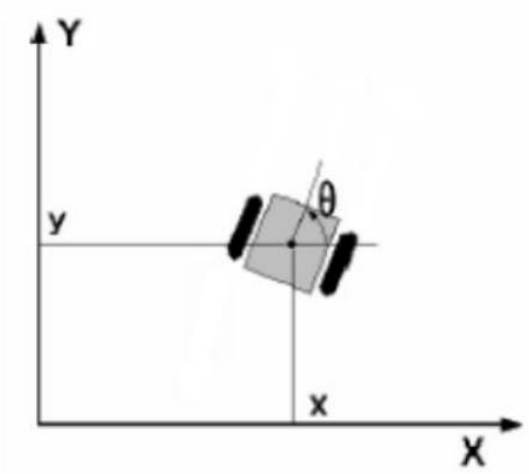


Figura 2.1.1. Sistema de coordenadas

Para obtener los valores de coordenadas X e Y, y el ángulo θ, se aplica un estudio abordando el problema como cinemática diferencial, donde existen dos ruedas alineadas en el mismo eje, pero donde cada una puede tener una velocidad de rotación y dirección diferentes.

La velocidad lineal se puede obtener estudiando el siguiente modelo (Figura 2.1.2).

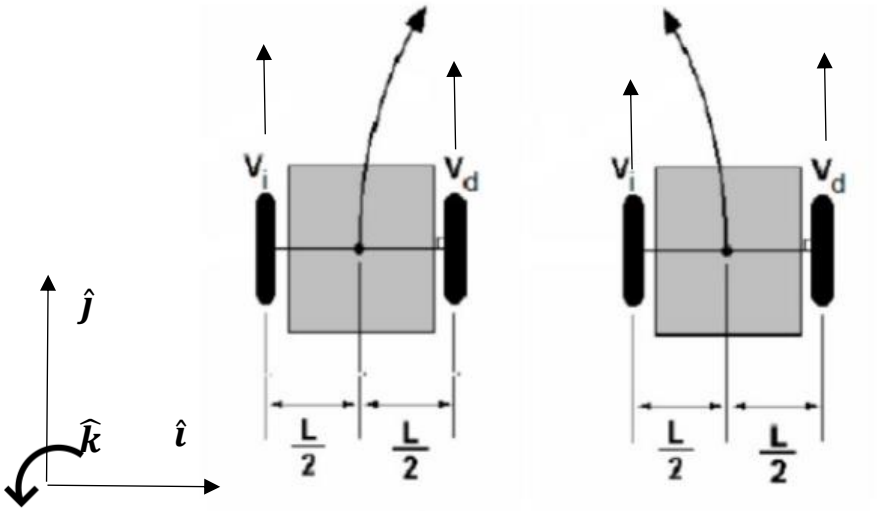


Figura 2.1.2. Modelo de vehículo diferencial

Estamos aplicando un sistema de referencia dextrógiro, en sentido contrario a las agujas del reloj.

La expresión general de la velocidad de un sólido rígido es la siguiente [2]:

$$V_P^{\rightarrow} = V_0^{\rightarrow} + \omega^{\rightarrow} \wedge OP^{\rightarrow}$$

Si ponemos Vd como punto de partida podemos obtener tanto la velocidad lineal, como la velocidad angular:

$$Vi^{\rightarrow} = Vd^{\rightarrow} + \omega^{\rightarrow} \wedge L^{\rightarrow}$$

$$Vi \cdot \hat{\jmath} = Vd \cdot \hat{\jmath} + \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ 0 & 0 & \omega \\ -L & 0 & 0 \end{vmatrix} = Vd \cdot \hat{\jmath} - \begin{vmatrix} 0 & \omega \\ -L & 0 \end{vmatrix} \cdot \hat{\jmath} \rightarrow$$

$$Vi = Vd - \omega \cdot L \rightarrow \ \omega = \frac{Vd - Vi}{L} \left(\frac{rad}{s}\right)$$

Con la velocidad angular se puede obtener la velocidad lineal del vehículo, en su eje de simetría.

$$V_{Centro}^{\rightarrow} = Vd^{\rightarrow} \cdot \omega^{\rightarrow} \wedge \frac{L^{\rightarrow}}{2}$$

$$V_{Centro} \cdot \hat{\jmath} = Vd \cdot \hat{\jmath} + \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ 0 & 0 & \dfrac{Vd - Vi}{L} \\ -\dfrac{L}{2} & 0 & 0 \end{vmatrix} = Vd \cdot \hat{\jmath} - \begin{vmatrix} 0 & \dfrac{Vd - Vi}{L} \\ -\dfrac{L}{2} & 0 \end{vmatrix} \cdot \hat{\jmath}$$

$$V_{Centro} = Vd - \frac{Vd - Vi}{\cancel{L}} \cdot \frac{\cancel{L}}{2} \rightarrow \ V_{centro} = \frac{Vd + Vi}{2} \left(\frac{m}{s}\right)$$

Las velocidades de las ruedas se obtendrán directamente con el sensor que implementaremos, pero en otros casos como con los *encoders* se obtiene con la velocidad angular de la rueda y el radio.

$$V_{rueda} = \omega_{rueda} \cdot 2\pi \cdot r$$

Una vez obtenidas las velocidades se puede obtener la posición y orientación con la iteración de los datos obtenidos en cada instante de tiempo distanciados en un Δt. Se calcula la variación de la posición en cada instante de tiempo y se suma a la posición del instante anterior:

$$x(k+1) = x(k) + V(k) \cdot \cos\big(\theta(k)\big) \cdot \Delta t \quad (1)$$

$$y(k+1) = y(k) + V(k) \cdot \text{sen}\big(\theta(k)\big) \cdot \Delta t \quad (2)$$

De la misma forma se puede obtener la variación del ángulo:

$$\theta(k+1) = \theta(k) + \omega(k) \cdot \Delta t \quad (3)$$

2.2 Efecto Doppler.

El efecto *Doppler* es fenómeno por el cual la frecuencia percibida de una onda cambia, cuando el emisor o foco de ondas, posee un movimiento relativo con el receptor [3].

Para entender esto mejor, debemos saber primero que la velocidad de una onda y su frecuencia están relacionadas por la siguiente ecuación:

$$V = \lambda \cdot f \quad (4)$$

Donde:

- f es la frecuencia de la onda medida en Hercios.
- λ es la longitud de la onda en metros. Expresa la distancia física entre dos puntos a partir de los cuales la onda se repite.
- V es la velocidad de propagación de la onda en el medio.

La velocidad de propagación de una onda depende del tipo de onda (mecánica, electromagnética), y del medio en el que se propaga. Por lo tanto, si el medio no cambia, la relación entre la frecuencia y la longitud de onda son inversamente proporcionales.

Normalmente el efecto *Doppler* se suele atribuir a las ondas de sonidos, pero se puede producir en otro tipo de ondas como las electromagnéticas.

Pues bien, vamos a suponer un caso en el caso en el que tenemos un emisor de ondas de sonido de frecuencia constante y un receptor en reposo. En primer lugar, si el emisor se encuentra en reposo, las ondas se propagarán y cada una tendrá una longitud de onda determinada según la expresión anterior. Al receptor le llegara una onda con la misma longitud de onda y por lo tanto con la misma frecuencia.

Ahora vamos a poner el emisor en movimiento y veamos qué pasa si se aleja o si se acerca al receptor. Cuando el emisor se acerca la longitud de la onda emitida se ve reducida, debido a que en un periodo de onda la distancia respecto al foco emisor ha disminuido. Esto se puede observar mejor con la siguiente imagen (Figura 2.2.1).

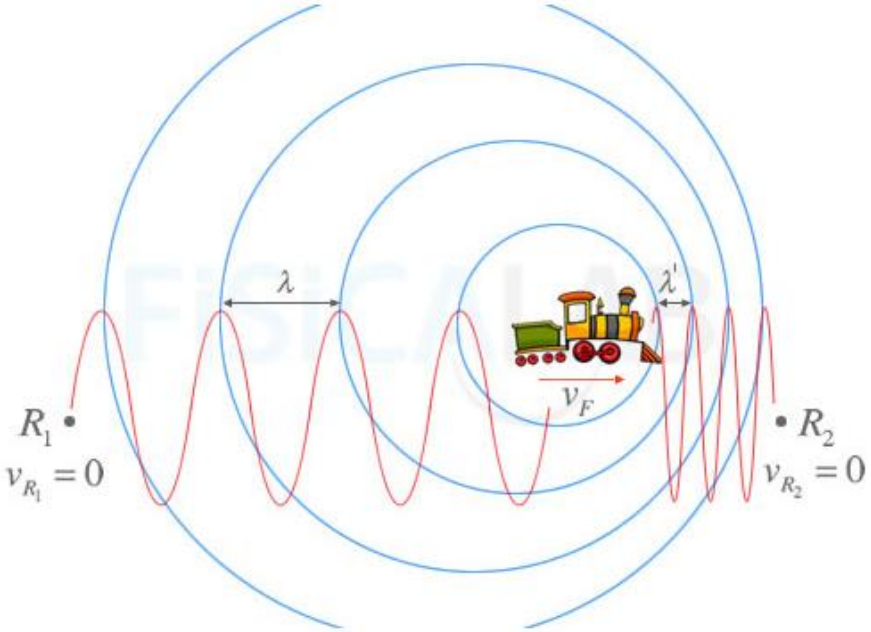


Figura 2.2.1. Efecto Doppler emisor en movimiento y receptor estático [3]

Como se reduce la longitud de onda, la frecuencia debe de aumentar. Podemos obtener la frecuencia del receptor obteniendo la variación de la longitud de la onda, y para eso aplicaremos las ecuaciones de la cinemática del movimiento rectilíneo uniforme (con velocidad constante).

La variación de la posición del emisor en un instante de tiempo determinado se puede obtener como:

$$\Delta x = v_{emisor} \cdot t \quad (5)$$

El tiempo en el que se determina la variación de la posición, corresponde con el periodo de la onda, es decir con la inversa de la frecuencia, por lo tanto:

$$\Delta x = v_{emisor} \cdot T = \frac{v_{emisor}}{f}$$

Ahora bien, como nos estamos acercando esta variación de la posición se resta a la longitud de onda, u obtendremos una nueva longitud de onda que llamaremos λ':

$$\lambda' = \lambda - \Delta x = \lambda - \frac{v_{emisor}}{f}$$

Podemos expresar la longitud de onda como $\lambda = \frac{c}{f}$ donde c será la velocidad de propagación de la onda por lo tanto la expresión final nos quedará de la siguiente forma:

$$\lambda' = \lambda - \frac{v_{emisor}}{f} = \frac{c}{f} - \frac{v_{emisor}}{f} = \frac{c - v_{emisor}}{f}$$

$$\lambda' = \frac{c}{f'} \rightarrow \frac{c}{f'} = \frac{c - v_{emisor}}{f} \rightarrow despejando\ f' \rightarrow$$

$$f' = \frac{c}{c - v_{emisor}} \cdot f$$

Como $c - v_{emisor} < c$, esto implica que la frecuencia percibida f' será mayor que frecuencia emitida.

En el caso de alejarse, ocurre el efecto contrario, la longitud de onda aumenta y por lo tanto la frecuencia percibida es menor. Su expresión es similar a la anterior, pero con el signo cambiado:

$$f' = \frac{c}{c + v_{emisor}} \cdot f$$

Como se puede observar, la frecuencia recibida depende de la velocidad, y esto será lo que aprovechamos calcular la posición. Si emitimos una onda de sonido de frecuencia conocida y medimos la frecuencia recibida, seremos capaces de obtener la velocidad.

Esto será realizado con un sensor de ultrasonidos que dispone de un emisor y receptor.

Lo que permite que la onda sea recibida y emitida en el mismo punto, es el fenómeno de la reflexión (Figura 2.2.2).

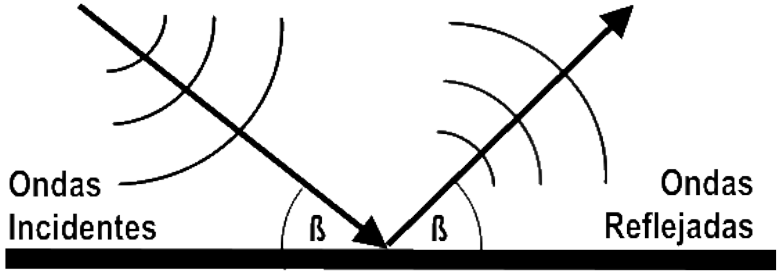


Figura 2.2.2. Reflexión del sonido [4]

"*Fenómeno por el cual una onda acústica es devuelta por una superficie de separación entre dos medios, con un ángulo de reflexión igual al ángulo de incidencia*". [4]

En un principio, por la descripción anterior parece que no se puede colocar el emisor y el receptor en el mismo punto a excepción de tener un ángulo de incidencia perpendicular a la superficie. Sin embargo, en la práctica la mayor parte de las superficies no suelen ser totalmente lisas, y poseen imperfecciones que les dan cierto grado de rugosidad.

Además, la reflexión depende de la frecuencia, es decir, una onda para que pueda ser reflejada debe de incidir sobre un obstáculo de tamaño superior a la longitud de la onda. Implica por lo tanto que, a mayor frecuencia, más fácil será que se refleje la onda en la misma dirección de emisión. [5]

Ahora bien, debido a que la recepción y emisión se hará en el robot en movimiento, lo que se producirá será un efecto *Doppler* doble. Las ondas emitidas modificarán su frecuencia por la velocidad añadida e incidirán sobre el suelo que será un receptor estático. Luego este emite las ondas con la frecuencia modificada, y serán recibidas por el sensor, el cual por su movimiento modificará aún más la frecuencia. También se puede llegar a considerar que tanto el emisor como el receptor son el robot y por lo tanto que ambos están en movimiento. Si el desplazamiento y la emisión son realizadas en el mismo sentido, el incremento de frecuencia será positivo y será negativo en el caso contrario.

El modo de instalación que se usará para el sensor será en la zona delantera del robot con desplazamiento y emisión en el mismo sentido. Esto es importante para la obtención de las ecuaciones y definir el signo.

Podemos hallar la ecuación de la velocidad, aplicando el caso general del efecto *Doppler* [3].

$$f' = f \cdot \frac{c \pm v_{receptor}}{c \pm v_{emisor}} \quad (6)$$

Donde:

- f, es la frecuencia emitida
- f', es la frecuencia observada
- c, es la velocidad de propagación de la onda, 340 m/s para el sonido en el aire.
- $v_{receptor}$, velocidad del receptor
- v_{emisor}, velocidad emisor

Y el signo elegiremos para el numerador el signo positivo (acercamiento) y para el denominador signo negativo (acercamiento). Además, las velocidades del receptor y del emisor son el mismo, por lo tanto:

$$f' = f \cdot \frac{c + v}{c - v}$$

Podemos expresar la frecuencia *Doppler* como la variación de la frecuencia.

$$\Delta f = f' - f = f \cdot \frac{c + v}{c - v} - f \rightarrow$$

$$\Delta f = f \cdot \frac{2 \cdot v}{c - v} \quad (7)$$

Con la expresión anterior ya obtendríamos la frecuencia *Doppler*, pero para simplificarla un poco podemos aproximarla un poco despreciando el termino de v del denominador.

$$Como\ c \gg v \rightarrow$$

$$\Delta f \approx f \cdot \frac{2 \cdot v}{c} \quad (8)$$

La ecuación anterior nos sirve para el caso en el que el vector de velocidad y el de emisión de ondas están en la misma dirección y sentido, pero este no es nuestro caso, véase la figura 2.2.3.

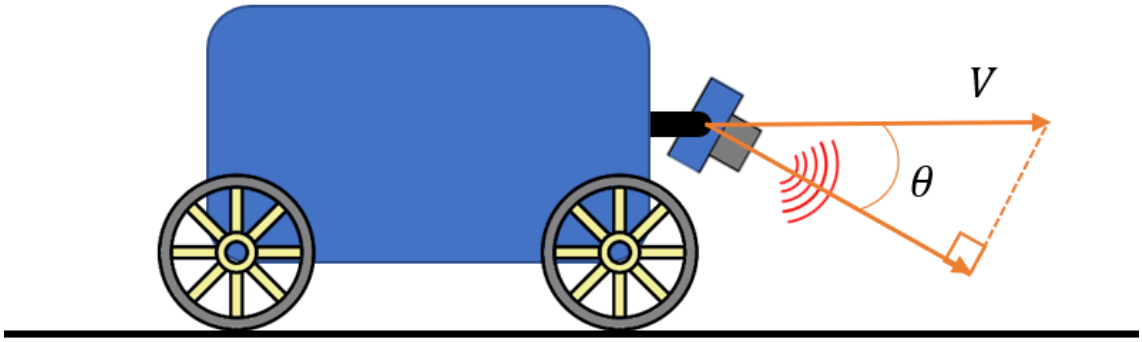


Figura 2.2.3. Modo de instalación del sensor

Como se puede ver en la figura 2.2.3, la velocidad que interviene en el efecto *Doppler* corresponde con la componente de $V \cdot \cos(\theta)$. La última expresión quedaría de la siguiente forma:

$$\Delta f = f \cdot \frac{2 \cdot V \cdot \cos(\theta)}{c} \quad (9)$$

Como nos interesa obtener la velocidad a partir de la diferencia de frecuencia, la expresión final y la que usaremos será:

$$V = \frac{c}{f \cdot 2 \cdot \cos(\theta)} \cdot \Delta f \quad (10)$$

Fíjese que en la expresión anterior es necesario que $\rightarrow 0 < \cos(\theta) < 1$. Un ángulo de $0°$ implicaría estar paralelo al suelo y apenas se reflejarían las ondas, pero si vamos al caso del otro extremo, cerca de los $90°$, cualquier mínima variación en la frecuencia implicaría una gran variación de la velocidad, y esto no nos interesa como veremos más adelante.

2.3 Propagación de ondas en el mismo medio.

"*Cuando dos trenes de ondas de igual amplitud pero frecuencias ligeramente diferentes coinciden en el espacio, dan lugar a una vibración cuya amplitud varía con el tiempo. Si se trata de ondas sonoras, estas variaciones de amplitud se percibirán como variaciones de sonoridad, o lo que es lo mismo, aumentos o disminuciones periódicas de intensidad, que se denominan batidos o pulsaciones*" [6].

Este fenómeno descrito ocurre en nuestro caso con la señal recibida por compartir el mismo medio con la emitida. El tipo de sensor a usar dispone de un emisor y un receptor separados a pocos centímetros, lo que produce que las ondas reflejadas choquen con las ondas emitidas.

Las ondas recibidas serán, por lo tanto, la resta de las reflejadas con las emitidas:

$$F_{recibida}(t) = F_{reflejada}(t) - F_{emitida}(t) \quad (11)$$

Vamos a estudiar este fenómeno de pulsación, suponiendo que ambas ondas son sinusoidales y que no existe desfase entre ellas:

$$Ecuacion\ de\ una\ onda\ sinusoidal \rightarrow f(t) = \sin\left(2 \cdot \pi \cdot f\ + \phi\right)$$

$$F(t) = \sin(2 \cdot \pi \cdot f_2) - \sin(2 \cdot \pi \cdot f_1)$$

Donde:

- f_1, es la frecuencia emitida en Hz
- f_2, es la frecuencia reflejada en Hz.
- $F(t)$, es la señal resultante y la que se medirá con el sensor.

Aplicamos a la expresión anterior la propiedad trigonométrica de la resta de senos de distintos ángulos:

$$F(t) = \sin(2 \cdot \pi \cdot f_2) - \sin(2 \cdot \pi \cdot f_1) =$$

$$= 2\cos\left(2\pi \cdot \frac{f_2 + f_1}{2}\right)\sin\left(2\pi \cdot \frac{f_2 - f_1}{2}\right)$$

Sabemos que $f_{reflejada} = \ f_2 \ = f_1 + \Delta f$, y que la frecuencia *Doppler* es mucho menor que la emitida $\rightarrow \ \Delta f \ll f1$. Por lo tanto, el primer término se puede aproximar de la siguiente forma:

$$\frac{f_2 + f_1}{2} = \frac{(f_1 + \Delta f) + f_1}{2} \approx f_1$$

Es decir, la frecuencia recibida será la media de ambas frecuencias, que es aproximadamente la frecuencia emitida.

Y luego tenemos el segundo término:

$$\frac{f_2 - f_1}{2} = \frac{\Delta f}{2}$$

$$f_{batido} = |f_2 - f_1| \ \ (12)$$

A la resta de las frecuencias se le denomina frecuencia de batido, y coincide con la frecuencia *Doppler* en nuestra aplicación. La señal recibida varia su amplitud con una onda de la mitad de la frecuencia de batido, y produce una inversión de la onda (cambio de fase de 180°) a la mitad de este periodo. Sin embargo, en el caso de señales acústicas, el observador no es capaz de notar la inversión de la onda y percibirá un cambio de amplitud a la frecuencia de batido, que será le envolvente de la onda (véase la Figura 2.3.1).

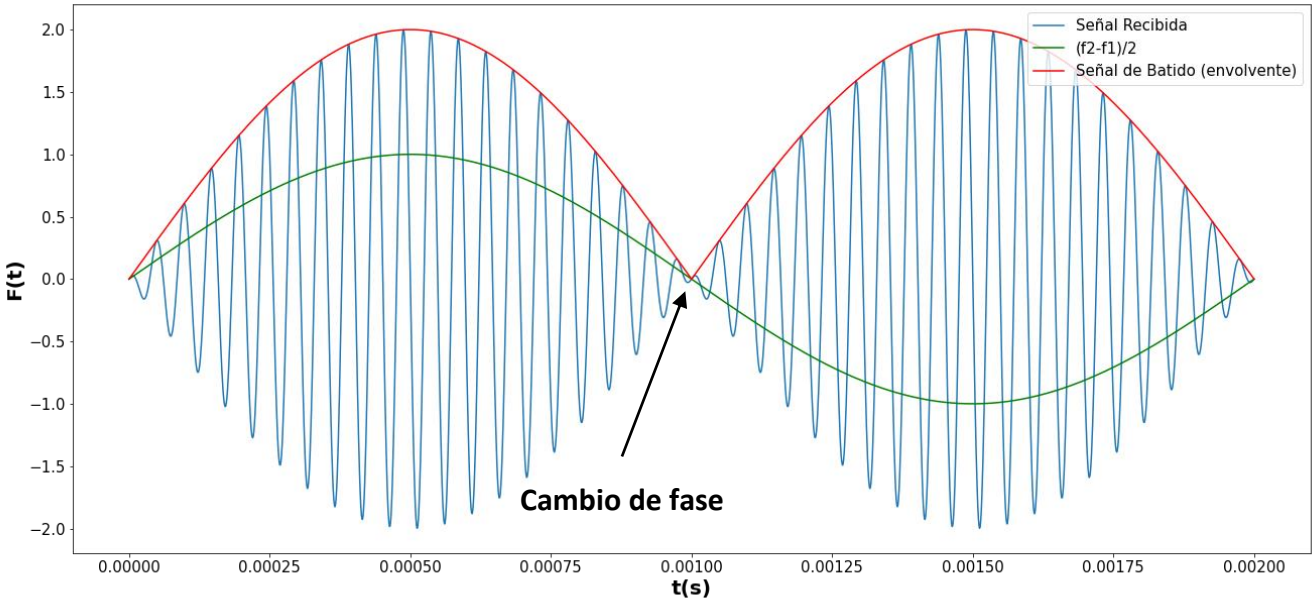


Figura 2.3.1. Batido de frecuencias

La señal obtenida y su envolvente tienen cierta similitud con la modulación en amplitud (AM) tal y como se observa a continuación en la Figura 2.3.2.

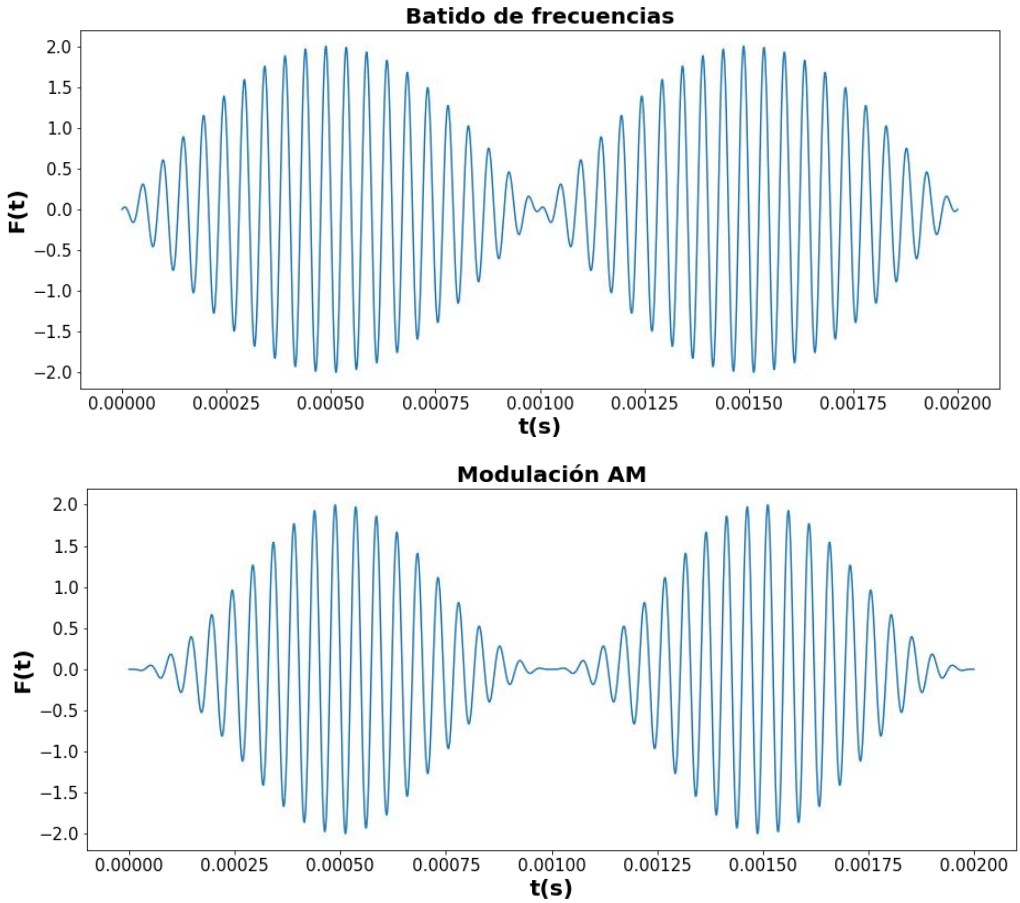


Figura 2.3.2. Comparación del batido de frecuencia con la modulación AM

Por lo tanto, por la similitud que presentan, se pueden aplicar las técnicas utilizadas para la demodulación AM, como pueden ser los detectores de envolventes.

Sim embargo, la situación anterior es para el caso de que la señal reflejada sea de la misma magnitud que la señal emitida, pero en la práctica esto rara vez pasa, pues una parte de las ondas reflejarán en otro sentido y otras serán absorbida por la superficie de reflexión. En su defecto lo que obtendremos una señal modulada con un índice de modulación inferior, además el efecto de inversión de la onda desaparece (Figura 2.3.3).

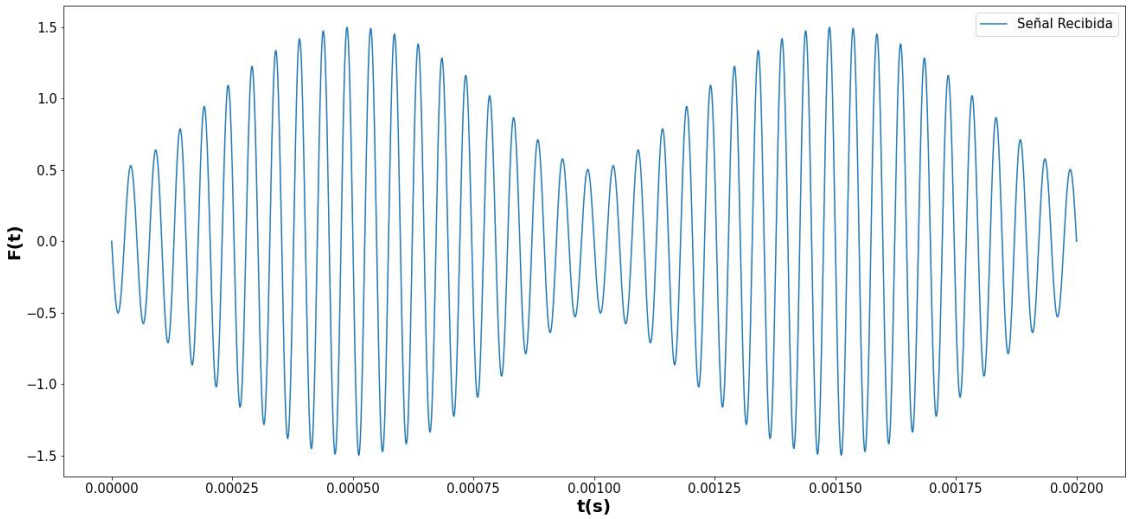


Figura 2.3.3. Señal recibida con la mitad de la amplitud de la señal

2.4 Modulación y demodulación AM

La modulación AM es una técnica utilizada para la transmisión de señales utilizando un medio físico. Fue uno de los primeros modos de transmisión de señales a larga distancia, propagando el mensaje a través del aire, lo que inició la era de la comunicación con la construcción de las primeras radios y televisores, y que hoy en día sigue siendo utilizada. [7]

La modulación AM es uno de los tipos de modulación analógica. Se compone de una señal sinusoidal de frecuencia fija que actúa como la portadora de la señal, y de una señal variable de más baja frecuencia que será nuestro mensaje. El mensaje es transmitido generando cambios en la amplitud de la señal portadora, es decir que la información de la señal se encontrará en la envolvente que une los picos de la portadora. La ecuación típica de la modulación AM es la siguiente:

$$s(t) = A_c \cdot \big(1 + m \cdot x(t)\big) \cdot \cos(2\pi f_c \cdot t + \varphi) \quad (13)$$

Donde:

- A_c, f_c y φ, representan respectivamente la amplitud, la frecuencia y la fase de la portadora.
- $x(t)$, es nuestro mensaje.
- m, es un parámetro que se define como el "índice de modulación". Define en que medida el mensaje afectará a la amplitud de la portadora y en general debería estar entre los valores de 0 y 1.

Como hemos explicado anteriormente la señal que recibimos se encontrará en un estado muy similar a la modulación AM, donde la portadora es la señal transmitida y el mensaje es nuestra frecuencia *Doppler.* Es posible por lo tanto aplicar las técnicas de demodulación AM para extraer el mensaje.

Una de las características más importante de la modulación AM es la facilidad que tiene para poder ser demodulada. Es posible construir con unos pocos componentes un circuito que sea capaz de recuperar el mensaje de la portadora, lo que produjo su rápida expansión en el mercado a través de la comercialización de las primeras radios.

Existen multitud de técnicas de demodulación AM, y en este proyecto nos enfocaremos sobre el circuito detector de envolvente. La envolvente de una señal viene definida por la siguiente expresión:

$$env\{S_{AM}(t)\} = \left|A_C \cdot \left(1 + m \cdot x(t)\right)\right|$$

En el caso de que se cumpla la condición de que m este entre los valores de 0 y 1, y $x(t)$ este normalizada, la expresión anterior será siempre positiva por lo que el valor absoluto se puede quitar y nos quedaría la siguiente expresión:

$$env\{S_{AM}(t)\} = A_C + A_C \cdot m \cdot x(t)$$

La expresión resultante de la envolvente se puede considerar como la suma de una señal continua de valor A_C, y una señal proporcional al mensaje transmitido. Por lo tanto, si tras detectar la envolvente, aplicamos un filtro que nos quite el nivel de continua deberíamos obtener una señal proporcional al mensaje.

3. Hardware utilizado

3.1 Protoboard

Una protoboard, es como su nombre indica una placa de pruebas, que se usa para la creación y prueba de circuitos electrónica. Este compuesto de multitud de filas y columnas, donde se pueden conectar multitud de componentes electrónicos como resistencias, condensadores, integrados, etc. Las filas de cada columna se encuentran conectadas con una división en el medio para poder conectar integrados, y también se dispone de una o varias filas con sus columnas conectadas usadas para las conexiones de alimentación. Esto se ve más claramente en la siguiente imagen (Figura 3.1)

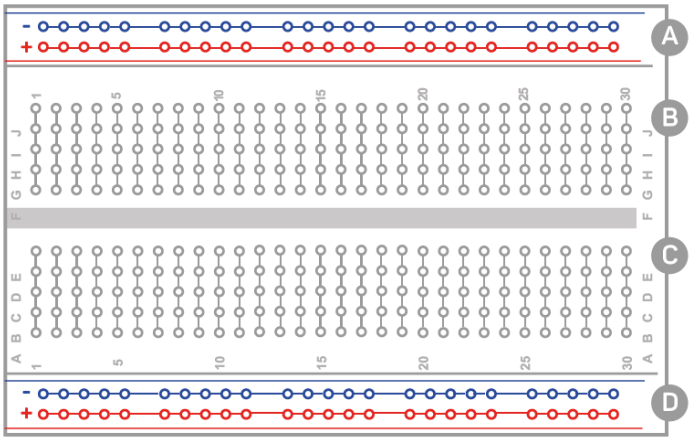


Figura 3.1. Esquema protoboard

3.2 Fuente de alimentación

Una fuente de alimentación (Figura 3.2) es un dispositivo de laboratorio que permite proporcionar las condiciones de alimentación (tensión y corriente) a un circuito o dispositivo eléctrico. Las fuentes de laboratorio mantienen sus salidas reguladas, es decir, con un nivel de tensión o corriente estable, y con un nivel bajo de ruido para asegurar el correcto funcionamiento del circuito y tomar medidas fiables. Disponen de varias salidas ajustables en tensión y corriente, algunas incluso permiten ajustar más parámetros como poner dos salidas en paralelo, para mayor corriente, o en serie para mayor tensión.

Figura 3.2. Fuente de alimentación

3.3 Osciloscopio

Un osciloscopio (Figura 3.3) es un instrumento de medición usado en el análisis de circuito electrónicos, y que permite la visualización y medidas de señales eléctricas. Dispone de una pantalla con dos ejes, el eje X es la escala de tiempo, el eje Y es la escala de tensión. Ambas

escalas se pueden ajustar con sus respectivos selectores. Cuando la señal es periódica, se puede capturar la señal, ve su forma de onda y medir su frecuencia, valores de tensión, etc. Los osciloscopios en su mayoría incorporan al menos dos entradas para poder visualizar dos señales eléctricas y compararlas entre sí. Además, hoy en día, los osciloscopios digitales disponen de varias funciones como “congelar” la pantalla, análisis espectral (FFT), capturar datos, etc.

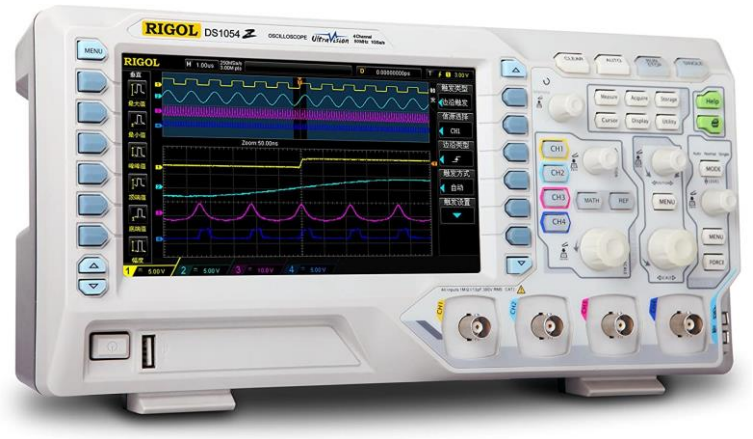


Figura 3.3. Osciloscopio

3.4 Arduino UNO

Arduino UNO (Figura 3.4) es una placa de desarrollo compuesta con un microcontrolador Atmel AVR de 8 bits, el ATmega328P. Arduino es una placa muy utilizada por su facilidad de programación, su programa y hardware libre hacen que existan multitud de librerías y documentación, y permiten la construcción personalizada de la placa por parte de cualquier compañía, por lo que se pueden encontrar en el mercado versiones muy económicas. Es una placa ideal para el inicio en la programación de sistemas embebidos y se puede utilizar en multitud de proyectos.

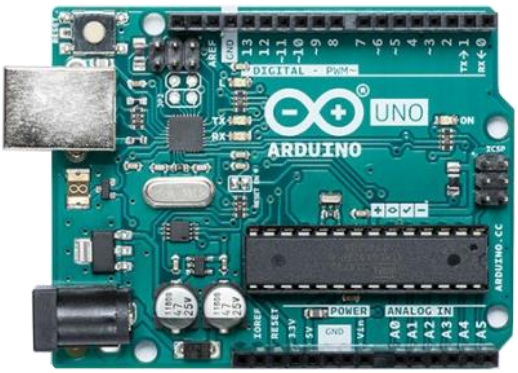


Figura 3.4. Arduino UNO

El Arduino dispone de varias entradas y salidas programables con multitud de usos, se muestran en la siguiente lista:

- **Entradas y salidas digitales (GPIO):** Las *General Port Input/Output* (GPIO), son puertos del Arduino configurables para que actúen como entrada o como salida. Los niveles lógicos de tensión son 0 y 5 voltios, que corresponden con el bit 0 y 1 respectivamente.
- **Entradas analógicas:** Arduino dispone de un convertidor analógico digital (DAC), el cual convierte una señal eléctrica entre 0 y 5 voltios, a una palabra digital de 10 bits, es decir de 0 a 1023.
- ***Output Compare Pins*:** Algunas salidas digitales se encuentran conectados o un comparador digital, el cual permite generar funciones como la modulación por ancho de pulso (PWM).
- **Buses de comunicación:** Conexiones del microcontrolador orientadas a manejar distintos protocolos de comunicación. Arduino Uno permite comunicación I2C, SPI y UART. El bus UART está conectado a un convertidor de UART a Serial y es el que permite transmitir los datos del Arduino al ordenador por USB.

Otras Características técnicas:

Microcontrolador	**Atmega328P**
Tensión de funcionamiento	5V
Voltaje de alimentación (Regulador)	7 – 12 V (Recomendado) 6 – 20 V (Máx)
Entadas/Salidas digitales	14
Output Compare / PWM	6
Entradas analógicas	6
Corriente E/S Máx.	20 mA
Memoria Flash	32 kB
SRAM	2 kB
EEPROM	1 kB
Velocidad reloj	16 MHz

Tabla 1. Características técnicas Arduino UNO. Véase anexo 14.6

3.4 HC-SR04

El HC-SR04 es un sensor de ultrasonidos bastante económico, que se suele operar en conjunto con Arduino, y que está pensado para detectar objetos y medir distancias. Su funcionamiento es simple, dispone de unos dos altavoces uno emisor y el otro receptor (Figura 3.4.1). El altavoz emisor está conectado a un puente H el cual le suministra la corriente necesaria para generar la onda de sonido. El receptor está conectado a varias etapas de amplificación formadas por un conjunto de amplificadores operacionales.

Figura 3.4.1. Sensor HC-SR04

Tanto la salida como la entrada son gestionadas por un microcontrolador el **EM78P153S**, este dispone de dos pines que conectan con el Arduino, el *trigger y* el *echo*. Cuando el microcontrolador recibe un pulso en la entrada *trigger,* enviará al puente H una serie de pulsos a 40 kHz (Figura X). Cuando esa onda de sonido viaja por el medio y rebota contra una superficie, genera una señal en el receptor. La señal amplificada es recibida por el microcontrolador el cual envía como respuesta un pulso en el pin *echo.* Por lo tanto, si se mide el tiempo que se tarda en enviar y recibir la onda, teniendo en cuenta la velocidad de propagación del sonido en el aire, se puede determinar la distancia al objeto. Esta es la aplicación básica del sensor.

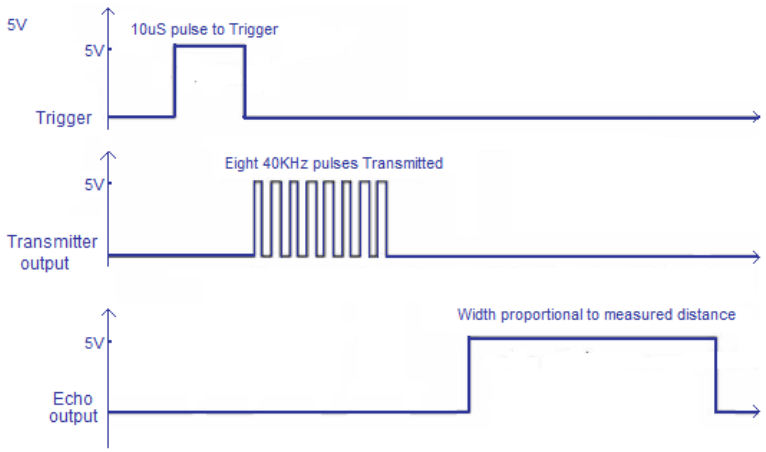


Figura 3.4.2. Funcionamiento del HC-SR04

En este caso, no vamos a medir distancias, sino el cambio de frecuencia de la señal emitida con respecto a la recibida, para con el efecto Doppler determinar la velocidad. Por esto mismo, este microcontrolador, el **EM78P153S**, será removido y la emisión y recepción de la onda será gestionada por Arduino.

Las características principales del sensor son:

Pinout	• Vcc • GND • Trigger • Echo
Alimentación	5V
Frecuencia de funcionamiento	40 kHz
Consumo (stand-by)	< 2mA
Consumo en funcionamiento	15 mA
Distancia de medida	2 cm a 400 cm (mejor resolución < 250 cm)
Ángulo efectico	< 15 °

Tabla 2. Características técnicas HC-SR04 [22]

3.5 Lijadora

En este proyecto estaremos usando una lijadora de correa que nos producirá una superficie en movimiento para poder tomar datos, diseñar y probar el sensor. La lijadora dispone de un motor universal, que es el encargado de hacer mover la cinta, el cual es capaz de funcionar tanto con corriente alterna como con corriente continua. Esta característica la usaremos para alimentar el motor con una fuente de alimentación de corriente continua con tensión regulable, de modo que al variar la tensión cambiara las revoluciones del motor y así podremos probar el sensor con distintas velocidades.

3.6 Encoder

Un encoder es un transductor rotativo, el cual convierte el movimiento rotativo de un eje en pulsos eléctricos para determinar su posición, velocidad y aceleración angular [8].

Un encoder se compone de un disco giratorio hecho por vidrio o plástico, que se encuentra codificado con partes transparentes y opacas, que bloquean el paso de la luz emitida por una fuente de luz infrarroja. La señal de luz es recibida por un sensor óptico que la convierte en una señal eléctrica, generando pulsos digitales que se pueden medir para determinar la dirección del movimiento, la velocidad…

Los encoder tienen multitud de aplicaciones, por ejemplo, se usan en proceso industriales y en robótica para controlar la velocidad de los motores, en informática los encoder se usan para la rueda de los ratones, etc.

En este caso estaremos usando el **E6A2-CS3E** de la empresa Yumo. (Figura 3.6)

Figura 3.6. Encoder E6A2-CS3E

Sus características principales son:

Característica	Descripción
Alimentación	5 a 12 V
Resolución	200 pulsos/revolución
Cable marrón	Vcc
Cable negro	Fase A
Cable blanco	Fase B
Cable naranja	Fase Z
Cable azul	Común (0 V)

Tabla 3. Características técnicas E6A2-CS3E. Véase anexo 14.6

De estas características las que más nos interesan son el voltaje de alimentación que serán los 5 V proporcionados por el Arduino, y la resolución que en este caso es de 200 pulsos por cada revolución.

En nuestra aplicación este el encoder nos permitirá determinar la velocidad de la cinta con una mayor precisión y resolución. Estos datos nos servirán para contrastarlos con la velocidad del sensor de ultrasonidos y comprobar la precisión y la proporcionalidad de los datos.

3.7 STM32F103C8T6

La familia de STM32F103xx, son una serie de microcontroladores de 32 bits que incorpora el alto rendimiento de la arquitectura ARM.[8] Esta familia fue creada y distribuida por la compañía STMicroelectronics especializada en el desarrollo de semiconductores. La compañía clasifica esta familia en tres rangos distintos en base a sus características y rendimientos. Tenemos dispositivos de densidad baja, de densidad media y de densidad alta (Figura 3.7.1).

<table>
<tr><th rowspan="3">Pinout</th><th colspan="2">Low-density devices</th><th colspan="2">Medium-density devices</th><th colspan="3">High-density devices</th></tr>
<tr><th>16 KB Flash</th><th>32 KB Flash(1)</th><th>64 KB Flash</th><th>128 KB Flash</th><th>256 KB Flash</th><th>384 KB Flash</th><th>512 KB Flash</th></tr>
<tr><th>6 KB RAM</th><th>10 KB RAM</th><th>20 KB RAM</th><th>20 KB RAM</th><th>48 KB RAM</th><th>64 KB RAM</th><th>64 KB RAM</th></tr>
<tr><td>144</td><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">5 × USARTs
4 × 16-bit timers, 2 × basic timers
3 × SPIs, 2 × I²Ss, 2 × I2Cs
USB, CAN, 2 × PWM timers
3 × ADCs, 2 × DACs, 1 × SDIO
FSMC (100 and 144 pins)</td></tr>
<tr><td>100</td><td></td><td></td><td colspan="2" rowspan="3">3 × USARTs
3 × 16-bit timers
2 × SPIs, 2 × I²Cs, USB, CAN, 1 × PWM timer
2 × ADCs</td></tr>
<tr><td>64</td><td colspan="2" rowspan="3">2 × USARTs
2 × 16-bit timers
1 × SPI, 1 × I²C, USB, CAN, 1 × PWM timer
2 × ADCs</td><td colspan="3"></td></tr>
<tr><td>48</td><td colspan="3"></td></tr>
<tr><td>36</td><td colspan="2"></td><td colspan="3"></td></tr>
</table>

Figura 3.7.1. Familia STM32F103xx

En la Figura X, podemos ver que esta familia incorpora multitud de funcionalidades como varios puertos para la comunicación UART, SPI, I^2C y comunicación USB, además de disponer de multitud de pines y de una gran cantidad y variedad de memoria incorporada.

En concreto en este proyecto estaré usando el **STM32F103C8T6**, el cual es un microcontrolador de densidad media con 48 pines, con 64 KB de memoria Flash y 20 KB de memoria RAM. Puede operar a una frecuencia máxima de 72 MHz, capturar datos en el ADC a una velocidad de muestreo de hasta 2MSa/s (si se usan los dos ADC) y dispone de varios periféricos como el DMA (*Direct Memory Access*). Estas características son las que utilizaremos para poder usar el STM32 como un dispositivo que nos permita capturar los datos recibidos para posteriormente estudiarlos más detenidamente.

El STM32F103C8T6 como ocurre con Arduino, también se vende con una placa de desarrollo la cual se le conoce comúnmente como "*blue pill*" (Figura 3.7.2). Esta placa es una de las alternativas a usar Arduino que se ha hecho famosa por incorpora un microcontrolador potente a un precio muy económico. Además, como ocurre con Arduino el software de STM32 se distribuye también como licencia abierta, por lo que cualquiera puede descargar, utilizar y compartir el código.

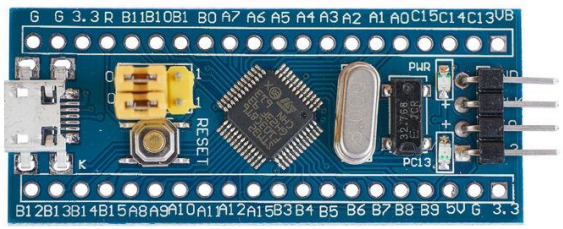


Figura 3.7.2. ***Blue Pill***

A diferencia de Arduino, la “Blue pill” debe ser programada por un programador externo, en este caso se usará el **ST-Link V2** (Figura 3.7.3), que además permite la depuración del código para la detección de fallos o ver el estado de las variables.

Figura 3.7.3. ***ST-Link***

4. Software

4.1 Arduino IDE 1.8.13

El programa Arduino IDE (Integrate Develpmet Environment), es un conjunto de herramientas que permite el desarrollo del código necesario para hacer que el Arduino funcione en base a nuestras necesidades. El programa es realmente simple y está pensado para la iniciación en la programación el mundo de los sistemas embebidos. [9]

El entorno de Arduino está construido sobre el lenguaje de programación de C++, en el cual se puede observar en el código del programa o “*Sketch*” dos funciones principales:

- ***Void Setup:*** Esta función se caracteriza porque solo es ejecutada una vez y al inicio del ciclo del programa. Se usa principalmente para inicializar variables, funciones…
- ***Void Loop:*** Esta función es un bucle infinito y se entra una vez terminado el *Void Setup*. El programa principal se escribirá dentro de esta función y será el que estará ejecutando continuamente.

Estas dos funciones son del tipo *Void* por lo que no devuelven nada. Fuera de ellas se pueden definir funciones y variables globales, e incluir librerías.

Arduino IDE se caracteriza por ser un *software* con licencia libre, que se encuentra apoyado por una gran comunidad de desarrollo que está constantemente incluyendo mejoras y librerías para el control de multitud de dispositivos.

4.2 LTspice XVII

LTspice es un *software* de simulación de circuitos analógicos, que incluye un visor de formas de onda que permite estudiar un circuito antes de crearlo en la parte práctica. Fue creado y distribuido por la empresa de semiconductores ***Analog Devices,*** basado en SPICE. Se caracteriza

por ser un programa ***freeware,*** lo que significa que el usuario final no debe pagar para poder utilizarlo. [10]

LTspice incorpora la mayor parte de los componentes electrónico necesario para el desarrollo de prácticamente cada circuito, además muchos fabricantes de semiconductores también distribuyen un archivo de SPICE para poder incorporar el modelo de su componente. Esto produce que sea uno de los programas más usado en la industria en el análisis de circuitos.

4.3 WXMaxima

Maxima es un programa para la manipulación de expresiones simbólicas y numéricas, incluyendo diferenciación, integración, series de Taylor, vectores, matrices… Produce resultados numéricos exactos aplicando fracciones exactas, números enteros de precisión arbitraria y números de punto flotante de precisión variable. También permite el trazado de funciones de datos en dos y tres dimensiones. [11]

Maxima es un descendiente de Macsyma, un sistema de álgebra por computadora desarrollado a fines de la década de 1960 en el Instituto de Tecnología de Massachusetts.

WxMaxima es distribuido bajo la Licencia Pública General (GNU).

Este programa nos facilitará la obtención de las ecuaciones necesarias que rigen el comportamiento de los circuitos analógicos implementados.

4.4 STM32CubeIDE

STM32CubeIDE es una plataforma de desarrollo basada en C/C++, que dispone de configuración de periféricos, generación de código, compilación y funciones de depuración, para los microcontroladores y microprocesadores STM32. [12]

La plataforma dispone de cientos de microcontroladores STM32 incluyendo la familia STM32F103xx. Recientemente con la incorporación de STM32CubeMX, se puede ahorrar bastante tiempo de instalación y desarrollo aplicando una herramienta de auto generación de código. Esto permite realizar la configuración de los periféricos e inicializar los dispositivos, además permite realizar modificaciones en cualquier parte del proyecto y regenera el código sin generar un impacto al código implementado por el usuario.

Otra herramienta que es interesante en este programa es la depuración del código en tiempo real que permite detectar los fallos mientras se ejecuta el código.

4.5 Processing

Processing es un lenguaje de programación, y es un entorno que permite la creación de proyecto multimedia y de artes visuales. El código que maneja está basado en Java y está orientado para el aprendizaje y creación de prototipos. [13]

Entre sus características destaca:

- Descarga gratuita y de código abierto.
- Programas interactivos con salida 2D, 3D, PDF o SVG
- Compatibilidad con varios sistemas operativos
- Gran variedad de bibliotecas adicionales

En este proyecto se estará usando en conjunto con el software y microcontrolador de ST para recibir los datos en el ordenador, visualizarlo y guardarlos.

4.6 Anaconda y Google Colab

Anaconda es una plataforma de distribución de Python, de código abierta y elegida por muchos para el "*data science*" [14]. Anaconda dispone de múltiples repositorios para encontrar multitud de paquetes basado en el estudio de la ciencia de datos y el aprendizaje automático. También dispone de herramientas para la generación de entornos que permiten crear, distribuir, instalar, actualizar software y gestionar los paquetes de Python y de las aplicaciones. Una de estas aplicaciones que estaremos usando es la de ***Jupyter Notebook,*** que nos permitirá programar en el entorno de Python para poder analizar los datos capturados y generar las gráficas para una mejor visualización.

Por otro lado, tenemos también el entorno de ***Google Colab***, que es un servicio en la nube basado en *Jupyter Notebook,* que pone a disposición un servicio gratuito que ofrece el uso de GPUs, TPU, RAM y todo esto sin la necesidad de instalar nada en el equipo del usuario. Es una alternativa interesante cuando se quiere trabajar entre equipos o cuando es necesario un *hardware* más potente que el equipo del que dispone el usuario.

4.7 Paquete Office

Por último, para la realización de este trabajo también se hizo uso del paquete Office de la empresa Microsoft, bajo la licencia de estudiante proporcionada por la universidad.

Por un lado, la redacción de este trabajo fue realizada en Microsoft Word y también se hizo uso de Microsoft Excel para el análisis de alguno de los datos y para la aplicación de algunas herramientas estadísticas.

5. Implementación del envío.

5.1 Modificación del sensor de ultrasonidos.

El primer paso para la construcción de este proyecto es modificar el sensor de ultrasonidos.

La señal será emitida por el sensor HCSR 04 el cual ha sido modificado eliminando de este el microcontrolador **EM78P153S** encargado de generar los pulsos y generar la señal de respuesta propia del funcionamiento habitual de este sensor. Para ver más detallado el motivo de porque quitamos este sensor podemos observar el siguiente esquema del circuito. (Figura 5.1.1). [15]

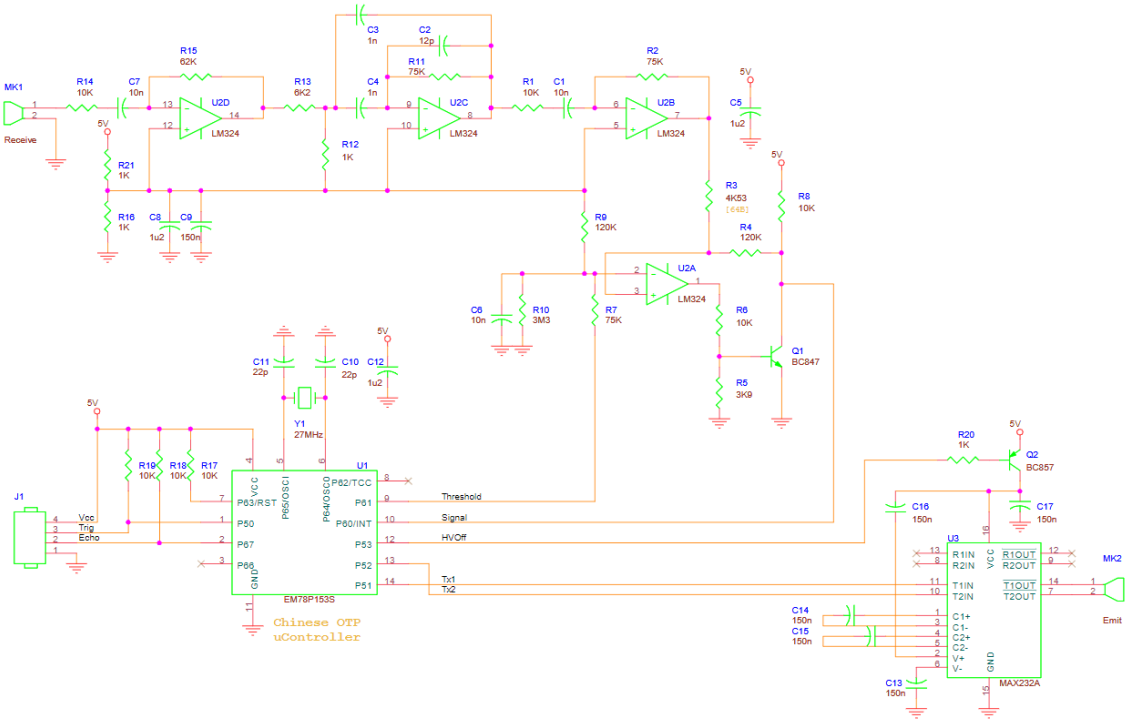


Figura 5.1.1. ***Esquema del circuito del HC-SR04***

Como se puede ver en la figura 5.1.1, el microcontrolador **EM78P153S** está conectado al emisor a través del circuito integrado **MAX232A** y está conectado al receptor a través de una serie de etapas de amplificación. El circuito **MAX232A**¸ consiste en un puente H que amplifica la corriente para poder alimentar correctamente el emisor. Pero esto solo es correcto en las primeras versiones del sensor, ya que ahora se fabrica usando un circuito integrado propio que incluye una etapa de potencia con transistores, construyendo un puente H. La señal que enviaremos al puente H, serán los pulsos generados de unos 40 kHz.

Por otro lado, la señal recibida será extraída de la salida de unas de las etapas amplificación. Es decir, que eliminamos el **EM78P153S** porque seremos nosotros quienes gestionemos él envío y la recepción, pero aprovecharemos los componentes que tiene instalado el sensor para nuestra conveniencia. Tras quitar el microcontrolador el resultado es el siguiente. (Figura 5.1.2)

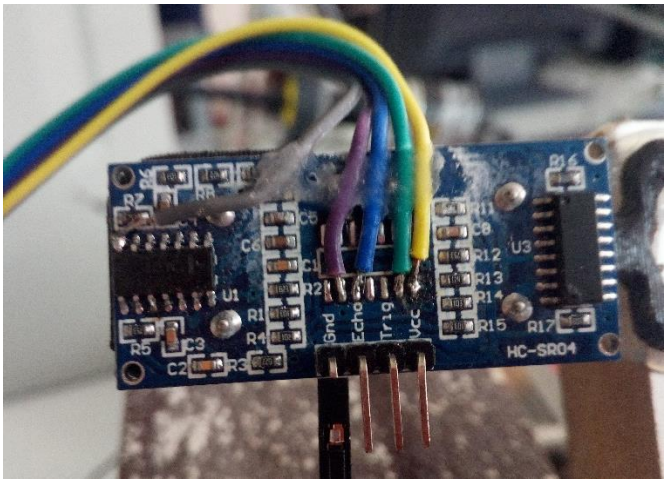


Figura 5.1.2. HC-SR04 Modificado

Los cables amarillo y verde corresponden con las entradas del puente H y serán las que utilicemos para transmitir la señal generada.

Por otro lado, tenemos el cable gris que será la salida de la señal recibida. Está conectado a la salida del operacional U2B del esquema (Figura 5.1.1), que corresponde con la última etapa de amplificación.

La etapa de amplificación desde la recepción del emisor hasta nuestra salida es la siguiente:

1. Operacional U2D es una etapa de amplificación con ganancia 6.
2. Operacional U2C es un filtro de retroalimentación múltiple de primer orden.
3. Operacional U2B es otro amplificador con una ganancia de 8.

Este es el máximo punto de amplificación, pues el amplificador restante el U2A realmente es un comparador con histéresis que actúa junto con el transistor Q1. Todos estos amplificadores amplifican la señal de entrada y le agregan un valor de continua de 2,5 voltios, para que el valor máximo y mínimo de la señal se encuentre entre 5 y 0 voltios.

Por último, los cables, morado y azul son la salida del comparador y de la señal de referencia de 2,5 voltios respectivamente. Estos dos cables se instalaron, aunque al final por las características del proyecto no fue necesario su uso.

5.2 Generador de pulso de 40 kHz.

Con el sensor modificado lo siguiente es la generación de la señal de 40 khz que emitirá el emisor de ultrasonidos, y para ello estaremos usando un Arduino.

La señal que queremos emitir será un tono de 40 kHz, sin embargo, el Arduino no dispone de un convertidor digital analógico y habría que usar electrónica analógica para generar la señal sinusoidal. Este procedimiento es algo costoso y realmente no es necesario, pues podemos plantear otra alternativa.

En este caso se optó por introducir al emisor un senoidal modificada (Figura 5.2.1). Esto consiste en introducir una señal cuadrada centrada en el voltaje pico y con un tiempo muerto. Debido a que un altavoz no puede responder instantáneamente y tiene un recorrido limitado, esto produce por la dinámica del sistema, una respuesta sinusoidal ante esta entrada, como veremos más adelante.

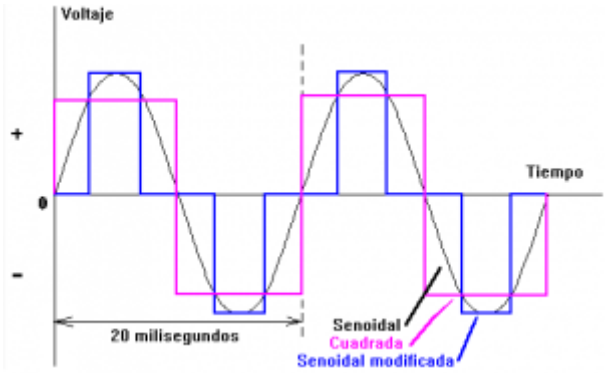


Figura 5.2.1. Sinusoidal modificada [23]

Esta salida se generará mediante la configuración del *hardware* que incorpora el microcontrolador Atmega328P de Arduino. Para ello, el Arduino dispone de varios comparadores de salida, que sirven para diversos usos, como la generación de PWM. En este caso, tendremos que manejar los registros para poder tener la salida deseada.

Podemos observar el diagrama de bloque de los comparadores (Figura 5.2.2). Véase el *datasheet* (Anexo 14.6)

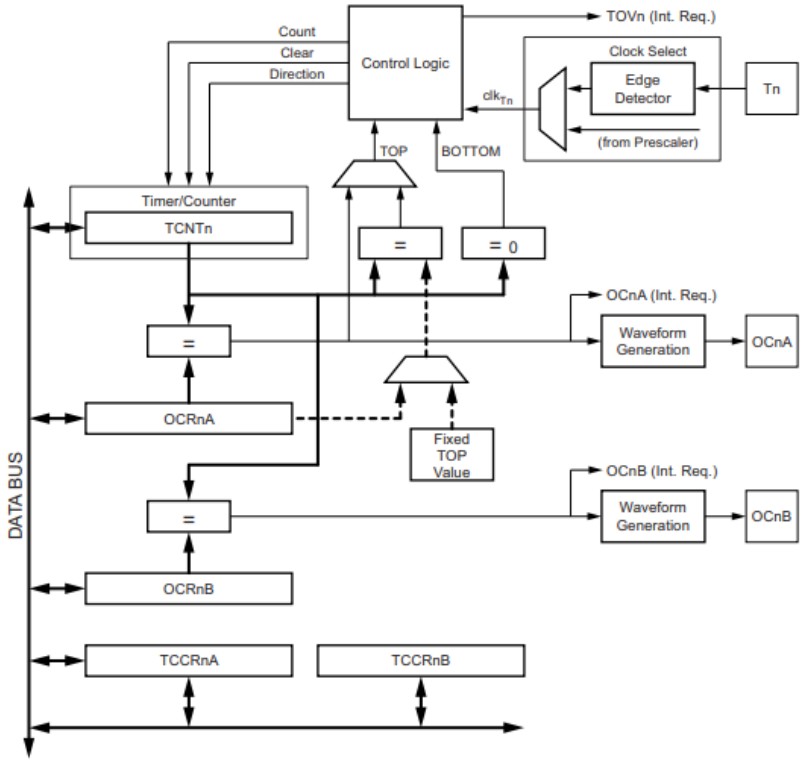


Figura 5.2.2. Diagrama del bloque de los comparadores

Como se puede observar en la figura 5.2.2, disponemos de dos comparadores asociados al mismo temporizador.

Arduino dispone de dos comparadores un de 16 bit (OCR1) y otro de 8 bit (OCR2). Para los propósitos de este trabajo es suficiente con el comparador de 8 bits, por lo tanto, el OCR2.

Los valores del registro que son importantes y que necesitaremos modificar son los siguientes:

- **TCNT2,** es el registro de la cuenta del temporizador y es el que se compara con el valor de los registros de OCR2A y OCR2B.
- **OCR2A y OCR2B,** es el valor de comparación de las salidas A y B. Dependiendo de su valor podremos cambiar el ancho de los pulsos.
- **TIMSK2,** su configuración permite generar una interrupción dependiendo del estado de los comparadores.
- **TCCR2A y TCCR2B,** son los registros encargados de la configuración del modo de comparación, así como habilitar las salidas o establecer el *prescaler.*
- **SREG,** habilita o deshabilita todas las interrupciones.

Como debemos generar una señal de 40 kHz, tenemos que elegir la salida de comparación que más no convenga, que en este caso es el modo CTC.

Este modo se caracteriza porque el valor del contador se resetea cuando se alcanza el valor del comparador OCR2A. Es decir, en este modo según establezcamos el valor del comparador OCR2A cambiaremos el ancho del pulso y por lo tanto la frecuencia. Este es el único modo que nos sirve porque en el resto de los modos el ancho del pulso depende del *prescaler* y de la frecuencia del reloj, que por los valores que tienen no permiten una frecuencia exacta de 40 kHz.

El funcionamiento del modo CTC se nuestra en la siguiente imagen. (Figura 5.2.3)

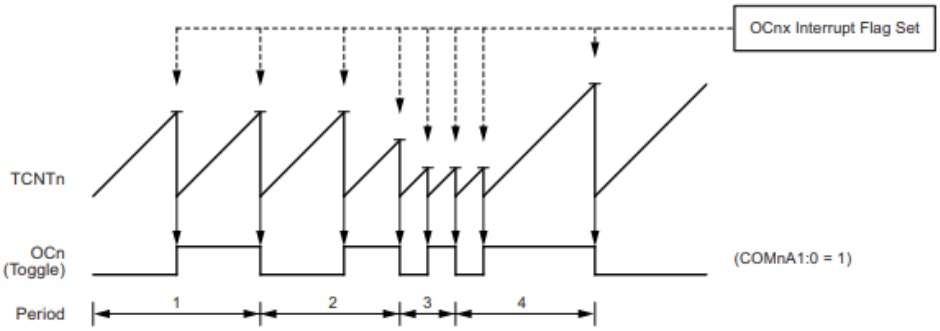


Figura 5.2.3. Funcionamiento del modo CTC

Cuando la cuenta llega al máximo aparte de reiniciar el contador (TCNT2) también nos genera una interrupción.

El ancho de los pulsos de la cuenta se puede calcular de la siguiente manera:

$$T_{pulso} = \frac{N}{f_{CLK}} \cdot (1 + OCR2A) \quad (14)$$

Donde:

- f_{CLK}, es la frecuencia del reloj del Arduino, unos 16 MHz.
- N, es el valor del *prescaler* (1, 8, 64, 256, o 1024).

Con el valor del registro OCR2A vamos a controlar el tiempo muerto y el ancho de los pulsos del a salida A y B.

En un periodo vamos a tener por lo tanto 4 pulsos en el contador, además mientras se genera el pulso en la salida A, la salida B no debe estar activa y, cuando la salida B esta activa la A debe estar desactivada, para hacer funcionar el puente H correctamente, como en la siguiente imagen (Figura 5.2.4)

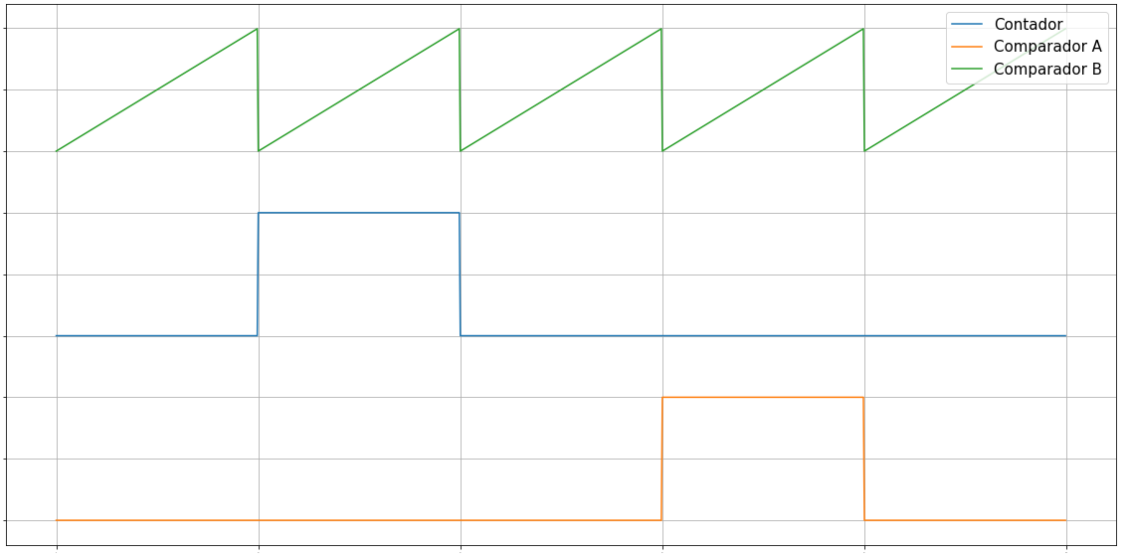


Figura 5.2.4. Salida del comparador generada

Por otro lado, también podemos definir la relación entre el ancho del pulso y el tiempo muerto, lo que puede producir distintos efectos.

Ahora pasamos al explicar el código para implementar esto.

```
//HA es la salida para la alimentacion positiva del altavoz
//HB es la salida para la alimentacion negativa del altavoz

#define HA 11
#define HB 3
#define A 99 // 99 es el valor donde B = A valor más estable

uint8_t a = 0;

void setup() {
  pinMode(HA, OUTPUT);
  pinMode(HB, OUTPUT);

 SREG = (SREG & 0b01111111); //Desabilitar interrupciones
 TIMSK2 = TIMSK2|0b00000010; //Habilita la interrupcion por el comparador A
 TCCR2A = 0b01010010; //Habilita el modo ctc
 TCCR2B = 0b00000001; //Preescaler 1
 //Establecemos los valores de los comparadores
 OCR2A = A;
 OCR2B = A;
 TCNT2 = 0; //Resetea contador

 SREG = (SREG & 0b01111111) | 0b10000000; //Habilitar interrupciones //Desabilitar interrupciones

}
```

Figura 5.2.5. Inicialización del generador de pulsos

La figura 5.2.5, corresponde al apartado del código para la inicialización de la generación de pulsos. Primero se establecen los pines 3 y 11 como salida que corresponden a la salida de los comparadores A y B.

Se deshabilitan las interrupciones y se procede a establecer los registros antes mencionados. Habilitamos con el registro TIMSK2 la generación de la interrupción por la salida A pues nos hará falta más adelante.

Luego se configuran los registros TCCR2A y TCCR2B, para habilitar el modo ctc y con un *prescaler* de 1 es decir una frecuencia de 16 MHz. Para más información acerca de estos registros véase página 127-132 del *datasheet* del atmega328P.

Los registros de los comparadores se han establecido en 99, esto es para poder generar 4 pulsos del mismo ancho. Atendiendo a la ecuación anterior:

$$T_{pulso} = \frac{N}{f_{CLK}} \cdot (1 + OCR2A) = \frac{100}{16 \cdot 10^6} = 6{,}25\ \mu s$$

$$T_{generada} = 4 \cdot T_{pulso} = 25\ \mu s$$

$$f_{generada} = \frac{1}{T_{generada}} = \frac{1}{25 \cdot 10^{-6}} = 40\ kHz$$

En este caso se optó por poner el ancho de los pulsos del mismo tamaño que el tiempo muerto, y esto es debido a que es el modo más fácil de implementarlo y porque en esta situación no hay que estar cambiando el registro OCR2A. Se probó con otros valores, pero el cambio repetitivo en el registro OCR2A producía efectos indeseados.

Con esto establecemos el valor del contador para los 40 kHz, ahora debemos activar y desactivar la salida de los comparadores A y B, cada vez que la cuenta llegue al valor máximo y para ello utilizamos la interrupción del comparador A (Figura 5.2.6).

```
void loop() {

}

ISR(TIMER2_COMPA_vect){
   a++;
   if(a > 3) a = 0;
   switch(a){
    case 0:
     TCCR2A = 0b01100010; //Establece COMPA en toggle y COMPB en clear
    break;

    case 1:
     TCCR2A = 0b10100010; //Establece COMPA en clear y COMPB en clear
    break;

    case 2:
      TCCR2A = 0b10010010; //Establece COMPB en toggle y COMPA en clear
    break;

    case 3:
      TCCR2A = 0b10100010;  //Establece COMPA en clear y COMPB en clear
    break;
   }

}
```

Figura 5.2.6. Interrupción del comparador A

En la figura 5.2.6 se puede ver la interrupción del comparador A. Disponemos de una variable que incrementa su valor en cada interrupción tomando un máximo de 4 estados correspondientes a los mostrados en la figura 5.2.4. En función del estado establecemos la salida

de los comparadores para que cambien su estado al finalizar la cuenta o para que mantengan su valor en 0.

Tras realizar la programación del Arduino y conectar el sensor podemos comprobar en el receptor que efectivamente la señal recibida es sinusoidal, de 40 kHz y con una componente de continua de 2,5 V aproximadamente (Figura 5.2.7).

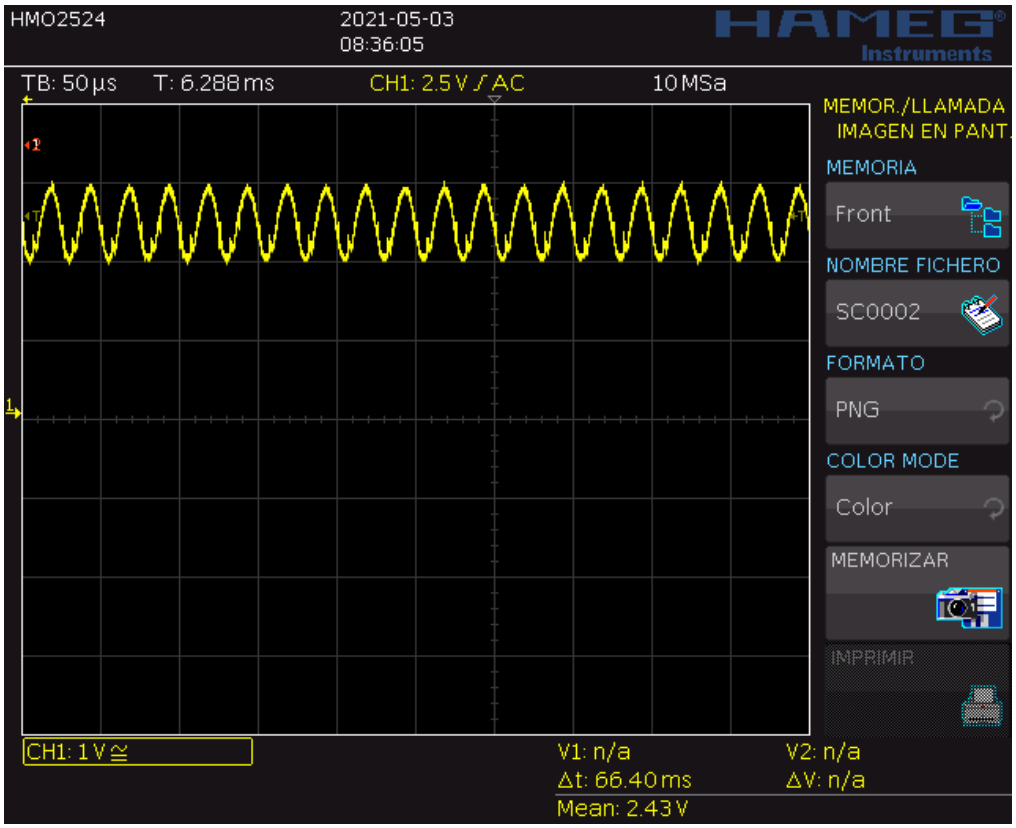


Figura 5.2.7. Señal recibida

5.3 Banco de pruebas.

Ya tenemos el sensor preparado para poder emitir y recibir una señal, sin embargo, debemos someter al sensor a diversas pruebas para mirar su comportamiento ante la medida de distintas velocidades.

Para generar una velocidad y estudiar el efecto *Doppler*, no moveremos el sensor como se realizaría en el caso real, pues la disposición de los cables haría muy complicada esta tarea, si no que moveremos la superficie que apunta el sensor. Si la superficie de reflexión se mueve, el efecto *Doppler* se producirá de la misma forma sin afectar al funcionamiento del sensor.

Esta superficie móvil la realizaremos en el laboratorio con una lijadora de cinta, la cual dispone de un motor universal que nos permitirá probar distintas velocidades según la tensión de alimentación que le apliquemos. Usar la lijadora como banco de pruebas nos trae ciertas ventajas por la forma de instalación y la superficie.

En primer lugar, el sensor se instalará apuntando a la cinta en un ángulo aproximado de 45°, por lo que ni el ángulo ni la distancia con respecto a la superficie cambiarán. Por otro lado, la

superficie de la lijadora es porosa y sin variaciones grandes en el relieve, un caso bastante idóneo para facilitar la reflexión de la onda.

Por otro lado, instalaremos en la misma cinta un *encoder* con una rueda que nos medirá la velocidad de la cinta. El *encoder* posee bastante resolución y además estará siempre en contacto con la cinta, por lo que no tendremos los problemas típicos de este sensor. Usaremos el *encoder* como un valor de comparación entre la velocidad real de la cinta y la velocidad obtenida con el sensor de ultrasonidos. La instalación final de este banco de prueba nos queda de la siguiente forma. (Figura 5.3)

Figura 5.3. Banco de pruebas del sensor

5.4 Detector de envolvente.

El siguiente paso es extraer la señal *Doppler* realizando la demodulación AM de la señal recibida.

Para la demodulación AM existen multitud de métodos, pero actualmente el más fácil de implementar y el más económico consiste en el método de detección de envolvente.

Este circuito tuvo un gran éxito y fue implementado en multitud de dispositivos como en las radios, televisiones canales móviles y en general en cualquier sistema de difusión masiva. [7]

El detector de envolvente consiste en un circuito compuesto de un filtro RC y un diodo rectificador. El diodo produce que solo pasen los picos de la señal modulada, acumulando la tensión en el condensador, luego el condensador se descarga a través de la resistencia suavizando la caída hasta el siguiente pico (Figura 5.4.1)

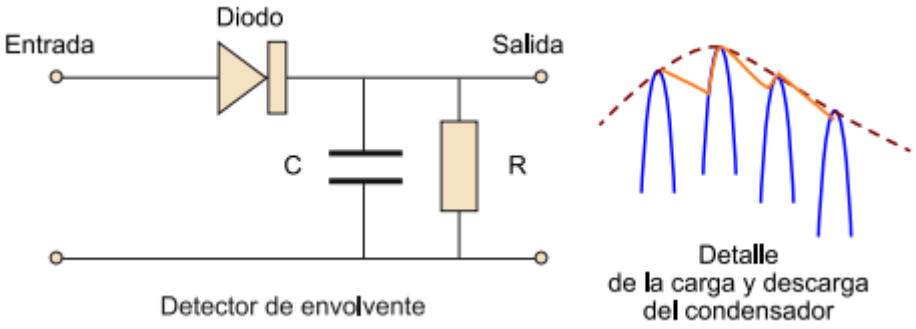


Figura 5.4.1. Esquema y funcionamiento del detector de envolvente

Es importante seleccionar correctamente el valor de la resistencia y el condensador. Para ello se define la constante de tiempo del filtro:

$$\tau = RC$$

Si la constante del filtro es un parámetro muy crítico, si es demasiado pequeña, el condensador se descargará muy rápido, pasando demasiada cantidad de la señal portadora y por lo tanto no demodulándose correctamente. Por otro lado, si la constante es demasiado grande, la descarga será demasiado lenta y no seremos capaces de recoger correctamente la envolvente de las ondas de mayor frecuencia. Sin embargo, en nuestro proyecto no están importante este último caso, pues a nosotros solo nos importa captar una señal periódica con la frecuencia *Doppler* y da igual si la señal esta algo deformada, por lo que es preferible en este caso seleccionar una constante de tiempo mayor.

Los valores de C y R se pueden seleccionar siguiendo la siguiente desigualdad:

$$B_X \ll \frac{1}{RC} \ll f_C \quad (15)$$

Donde:

- $\boldsymbol{B_X}$, es el ancho de banda de la señal en Hz
- $\boldsymbol{f_C}$, es la frecuencia de la portadora en Hz

Esta desigualdad viene a explicar que la constante de tiempo de la portadora (el periodo) ha de ser mucho menor que la constante de tiempo del filtro. Al mismo tiempo la constante del filtro ha de ser mucho menor que la constante de tiempo del ancho de banda, es decir del periodo de la frecuencia más alta que se recibirá.

La frecuencia de la portadora es aproximadamente la frecuencia emitida es decir 40 kHz.

Y la frecuencia máxima de la señal del efecto *Dop*pler la podemos estimar con la expresión:

$$Usando\ (10) \rightarrow\ V = \frac{c}{f_c \cdot 2 \cdot \cos(\theta)} \cdot \Delta f$$

$$B_X = \Delta f = \frac{f_c \cdot 2 \cdot \cos(\theta)}{c} \cdot V_{max} \quad (16)$$

Como este sensor será instalado para un robot de interior la velocidad máxima no será mayor de 25 $\frac{km}{h}$, aproximadamente, lo que equivale a unos 6,9 $\frac{m}{s}$.

Sabiendo que la velocidad del sonido a presión atmosféricas y en condiciones normales de temperaturas es de 340 $\frac{m}{s}$, si suponemos el peor de los casos con $\cos(\theta) = 1$:

$$B_X = \frac{40 \cdot 10^3 \cdot 2}{340} \cdot 6{,}9 \approx 1600\ Hz$$

En la práctica el ancho de banda es mucho menor pues el ángulo de instalación no puede ser de 0°, pero sabemos que si cumplimos con este criterio estaremos operando correctamente en cualquiera de los casos.

Los valores para R y C fueron elegidos en base a los valores normalizados y cumpliendo con el criterio anterior.

$$R = 10\ k\Omega$$

$$C = 47\ \rho F$$

Verificamos que se cumple la desigualdad:

$$B_X \ll \frac{1}{RC} \ll f_C \quad \rightarrow \quad 1600 \ll 2127 \ll 40000$$

Además, podemos comprobar con la siguiente simulación en LTSPICEVII que los valores seleccionados son correctos. Para la configuración de la simulación se ha supuesto un caso donde la frecuencia modulada es de 1500 Hz y que la señal recibida ha llegado con la mitad de la amplitud. A parte, se ha seleccionado el diodo utilizado en el proyecto el **1N4148**.(Figura 5.4.2)

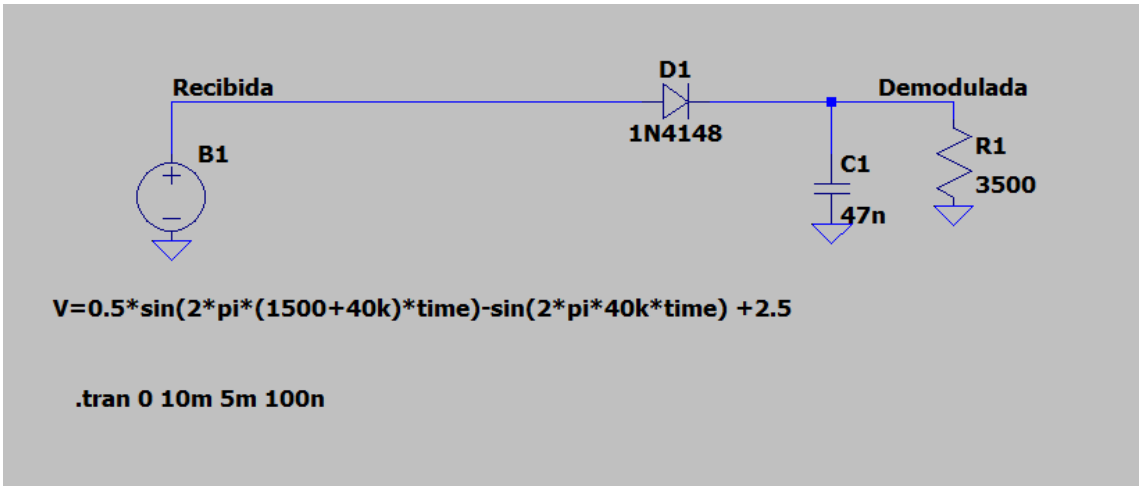


Figura 5.4.2. Configuración de la simulación del detector de envolvente

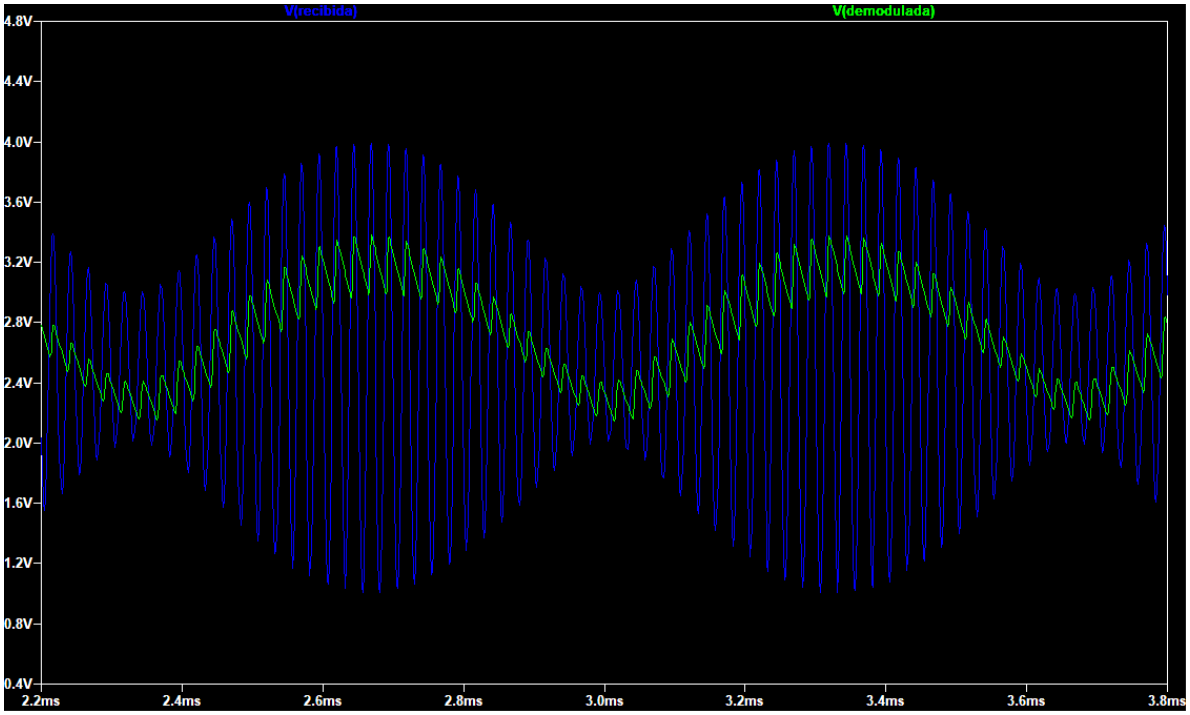


Figura 5.4.3. Resultados simulación del detector de envolvente

Como se puede observar en la figura 5.4.3, el circuito es capaz de recoger la envolvente de la señal modulada. La señal demodulada posee una cierta cantidad de rizado de la frecuencia de la portadora, pero esto se soluciona fácilmente con otra etapa más de filtrado. También se observa que la señal modulada es de menor amplitud debido a la caída de tensión del diodo que es aproximadamente de 0,7 V (diodo de silicio).

Esta carga y descarga del circuito también la podemos observar en el montaje experimental con el osciloscopio. (Figura 5.4.4)

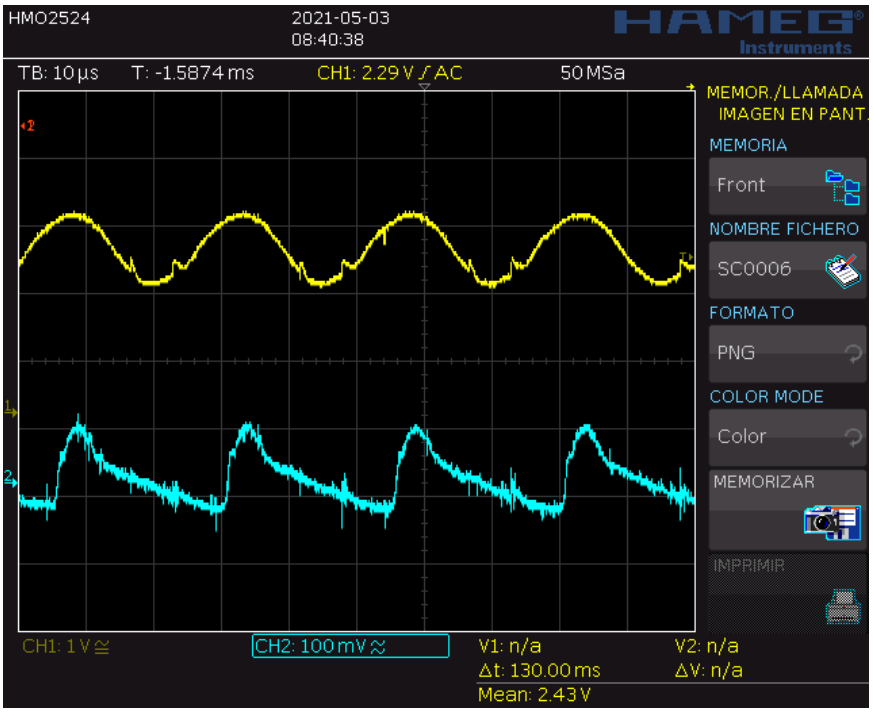


Figura 5.4.4. Carga y descarga del circuito demodulador

6. Análisis de la señal recibida.

6.1 Captura de datos con STM32.

Ya hemos generado la señal, recibido e incluso hemos realizado la demodulación para extraer la señal del efecto *Doppler*, sin embargo, necesitamos conocer el comportamiento de la onda ante distintos estímulos, para encontrar luego la configuración y el circuito que mejor se adapte para convertir la señal en una que el Arduino pueda manejar sin demasiadas complicaciones.

Es aquí cuando vamos a realizar una captura de los datos recibidos usando el microcontrolador STM32F103C8T6. La idea es capturar los datos en un intervalo de muestreo fijo y conocido para poder luego visualizar la onda, ver su espectro en frecuencia con una FFT, e incluso generar un archivo de datos que permita introducir la onda en el simulador.

Empecemos generando del código de inicialización del STM32 con la herramienta de STM32CubeMX que incorpora el *software* de STM32CubeIDE.

Lo primero es establecer los pines se van a utilizar (Figura 6.1.1).

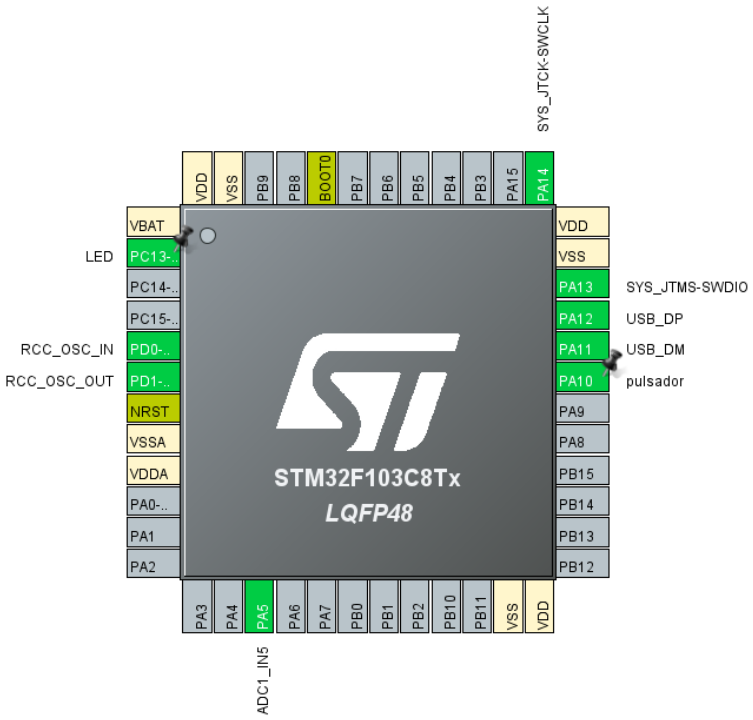


Figura 6.1.1. Seleccion de pines STM32

Utilizaremos el pin del LED (PC3) como indicador de que los datos ya están disponibles. Usaremos un botón (PA10) para iniciar manualmente la captura de los datos, que serán convertidos a través del puerto analógico PA5. Por último, se hará uso de la comunicación USB para transmitir los datos al ordenador a través de puerto virtual y del reloj externo de alta velocidad de 8 MHz.

El pulsador se configura como entrada con *Pull-Up* y el Led como salida. (Figura 6.1.2)

GPIO | ADC | RCC | SYS | USB

Search Signals

Search (Crtl+F)

Pin Name	Signal on Pin	GPIO output level	GPIO mode	GPIO Pull-up/Pull-down	Maximum output speed	User Label
PA10	n/a	n/a	Input mode	Pull-up	n/a	pulsador
PC13-TAMPER-RTC	n/a	Low	Output Push Pull	No pull-up and no pull-down	Low	LED

Figura 6.1.2. Configuración GPIOs STM32

Para la entrada analógica utilizaremos el convertidor ADC1 en el pin PA5, el cual tiene una resolución de 12 bits. La configuración es la siguiente (Figura 6.1.3)

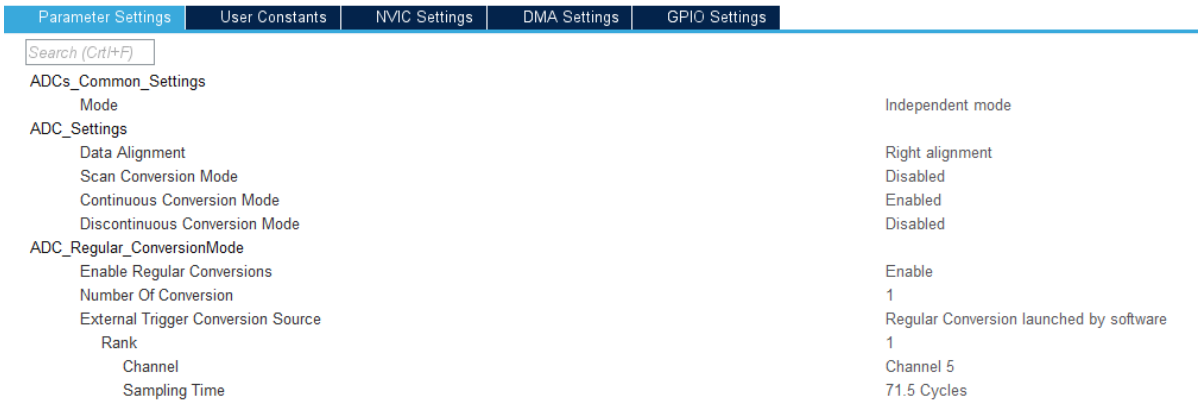


Figura 6.1.3. Configuración ADC STM32

De esta configuración los aspectos más destacables es el *Continuos Conversion Mode*, que habilita el modo de conversión continua, es decir, el ADC estará convirtiendo todo el rato por lo que el registro del ADC se mantendrá siempre actualizado con el último dato recogido.

El otro parámetro importante es el *Sampling Time* que nos determinará junto con los *prescaler* la frecuencia de muestreo. El tiempo de conversión viene explicado en el *Reference Manual* del STM32F103[18]:

$$T_{conv} = Sampling\ Time + 12{,}5 \quad (17)$$

Debemos de establecer un tiempo de conversión correcto junto con la frecuencia del ADC, para que la frecuencia de muestreo nos permita recoger el rango de frecuencias a estudiar. Esto implica que para que podamos observar la frecuencia de la portadora, los 40 kHz, debemos de elegir una frecuencia de muestreo que sea igual o superior al doble de esta frecuencia según el teorema de muestreo de Nyquist. Por lo tanto, debemos de seleccionar una frecuencia superior a los 80 kHz.

Esto nos lleva al siguiente apartado que es la configuración de los *timers,* que es otro aspecto en el que nos facilita bastante la herramienta de STM32CubeMX. (Figura 6.1.4)

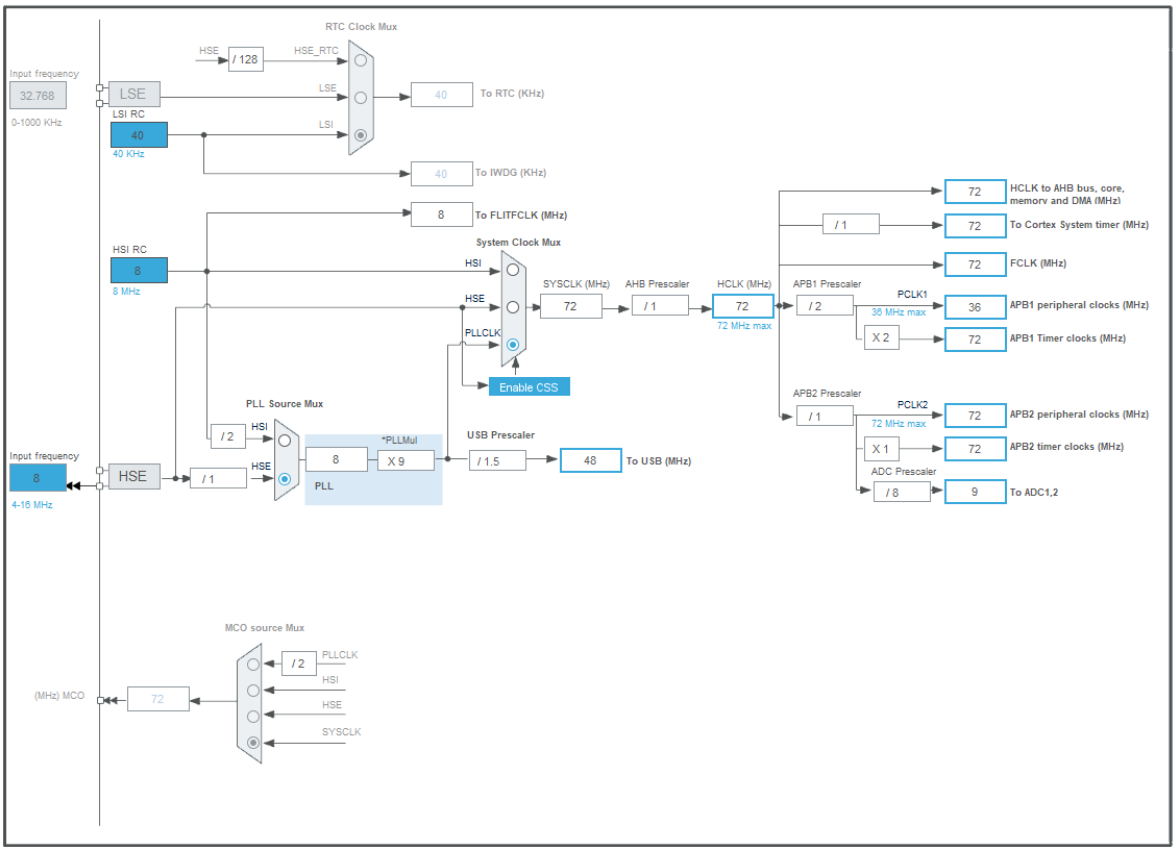


Figura 6.1.4. Configuración *Timers* STM32

Se han seleccionado como frecuencias destacables una frecuencia de 72 MHz para el reloj, la frecuencia máxima seleccionable. Y para el ADC una frecuencia de unos 9 MHz.

La frecuencia de muestreo es, por lo tanto:

$$f_{muestreo} = \frac{9 \cdot 10^6}{71{,}5 + 12{,}5} = 107{,}14\ kHz > 80\ kHz$$

Por último, utilizaremos un periférico del microcontrolador que se encargará de gestionar los datos del conversor analógico-digital, el DMA. El DMA (*Direct Memory Access*), es un periférico que tiene la capacidad de poder gestionar el traslado de memoria de un periférico a otro, de una posición de memoria a otra, o de periférico a memoria y viceversa. Todo esto, sin la necesidad de la intervención del microprocesador del STM32, por lo que puede liberar al núcleo principal de tener que gestionar esta tarea. En este caso, el DMA se encargará de llevar los nuevos datos obtenidos en el ADC a un *buffer* en la memoria SRAM.

Figura 6.1.5. Configuración *DMA* STM32

En la figura 6.1.5, tenemos la configuración del DMA. Se encargará de llevar los datos desde nuestro periférico (ADC) a la posición de memoria asignada en la SRAM. El tamaño de los datos es de media palabra, 16 bits pues el tamaño de una palabra es de 32 bits.

El DMA empezara a realizar la transferencia de datos cuando se lo indiquemos y generara una interrupción cuando haya finalizado.

Tras realizar todas las configuraciones pasamos a escribir el código el cual se escribe en el "main.c". A diferencia de Arduino el STM32 se programa preferiblemente en C, aunque también es posible incorporar C++.

Primero definimos el tamaño del Buffer de los datos recogido, en total será de 5000 datos de 16 bits. Esto implica con la frecuencia de muestreo, que la frecuencia de resolución mínima es aquella que pueda entrar en un periodo completo.

$$f_{min} = \frac{f_{muestreo}}{5000} = \frac{107140}{5000} \approx 21{,}43\ Hz$$

también definimos un *Buffer* para el envío de datos y calculamos en cuantos paquetes tenemos que dividirlo. También añadiremos una variable para cuando el *Buffer* de envío este completado y otra para detectar el flanco del pulsador (Figura 6.1.6)

```
#define LEN 5000 //Tamaño del buffer de los datos recogidos
#define BUFFERLEN 2001 //Añado 1 más como identificador de la cadena
#define CHAINNUM LEN/((BUFFERLEN-1)/2) //Determina en cuantos paquetes dividir el buffer

uint8_t flancoP = 0;
uint8_t bufferListo = 0;
```

Figura 6.1.6. Configuración *DMA* STM32

Ahora pasamos a realizar la parte del código que se ejecutará de manera continua y esto es con el siguiente bucle infinito. (Figura 6.1.7)

```
while (1)
{     //Obtenemos el estado del pulsador
    uint8_t state = HAL_GPIO_ReadPin(pulsador_GPIO_Port,pulsador_Pin);

    //Verificamos el flanco de subida del pulsador
    if(!state && !flancoP) flancoP = 1;
    else if (state && flancoP) flancoP = 0;
    else state = 1;

    if(datosListos){
        //Si los datos ya estan listos generamos el buffer y lo enviamos
        if(!bufferListo){
            for(int j = 0; j < CHAINNUM; j++) {
                for(int i = 0; i < (BUFFERLEN - 1)/2; i++) {
                    crearBuffer(datos[i+j*(BUFFERLEN - 1)/2], &buffer[2*i], &buffer[2*i + 1]);

                }
                //Añadimos a paquete su número para identificarlo en la recepción
                buffer[BUFFERLEN-1] = j;

                for(int k = 0; k < 5; k++){
                    //Trasmitimos y dejamos un tiempo entre envio
                CDC_Transmit_FS(buffer, BUFFERLEN);
                HAL_Delay(50); // 50 ms
                }

            }
            //Encendemos el Led y establecemos que el envio ha finalizado
            bufferListo = 1;
            HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 0);

        }

    }
    if(!state) {
        //Si se pulsa el boton y se solicitan nuevos datos
        datosListos = 0;
        bufferListo = 0;
        //Empezamos la conversion por DMA
        HAL_ADC_Start_DMA(&hadc1, (uint32_t *) datos, LEN);
        //Apagamos el LED
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 1);

    }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

Figura 6.1.7. Código del bucle STM32

En el código hacemos uso del conjunto de funciones que incorpora la librería HAL. La función de HAL_GPIO_ReadPin se usa para leer el estado del pulsador, y HAL_GPIO_WritePin para establecer el valor del LED. La función que inicia el DMA es HAL_ADC_Start_DMA, la cual genera una interrupción finalizada la conversión, llamando a la función HAL_ADC_ConvCpltCallback . (Figura 6.1.8). Para más ver con mejor detalle cada función véase el manual de usuario de las funciones HAL. [16]

```
//Cuando los datos han finalizado se ejecuta la siguiente interrupción
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    //Establecemos que los nuevos datos ya están listos
    datosListos = 1;

}
```

Figura 6.1.8. Interrupción del DMA

Una vez que los datos estén listos generamos un nuevo *Buffer* de 8 bits, donde se guardarán los datos del *Buffer* del ADC el cual es de 16 bits.

Hay que convertir por lo tanto los datos de 16 bits a 8 bits y para ello, lo que hacemos es dividir repartir el valor de los bits en dos bytes distintos con la siguiente función. (Figura 6.1.9)

```
void crearBuffer(uint16_t dato, uint8_t *Hbyte, uint8_t *Lbyte ){
    //Pasamos el vector por referencia y separamos los datos originales
    // de 16 bits en dos de 8 bits

    *Lbyte = dato & 0xFF;
    *Hbyte = dato >> 8;

}
```

Figura 6.1.9. Creación del nuevo *Buffer*

Por último, el envío se gestiona con la función CDC_Transmit_FS que incorpora la librería "usbd_cdc_if.h". Hay que aclarar que el código mostrado no es el mejor modo de realizar el envío ni el más eficiente, pues no se está estableciendo ningún método de verificación de errores, para comprobar si se han recibido todos los paquetes. En su defecto se optó por enviar el paquete varias veces y comprobar a posteriori si están todos los nuevos datos, que para el uso dado es más que suficiente.

Esto es lo necesario para que el código de recogida y transmisión de los datos funcione. Para poder recoger los datos las conexiones son sencillas, el único aspecto importante es recordar que el STM32 opera a 3,3 V por lo que si se le aplica más tensión a uno de los pines puede estropear el microcontrolador. Para evitar esto, podemos utilizar primero un amplificador operacional en configuración de seguidor de tensión, posteriormente utilizar un partidor de tensión con dos resistencias y luego para mayor seguridad puede poner un Zener o usar otro seguidor de tensión de forma que sature antes de los 3,3 V.

6.2 Recepción de los datos con Processing.

El STM32 junto con el código anterior, nos permiten enviar los datos por un puerto serial virtual al ordenador, pero necesitamos tener una interfaz o un programa que pueda recibir estos datos para poder estudiarlos. Aquí cuando entra el uso del programa Processing, que aparte de recibir los datos nos permitirá visualizarlo y generar un archivo con los datos recibidos.

Processing al igual que Arduino dispone en si plantilla del *Sketch* dos funciones, la función *Void Setup,* y la función *Void Draw.* Ambas funciones son análogas a las funciones de Arduino *Void Setup* y *Void Loop* respectivamente. Processing es un entorno de programación basado en Java, aunque posee cierta similitud con C, y esto le hace una herramienta ideal junto con la librería "*processing.serial.**", para poder hacer proyectos visuales en conjunto con Arduino y otros microcontroladores.

Lo primero que realizamos en el código es llamar a la librería y definir la clase. También creamos los *Buffers* de datos y de la recepción, y algunas variables para registras los eventos ocurridos. (Figura 6.2.1)

```
//Importamos la librería de comunicación serial y llamamos a la clase
import processing.serial.*;
Serial puerto;
//Definimos las constantes de tamaño de los buffer
final int buffersize = 2001;
final int datossize = 5000;
//Generamos el buffer de todos los datos
int[] datos = new int[datossize];

//Creamos una variable para determinar si hay dispositivo conectado
boolean comLista = false;

//Generamos el buffer de la recepción de los paquetes
 byte[] Buffer = new byte[buffersize];
 boolean evento = false;

//Generamos la clase del fichero
 PrintWriter fichero;
 boolean escrito = false;
```

Figura 6.2.1. Declaraciones de Processing

En el *Void Setup*, inicializamos la comunicación y definimos el puerto.(Figura 6.2.2)

```
void setup() {
  //Inicializamos la comunicacion y miramos los elementos conectados en el puerto COM
  String[] lista = Serial.list();

   if(lista.length != 0){
    for(int i = 0; i < lista.length; i++){
    println("Lista[" + i + "]= " + lista[i]);
    }
    //Elegimos el primer elemento de la lista
    // (puede ser otro si tiene mas de un dispositivo conectado)
  puerto = new Serial(this,lista[0], 115200); //Los baudios en el caso del STM32 no son relevantes
  comLista = true;

  puerto.buffer(buffersize);
   }

   else
   //Generamos el fondo
   size(800,600);
   background(0);
  frameRate(30);

}
```

Figura 6.2.2. *Void Setup* de Processing

El código implementado hay multitud de funciones que son para poder realizar el dibujo de la señal. Vamos a centrarnos en la recepción y en la conversión de los datos, si desea consultar todo el código se encuentra al final de este documento en el Anexo.

Cuando se reciben los datos, se llama a la siguiente función: (Figura 6.2.3)

```
//Si se produce una llegada de los datos
void serialEvent (Serial c){
  //Leemos los bytes y los guardamos en el Buffer
   Buffer = puerto.readBytes();
   evento = true;

}
```

Figura 6.2.3. *Evento Serial*

Cuando se produzca el evento, guardamos los datos en el *Buffer* y establecemos el estado de nuestra variable "evento" en "*true*".

Esto nos lleva a la siguiente parte del código donde generamos el *Buffer* de todos los datos reconvirtiendo los datos recibidos. (Figura 6.2.4)

```
int j = Buffer[buffersize -1];
if(j >= 0 && j <=4){
  j = j*(buffersize-1)/2;

  for(int i = 0; i < (buffersize-1)/2; i ++){
//Reconstruimos los datos de nuevo a 16 bits y vamos guardando segun
// lleguen los nuevos paquetes

    datos[i + j] = (Buffer[2*i] << 8) + (Buffer[2*i +1] & 0xFF);

  }
}
```

Figura 6.2.4. Reconversión de los datos

Lo que se hace es generar un nuevo dato volviendo a unir los bits más significativos y los menos significativos. La posición del *array* se desplaza a medida que recorremos el *Buffer* de recepción y el número del paquete incrementa.

Estos datos se representan luego en una ventana generada por el programa, a través de las funciones creadas de "grid" y "lineas". (Figura 6.2.5)

```
background(0);
float largo = 800;
float ancho = 600;
float rlargo = largo - largo*0.1;
float rancho = ancho - ancho*0.04;
 //Pintamos la lineas de fondo
 grid(largo,ancho, rlargo, rancho);

 //Dibujamos lineas entre cada punto de los datos
 lineas(largo,ancho,rlargo, rancho, datos);
```

Figura 6.2.5. Creación del dibujo

Las variables de "largo" y "ancho" definen las dimensiones de la ventana entre otras cosas.

Por último, tenemos un fragmento de código donde creamos y generamos un fichero con los datos recibidos, para posteriormente utilizarlo en otros entornos que nos permitan realizar otro tipo de análisis como su espectro en frecuencia o para incluirlo en las simulaciones. (Figura 6.2.6)

```
if(Buffer[buffersize -1] == 4 && !escrito){
// Si ha llegado el último paquete
// Escribimos el fichero con los datos
   fichero = createWriter("datos.txt");
   for(int i = 0; i < datossize; i++) fichero.println(datos[i]);
   escrito = true;
   fichero.flush();
   fichero.close();
}
```

Figura 6.2.6. Creación del fichero Processing

Este código nos generara entonces una ventana con el siguiente aspecto. (Figura 6.2.7)

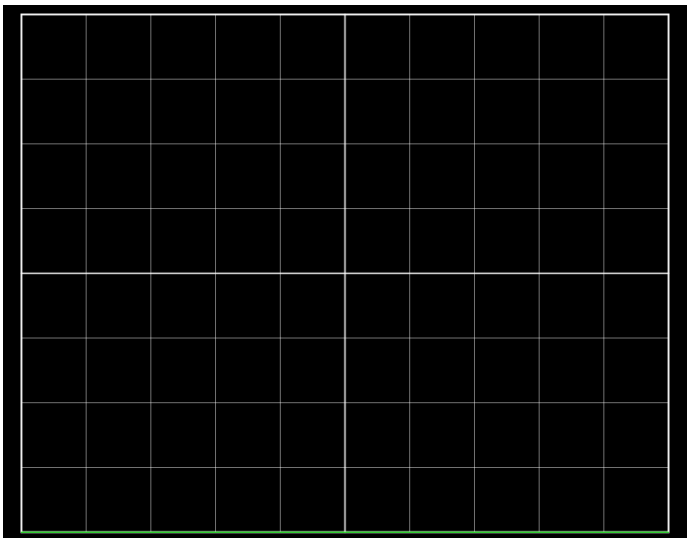


Figura 6.2.7. Ventana Processing sin datos

Ahora ya podemos capturar unos cuantos datos probando distintos puntos y a medir antes y después de la demodulación.

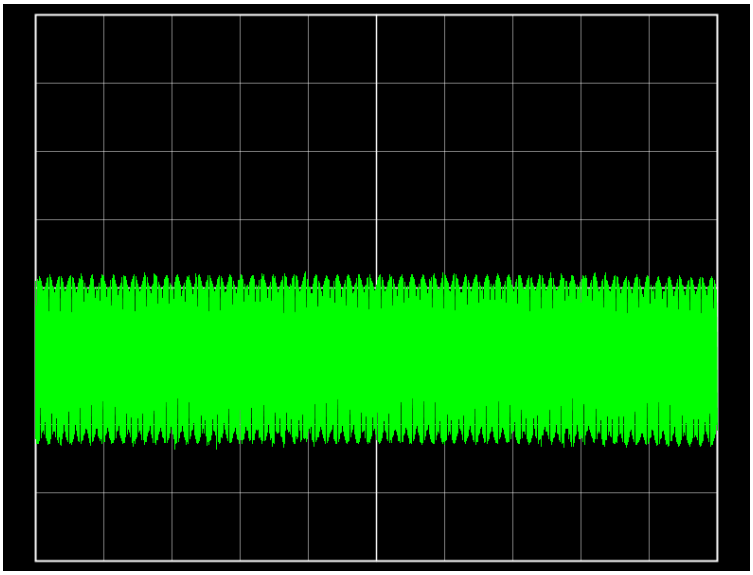


Figura 6.2.8. Datos sin demodular en reposo

La figura 6.2.8, muestra el caso de los datos sin demodular en un estado de reposo. Inicialmente vemos efectivamente que la señal se está recibiendo, pero no podemos extraer un valor cuantizado como puede ser la frecuencia o los niveles de tensión, pero esto lo podemos hacer a posteriori con el fichero generado.

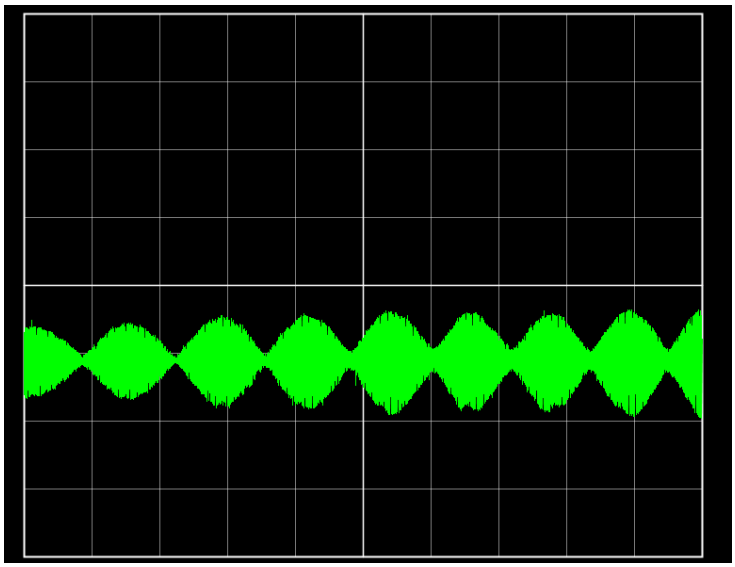


Figura 6.2.9. Datos sin demodular con reflexión total

Los datos de la figura 6.2.9, han sido generados acercando un objeto al sensor de forma perpendicular, lo que implica que el ángulo de reflexión es 0° y por lo tanto se produzca una modulación total, como vimos en el apartado de marco teórico de este documento.

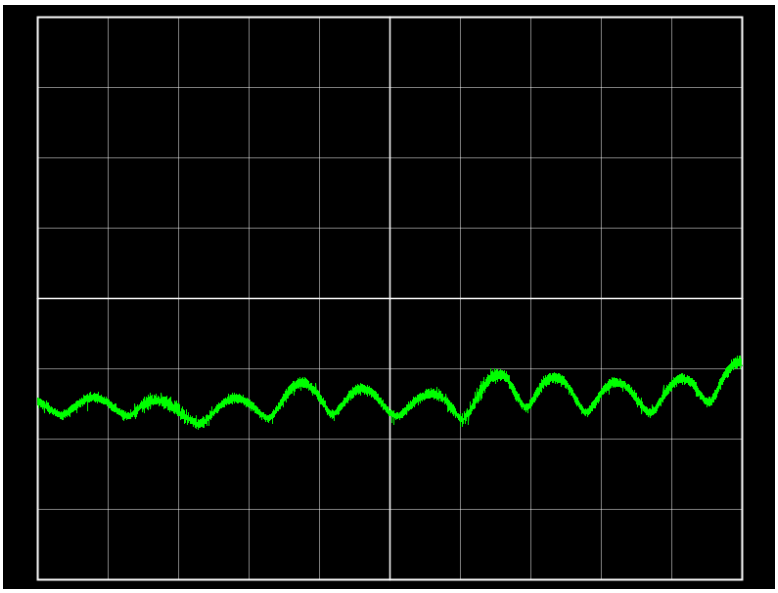


Figura 6.2.10. Datos demodulados

Esta última imagen (Figura 6.2.10), es un caso midiendo ya después de la demodulación. Se observa que efectivamente se recoge la envolvente de la señal, y que esta no posee una amplitud constante ni tampoco un nivel de continua fija.

6.3 Profundizando el análisis con Python

Ya hemos recogido una gran cantidad de datos suficientes para poder empezar a sacar ciertas conclusiones sobre el comportamiento de la onda. Pero todavía necesitamos cuantizar un poco los datos recogidos, poder observar su especto en frecuencia, niveles de tensión a incluso preparar los datos para introducirlos en un simulador como el LTSPICEVII.

En este caso, voy a estar usando Jupyter Noyebook para programar en Python.

Vamos a analizar primero los datos modulados con reflexión total obtenido anteriormente.

Primero llamaremos al módulo de "numpy" para poder manejar array y generar la FFT, y también usaremos "matplotlib" para representar graficas de los resultados. Así mismo, abriremos el fichero generador y crearemos un *array* con los datos:

```
#Importamos los modulos que usaremos
import numpy as np
import matplotlib.pyplot as plt


#Leemos el fichero y generamos un array con los datos
fichero = 'datosmodulados.txt'
with open(fichero) as f:
  lineas = f.readlines()
datos = []

for i in range(len(lineas)):
  datos.append(int(lineas[i]));

datos = np.array(datos)
print(datos[0:3], datos.size)
```

Posteriormente generamos un par de parámetros que nos serán útiles como la frecuencia de muestreo (107,14 kHz), el periodo de muestro, y el intervalo de tiempo:

```
#Establecemos la frecuancia de muestreo
fm= 9E6/(12.5+71.5)
Tm = 1/ fm
N = datos.size #numero de muestras
Tmax = Tm * N
t = np.arange(0,Tmax,Tm)
```

Ahora podemos representar los datos para comprobar que la señal guardada en el fichero es la misma.

```
#Representamos los datos para comprobar
plt.figure(figsize=(15, 5))
plt.plot(t,datos)
plt.legend(['Datos fichero'])
plt.xlabel("t(s)")
plt.ylabel("Dato")
```

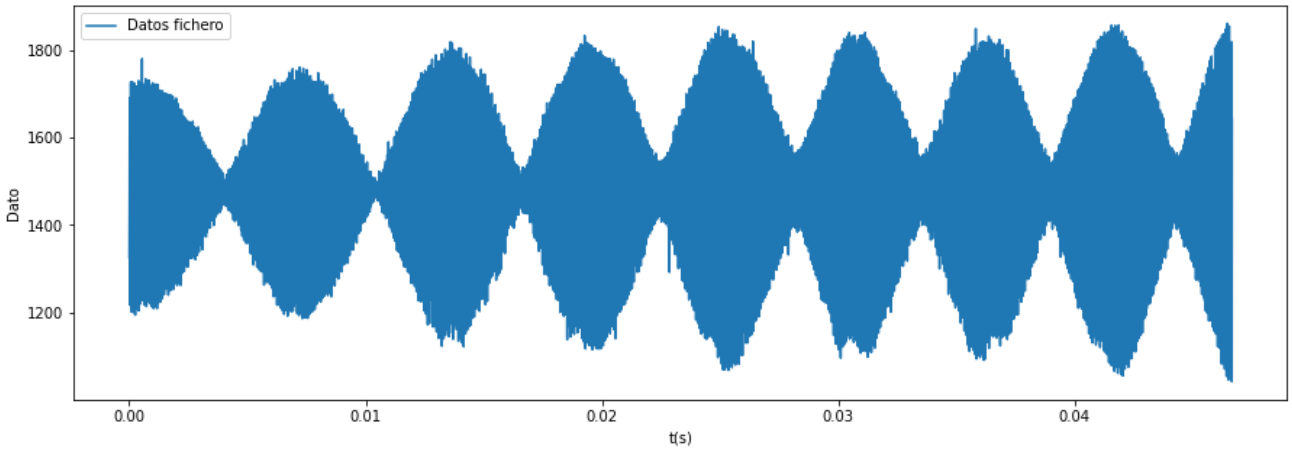


Figura 6.3.1. Datos sin demodular con Python

La figura 6.3.1, los datos del eje Y corresponden con los datos proporcionados por el convertidor ADC de 12 bits, con un rango de valores entre 0 y 4095.

Estos datos pueden ser analizados con la FFT aplicando la funcion de numpy “np.fft.fft”. Esta función devolverá un *array* con la FFT de los datos, pero de todos esos datos son solo válidos la primera mitad, pues la máxima información que se puede extraer es hasta la mitad de la frecuencia de muestreo (teorema de Nyquist), a partir de ahí se produce la misma señal simétrica a esa frecuencia en el caso de señales puramente reales. Por otro lado el array devuelto tendrá componentes reales y complejas, para magnitud y fase, pero a nosotros nos interesa solo la magnitud así que calculamos su módulo.

Creamos el código y representamos la gráfica en escala logarítmica:

```
#Habilitamos la interacción con la gráfica
%matplotlib notebook
#Generamos un array con las frecuencias a representar
f = np.arange(0, fm/2, fm/N)
FFT = (np.abs(np.fft.fft(datos/4096))[0:f.size])
plt.figure(figsize=(9, 6))
plt.loglog(f, FFT)
plt.xlabel("F(Hz)")
plt.ylabel("Datos FFT")
```

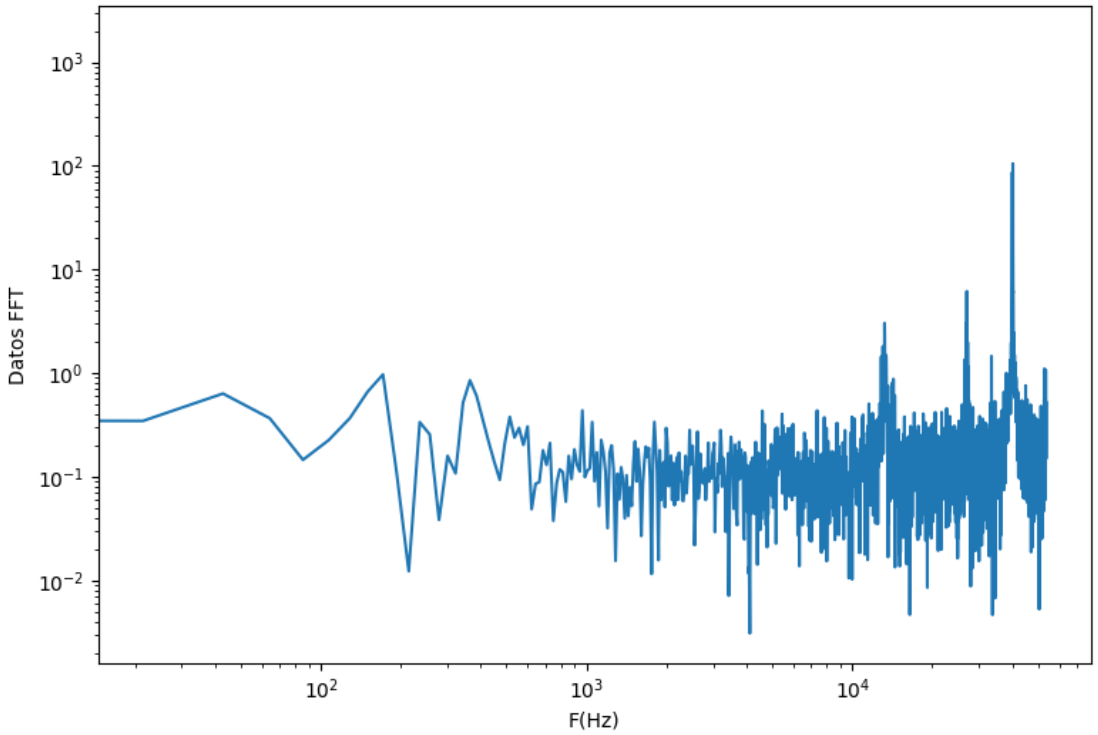


Figura 6.3.2. FFT datos modulados

La figura 6.3.2 nos muestra la FFT de la señal modulada. Se observa que el mayor pico está en la frecuencia de los 40 kHz. Podemos realizar un enfoque en esa zona. (Figura 6.3.3)

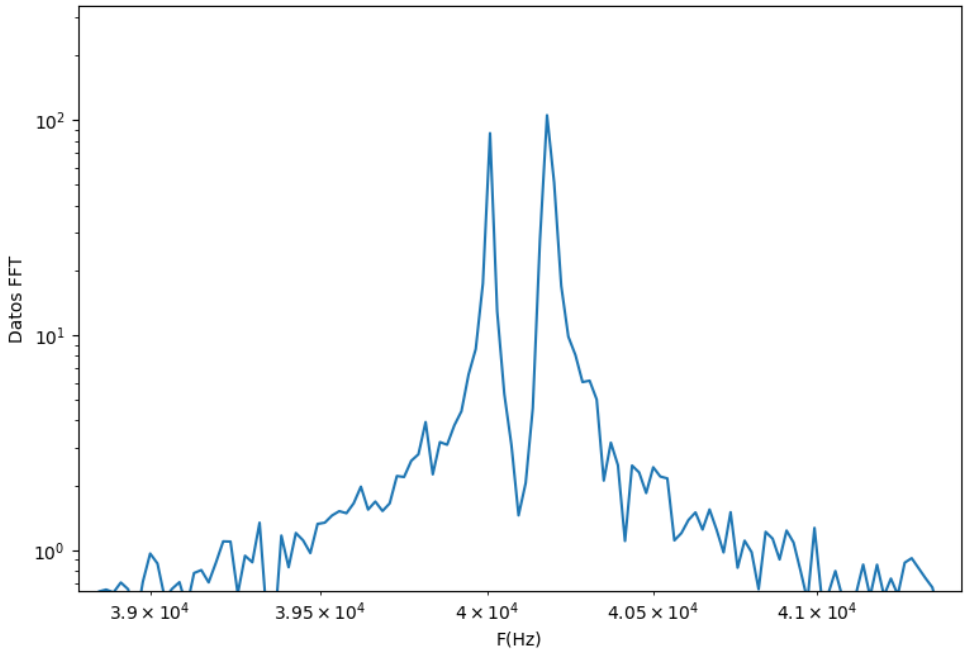


Figura 6.3.3. FFT datos modulados ampliada

Ahora que se ha amplio la figura, se ve con mayor detalle que realmente no es un pico de frecuencia sino dos. Este resultado se puede explicar por las características de la transformada en frecuencia de las señales continuas en invariantes en el tiempo. Dado que la señal recibida es producto de la modulación de la suma de dos señales sinusoidales, la transformada en

frecuencia de la señal, será igual a la suma de la transformada de las dos señales por el principio de superposición. [17]

Los dos picos son de 40000 Hz y de 40178 Hz, que corresponde con la señal emitida y la recibida respectivamente. Su diferencia es 178 Hz y equivale a la frecuencia *Doppler* y a la señal que se observa en la envolvente.

Podemos realizar el mismo análisis con los datos demodulados. Estos son los datos demodulados (Figura 6.3.4)

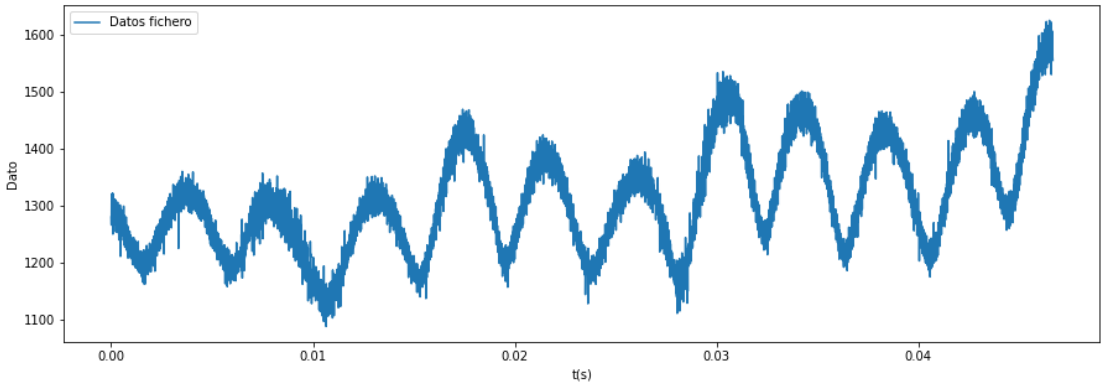


Figura 6.3.4. Datos demodulados Python

Aplicamos la FFT. (Figura 6.3.5)

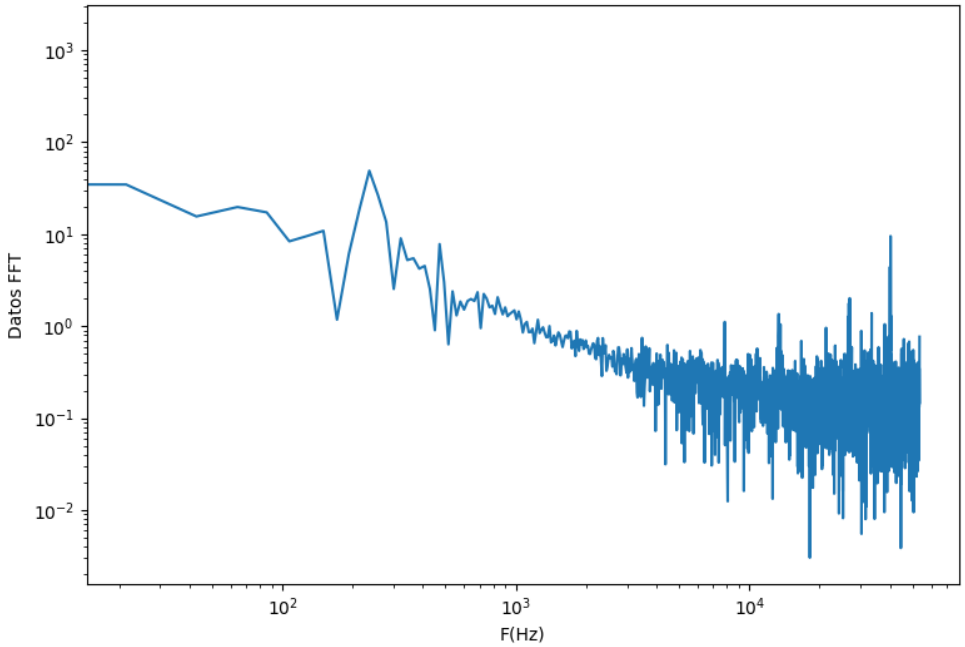


Figura 6.3.5. FFT datos demodulados

La componente de la señal portadora se ha reducido bastante, siendo el mayor pico ahora el localizado en la frecuencia de 235 Hz. Si ahora realizamos el mismo aumento en la banda de los 40 kHz (Figura 6.3.6), observaremos de nuevo los dos picos uno a 40000 Hz y otro a 40242 Hz. La diferencia es de 242 Hz muy próxima a los 235 Hz, con pequeñas diferencias que se pueden deber a la falta de resolución por cantidad insuficiente de datos.

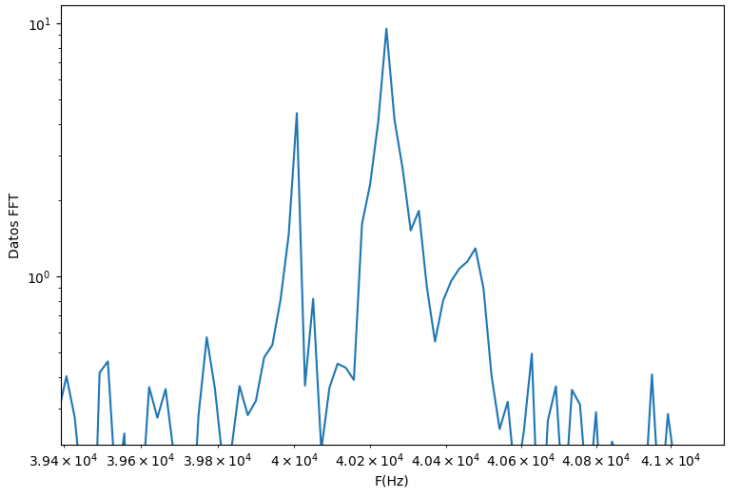


Figura 6.3.6. FFT datos demodulados aumentado

Esto demuestra y corrobora con los fundamentos teóricos a cerca de la demodulación AM y del detector de envolvente.

Por último, realizaremos en este código de Python una conversión de los datos y la generación de un fichero para crear luego una fuente PWL en LTSPICE que nos ayudara a probar varias configuraciones de circuito antes de su implementación práctica. Hay que escalar los datos dividiendo entre la resolución de 12 bits y multiplicando por 3,3 correspondiente al valor máximo medible por el ADC. Como los datos vienen de una alimentación de 5 V para adaptar el nivel de tensión se usó un partidor de tensión que divide a la mitad la tensión de entrada. Este cambio en la tensión se corrige aplicando un factor de 2 a los datos leídos. El código para esta implementación es el siguiente:

```
#Escribimos los datos en un fichero en formato
#Convertimos los datos con formato PWL de LTSPICE para usarlo luego
with open('datosPWL.txt', 'w') as writefile:
    for i in range(len(datos)):
      voltage = 2*3.3*(datos[i]/4096)
      writefile.write(str(t[i]) + " " + str(voltage) + "\n")
```

7. Diseño del circuito.

7.1 Circuito Pasa banda

Arduino dispone de entradas digitales y analógicas, ambas nos permiten capturar señales eléctricas, aunque la entrada digital solo permitirá señales cuadradas de un nivel de tensión de 0 a 5 V. La diferencia en ambos tipos de entradas además del tipo de señal radica en la velocidad, porque mientras la entrada analógica mide desde 112- 120 μs, la entrada digital mide cada 20 μs incluso, dispone de dos entradas de interrupción externa que pueden ser atendida en cualquier lugar del programa.

Esta característica de interrupción externa hace que sea preferible elegir esta entrada para medir la frecuencia de la señal recibida, sin embargo, requiere de diseñar e implementar un

circuito que permita convertir la señal demodulada en una señal cuadrada con la misma frecuencia.

Lo primero que realizaremos es amplificar y filtrar la señal demodulada, por medio de los amplificadores operacionales del circuito integrado LM324.

Hay que tener en cuenta un par de consideraciones en el diseño del filtro:

- Ancho de banda de la señal recibida:

  Este aspecto es simple y como ya se ha explicado depende de la velocidad y el ángulo siendo el caso contemplado con una frecuencia máxima de 1600 Hz

- Nivel mínimo de tensión pico a pico:

  La tensión de pico a pico depende entre otras cosas de la cantidad de señal reflejada, por lo que es un parámetro que no podemos definir con exactitud. En su lugar, vamos a suponer un caso desfavorable en el que necesitemos amplificar la señal unas 10 veces. Podemos amplificar mucho sin preocuparnos por saturación del amplificador, pues la distorsión de la onda no nos afecta a la frecuencia, además de que posteriormente será convertida a una señal cuadrada.

- Cantidad y característica del ruido en la medida.

  La cantidad y tipo de ruido depende de la superficie de reflexión. Una superficie muy irregular introducirá en la medida ruido, pero por otro lado una superficie muy lisa no reflejará correctamente las ondas emitidas. El caso más idóneo sería una superficie lisa, pero porosa que permita la reflexión.

  Del tipo de ruido del que más debemos preocuparnos es el de baja frecuencia, pues este cambia la cantidad de continua de la señal y producirá perdida de información al convertirlo en una señal cuadrada con un comparador con histéresis. Pero la baja frecuencia es importante cuando se quiere medir en baja velocidad.

La configuración del amplificador que más nos conviene es la de filtro pasa banda (Figura 7.1.1). Esta es la configuración típica de un filtro pasa banda, pero está pensado para una alimentación del operacional simétrica, y no para la alimentación de Arduino de 0 a 5V. Es por eso por lo que la señal resultante hay que añadirle un valor de continua a la mitad de la tensión máxima, 2,5 V. Usando como referencia el circuito que incorpora el HC-SR04 [15], lo que haremos es introducir en la entrada no-inversora la tensión de referencia de 2,5 V.

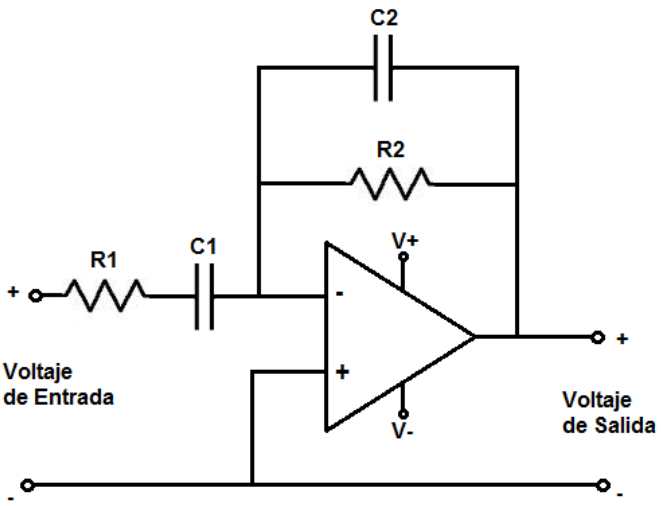


Figura 7.1.1. Filtro Pasa Banda

Realizando ese cambio en el operacional, sacaremos el modelo del circuito y las ecuaciones que lo definen para su diseño. El circuito de estudio es el siguiente (Figura 7.1.2).

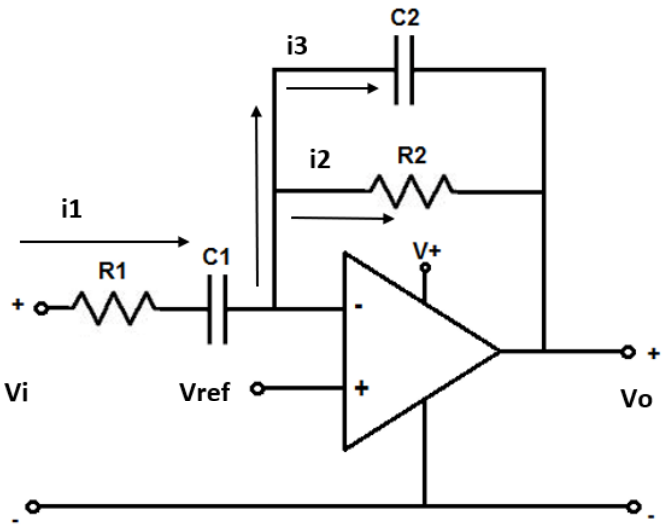


Figura 7.1.2. Circuito Filtro Pasa Banda

Para obtener las ecuaciones que rigen el comportamiento de este circuito, aplicaremos:

- La ley de ohm, la tensión entre los extremos de un conductor es proporcional a la intensidad que circular por el mismo. → $V = I \cdot R$
- Ley de nodos o primera ley de Kirchhoff, el sumatorio de todas las corrientes que pasan por un nudo es igual a 0 → $\sum_{k=1}^{n} i_k = 0$
- La impedancia de entrada de un operacional es alta, del orden de los 100 $M\Omega$, por lo que la corriente de entrada es despreciable → $I_- = I_+ = 0$
- Cuando un operacional se encuentra realimentado negativamente, se cumple que → $V_+ = V_- \rightarrow V_+ = Vref$
- La impedancia de un condensador depende de la frecuencia según la forma:
- $Z_C(\omega) = \frac{1}{j\omega \cdot C} \rightarrow Z_C(s) = \frac{1}{s \cdot C}$

Teniendo en cuenta las consideraciones anteriores, aplicamos análisis de transitorios y obtenemos las siguientes tres expresiones:

$$i_1 = \frac{Vi - Vref}{R1 + \frac{1}{s \cdot C1}}$$

$$i_2 = \frac{Vref - Vo}{R2}$$

$$i_3 = s \cdot (Vref - Vo) \cdot C2$$

La función del voltaje de salida se puede obtener despejando la siguiente expresión:

$$i_1 = i_2 + i_3 \rightarrow \frac{Vi - Vref}{R1 + \frac{1}{s \cdot C1}} = \frac{Vref - Vo}{R2} + s \cdot (Vref - Vo) \cdot C2 \quad (18)$$

Como la expresión anterior tiene muchos términos y no queremos producir fallos por una resolución inadecuada simplificaremos el proceso usando el programa wxMaxima.

Primero definimos las corrientes, luego generamos la expresión del nudo y con la función "solve" obtenemos el resultado. (Figura 7.1.3)

```
(%i1) i1:(Vi-Vref)/(R1+1/(s*C1));
      i2:(Vref-Vo)/R2;
      i3:(Vref-Vo)/(1/(s*C2));

(%i4) ec1: i1 = i2 + i3;
      VO:rhs(solve(ec1,Vo)[1]);
```

Figura 7.1.3. Expresiones máxima

El código anterior nos devuelve el siguiente resultado:

$$Vo = \frac{(s^2 Vref \cdot C1 \cdot R1 + s \cdot Vref \cdot C2 + (s \cdot Vref - s \cdot Vi)C1)R2 + s \cdot Vref \cdot C1 \cdot R1 + Vref}{(s^2 \cdot C1 \cdot C2 \cdot R1 + s \cdot C2)R2 + s \cdot C1 \cdot R1 + 1}$$

Para entender mejor la expresión anterior podemos aplicar una propiedad de los circuitos electrónicos, el principio de superposición y estudiar el resultado de la señal de continua y alterna como la suma de la contribución de cada una por separado.

En el caso de señales sinusoidales sabemos que su resultado se puede obtener realizando el cambio de $s = j\omega$. La contribución de la señal de continua se obtiene por tanto realizando $s = 0$ de la expresión anterior.

$$Vo_{continua} = Vref$$

La señal de continua a la salida es directamente el valor de Vref, que exactamente lo que buscamos.

Así mismo, como el valor de continua es Vref si anulamos este término nos quedará la expresión de la señal alterna, que al dividir por la entrada nos dará como resultado la ganancia:

$$H(s) = -\frac{s \cdot C1 \cdot R2}{(s \cdot C1 \cdot R1 + 1) \cdot (s \cdot C2 \cdot R2 + 1)} \quad (19)$$

La ganancia es negativa, indicando que el amplificador invierte la señal de entrada lo que no supone ningún problema en este proyecto.

La ganancia posee un cero en $s = 0$, en cambio posee dos polos:

$$p_1 \rightarrow s = \frac{1}{C1 \cdot R1} \rightarrow \omega_1 = \frac{1}{C1 \cdot R1} \rightarrow f_1 = \frac{1}{2\pi \cdot C1 \cdot R1} \quad (20)$$

$$p_2 \rightarrow s = \frac{1}{C2 \cdot R2} \rightarrow \omega_2 = \frac{1}{C2 \cdot R2} \rightarrow f_2 = \frac{1}{2\pi \cdot C2 \cdot R2} \quad (21)$$

La ganancia inicialmente será baja, e incrementará con la frecuencia a 20 dB/dec hasta llegar al primer polo con una ganancia máxima, tras alcanzar el segundo polo descenderá con la misma pendiente.

Esta ganancia máxima, se encuentra en la mitad del ancho de banda del filtro. Para su cálculo podemos suponer que el primer polo interviene mucho antes que el segundo → $p_1 \ll p_2$, y por lo tanto estudiar ganancia que tendría el filtro con frecuencia infinita.

$$H_{max} = \lim_{\omega \to \infty} \left| -\frac{j\omega \cdot C1 \cdot R2}{j\omega \cdot C1 \cdot R1 + 1} \right| \cong \lim_{\omega \to \infty} \frac{\cancel{j\omega \cdot C1} \cdot R2}{\cancel{j\omega \cdot C1} \cdot R1}$$

$$H_{max} = \frac{R2}{R1} \quad (22)$$

La relación de las resistencias define la ganancia del filtro (siempre que se cumpla $p_1 \ll p_2$), y el valor de C1 y C2 las frecuencias de corte. Los valores seleccionados fueron:

- $R2 = 10\ k\Omega$
- $R1 = 1\ k\Omega$
- $C2 = 10\ \text{nF}$
- $C1 = 2{,}2\ \mu F$

Estos valores dan como resultado una ganancia máxima de 10 y las siguientes frecuencias de corte:

$$f_1 = \frac{1}{2\pi \cdot C1 \cdot R1} = 72{,}34\ Hz$$

$$f_2 = \frac{1}{2\pi \cdot C1 \cdot R1} = 1591{,}55\ Hz$$

La primera frecuencia de corte se encuentra en 72 Hz, con incremento de 20 dB/dec desde la baja frecuencia hasta ganancia 10, implica que aproximadamente para una frecuencia de unos 7,2 Hz la ganancia es de amplitud 1, con menos frecuencia se atenuará la señal.

Actualmente desconocemos si este valor será adecuado para medir la velocidad mínima del robot, pero considerando que el ángulo de instalación será de 45°, si llegamos a medir 7,2 Hz la velocidad sería de 0,04 $\frac{m}{s}$, que es más que aceptable para la resolución del sensor.

La segunda frecuencia es de 1591 Hz y está elegida en base al ancho de banda. El diagrama de bode en amplitud del filtro queda de la siguiente forma (Figura 7.1.4).

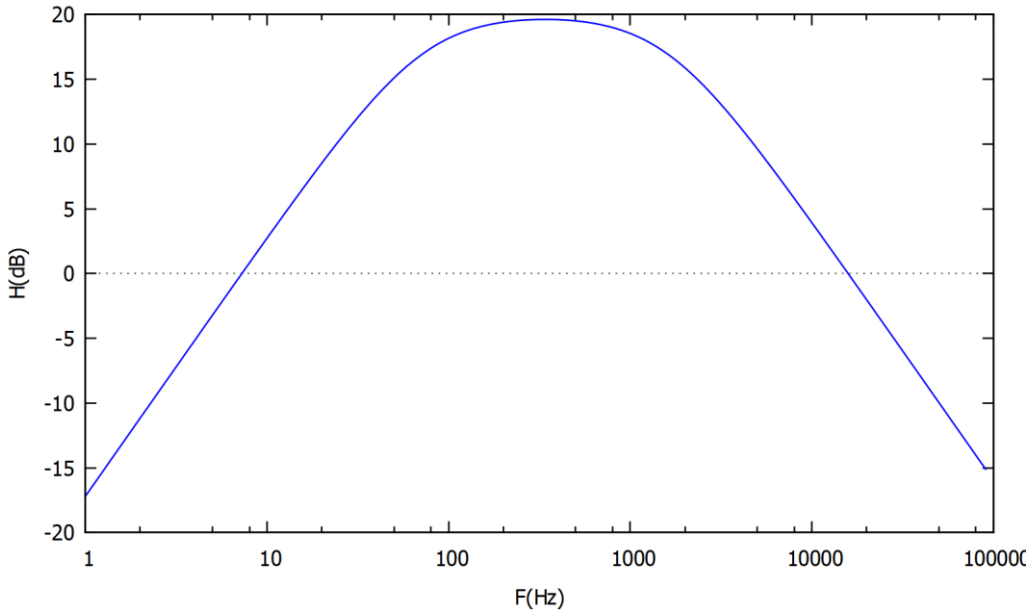


Figura 7.1.4. Diagrama de amplitud del filtro pasa banda

El valor de Vref será de 2,5 V y se generará dividiendo la tensión de 5 V con un partidor de tensión con dos resistencias de igual valor ($10\ k\Omega$).

La señal de entrada será la señal demodulada, pero a través de un amplificador operacional en configuración de seguidor de tensión (Figura 7.1.5). Esto garantiza una conexión con alta impedancia entre los dos circuitos para evitar afectar el comportamiento de estos.

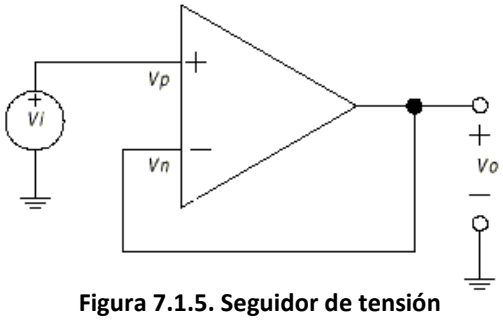


Figura 7.1.5. Seguidor de tensión

Podemos simular el filtro en LTSPICE con la señal capturada con el STM32 de la figura 6.2.10.

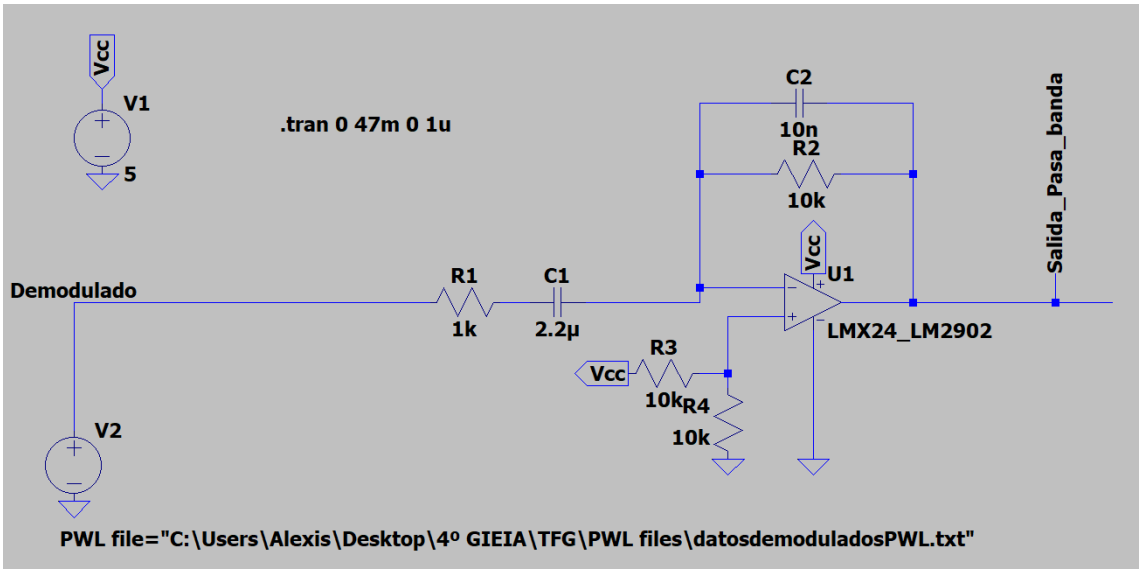


Figura 7.1.6. Circuito LTSPICE filtro pasa banda

En el caso de la simulación (Figura 7.1.6), se ha removido el seguidor de tensión pues no es necesario ya que el simulador fuerza la salida la señal en los valores indicados, y además aligera la simulación.

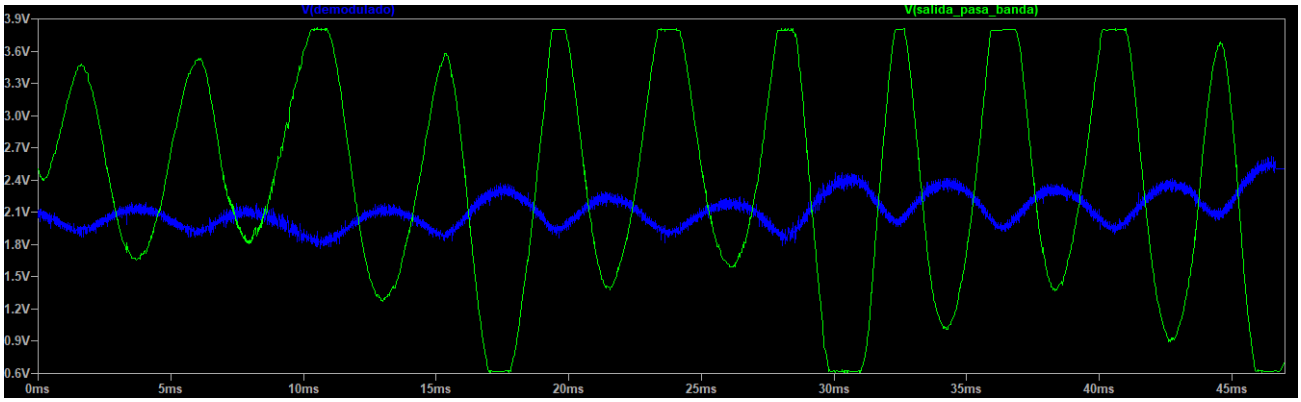


Figura 7.1.7. Simulación del filtro pasa banda

En la figura 7.1.7, se observa en primer lugar que efectivamente existe una inversión de la señal de entrada. La señal resultante ha aumentado su amplitud, hasta el punto de la saturación del

operacional en unos 3,8 V que está dentro del 1,5 V de *dropout* que establece el *datasheet* del LM324. (Véase anexo 14.6)

Por último, el nivel de continua esta próxima a 2,5 V con una desviación por la saturación producida, pero se cumple que la señal cruza por este nivel en los ciclos observados, importante para diseñar el siguiente circuito.

LTSPICE, también tiene la opción de hacer FFT, así que podemos ver el cambio de la señal antes y después del filtro. (Figura 7.1.8)

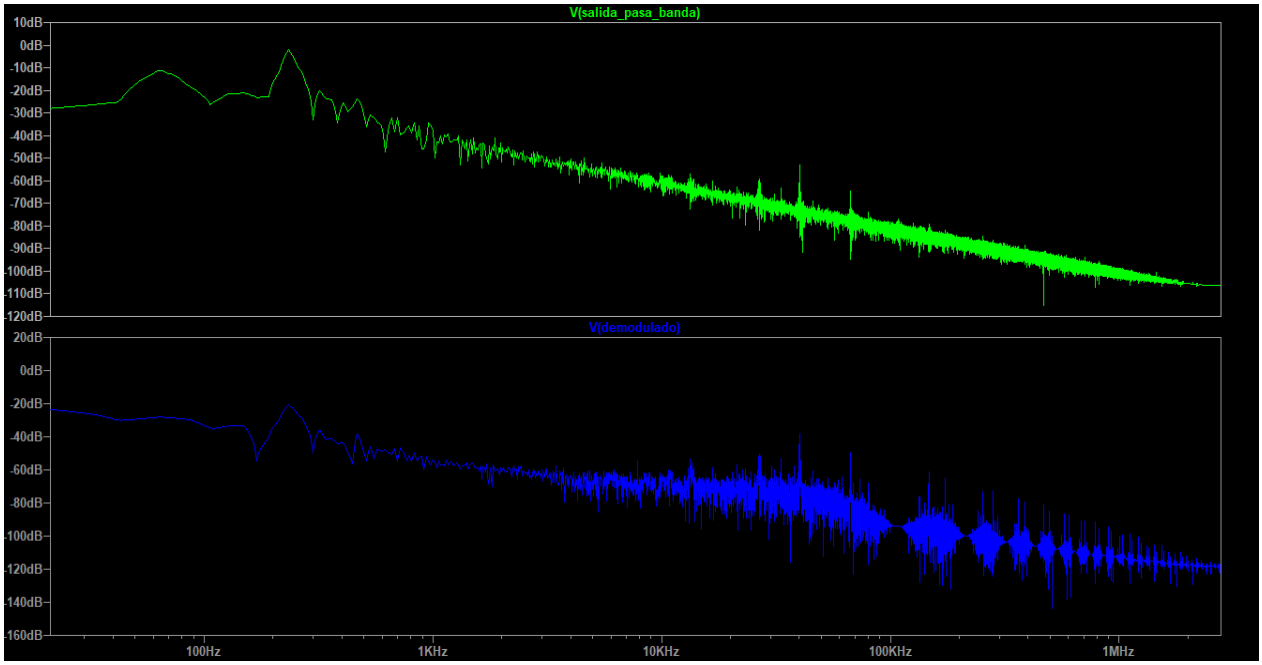


Figura 7.1.8. FFT pasa banda

La señal a salida del filtro ha logrado reducir prácticamente en su totalidad todos los armónicos procedentes de la señal portadora. Al mismo tiempo ha incrementado las frecuencias dentro del ancho de banda del filtro siendo la mayor de toda, la frecuencia de 235 Hz que habíamos obtenido con el Python. Pero también se incrementa una señal de baja frecuencia a unos 65 Hz, proceden del ruido generado por las variaciones de la superficie de reflexión. Si este ruido supera o tiene el mismo nivel que la frecuencia *Doppler,* introducirá errores en la medida, y tendremos que comprobar posteriormente si con el método de medida y las condiciones de trabajo, los datos recogidos serán fiables o no.

7.2 Generación de señal cuadrada

Para que la señal pueda ser leída por Arduino a través de una entrada digital con interrupción debe ser convertida a una señal cuadrada que conserve la misma frecuencia.

El circuito típico usado para convertir una señal analógica a una señal cuadrada es lo que se conoce como disparados de Schmitt, el cual consiste en un comparador con histéresis.

Un comparador con histéresis dispone de dos niveles de umbral, alto y bajo, esto hace el comparador más robusto ante le presencia de ruido en la señal. Este circuito también se realiza

con un amplificador operacional, pero en un caso en el que en vez de realimentación negativa se realiza realimentación positiva. (Figura 7.2.1)

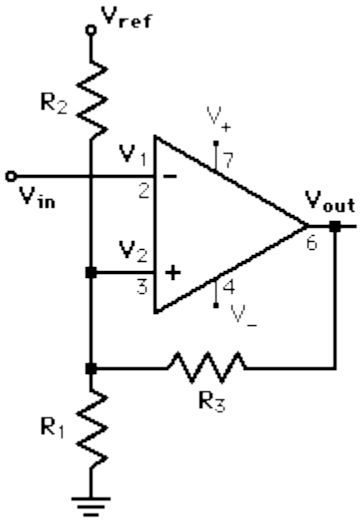


Figura 7.2.1. Disparador de Schmitt

Una forma de resolver este circuito es aplicar un sistema de ecuaciones en el nudo de V_2 con la aplicación de la primera ley de Kirchhoff:

$$\sum_{k=1}^{n} i_k = 0$$

$$\frac{V_2 - Vo}{R3} + \frac{V_2 - Vcc}{R1} + \frac{V_2}{R2} = 0 \rightarrow$$

$$V_2 = \frac{Vcc \cdot R2 \cdot R3 + Vo \cdot R1 \cdot R2}{(R2 + R1) \cdot R3 + R1 \cdot R2} \quad (23)$$

Como se puede ver el valor de la tensión en el nudo V_2 depende del valor de la salida del operacional. La expresión de la tensión de salida en un amplificador operacional es del modo:

$$Vo = A \cdot (V_+ - V_-) \quad (24)$$

Donde:

- A, es la ganancia del amplificador en lazo abierto.
- V_+, es la tensión en la entrada no inversora del operacional.
- V_-, es la tensión en la entrada inversora del operacional.

La ganancia de los amplificadores en lazo abierto es bastante grande del orden de los 10^5 o más, por lo que se suele considerar, que en lazo abierto el operacional trabaja entre dos estados que corresponden con la saturación inferior y superior del operacional. Por lo tanto, lo que define este estado es el signo de la diferencia de $V_+ - V_-$, si es positivo la tensión de salida será de Vsat, y si es negativa 0 voltios en este caso.

Esta no linealidad generada por la saturación hacer que para poder resolver la ecuación anterior debamos suponer primero unos valores y estudiar el comportamiento al incrementar el voltaje de entrada. Podemos suponer primero que la tensión a la salida inicialmente está en 0 V, y la tensión de entrada también.

$$V_{in} = V_-$$

$$Vo = A \cdot (V_+ - V_-) = A \cdot \frac{Vcc \cdot R2 \cdot R3}{(R2 + R1) \cdot R3 + R1 \cdot R2} > 0$$

Como el resultado es mayor que cero, en este caso Vo no será 0 voltios si no Vsat. Además, este valor será así hasta que $V_- > V_+$, es decir que el umbral del cambio coincide con el valor de V_+.

Existen dos valores de umbral que corresponde cuando la salida es Vsat o 0 V, y los podemos obtener con la ecuación de V_+:

$$VH = \frac{Vcc \cdot R2 \cdot R3 + Vsat \cdot R1 \cdot R2}{(R2 + R1) \cdot R3 + R1 \cdot R2} \quad (25)$$

$$VL = \frac{Vcc \cdot R2 \cdot R3}{(R2 + R1) \cdot R3 + R1 \cdot R2} \quad (26)$$

El umbral superior es en caso de $Vo = Vsat$, pues contribuye a aumentar la tensión de V_+.

De las ecuaciones anteriores queremos determinar los valores de las resistencias para poder establecer los niveles de umbral deseado, así que podemos despejar los valores de R2 y R3:

$$R3 = -\frac{Vsat \cdot R1 \cdot VL}{Vcc \cdot VL - Vcc \cdot VH} \quad (27)$$

$$R2 = -\frac{Vsat \cdot R1 \cdot VL}{(Vsat - Vcc) \cdot VL + Vcc \cdot VH - Vcc \cdot Vsat} \quad (28)$$

Tanto Vcc como $Vsat$ son valores conocidos, VL y VH son los umbrales que establecemos, y $R1$ lo podemos elegir arbitrariamente. Hemos elegido los valores de los umbrales para que estén centrados en los 2,5V y tenga un rango de 0,4 V:

- $Vcc = 5\ V$
- $Vsat = 3{,}8\ V$
- $VH = 2{,}7\ V$
- $VL = 2{,}3\ V$
- $R1 = 1\ k\Omega$
- $R2 = 1\ k\Omega$
- $R3 = 4{,}3\ k\Omega$

Aunque el valor de R3 este normalizado y exista en el mercado debido a que no es valor demasiado utilizado se optó por sustituirla por dos resistencias en paralelo de $4{,}7\ k\Omega$ y $47\ k\Omega$, que dan como resultado el mismo valor. El paralelo de dos resistencias se calcula como:

$$R_1 // R_2 = \frac{R_1 \cdot R_2}{R_1 + R_2} = \frac{4{,}7 \cdot 47}{4{,}7 + 47} = 4{,}27\ k\Omega$$

Con la siguiente gráfica se muestra en mayor detalle el comportamiento de histéresis del circuito. (Figura 7.2.1)

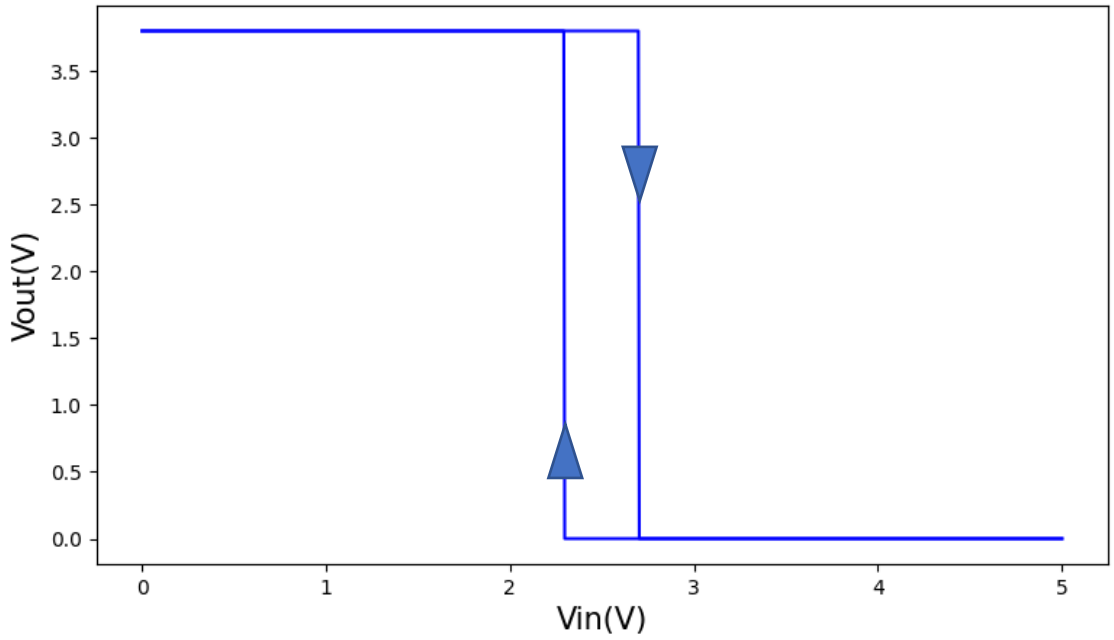


Figura 7.2.1. Comparador con histéresis diseñado

Se observa que la salida presenta un comportamiento contrario a la variación de la entrada, es decir, la salida se encuentra invertida a la entrada. Como esta etapa se conecta después del filtro pasa banda, la señal se invertirá dos veces desde la señal demodulada, por lo que la señal final no presentará cambio de fase, a excepción de la añadida por las frecuencias de corte del filtro.

7.3 Montaje y comprobación del circuito

El comportamiento del circuito esperado para una determinada señal, en las distintas etapas es la que se muestra en la siguiente simulación. (Figura 7.3.1)

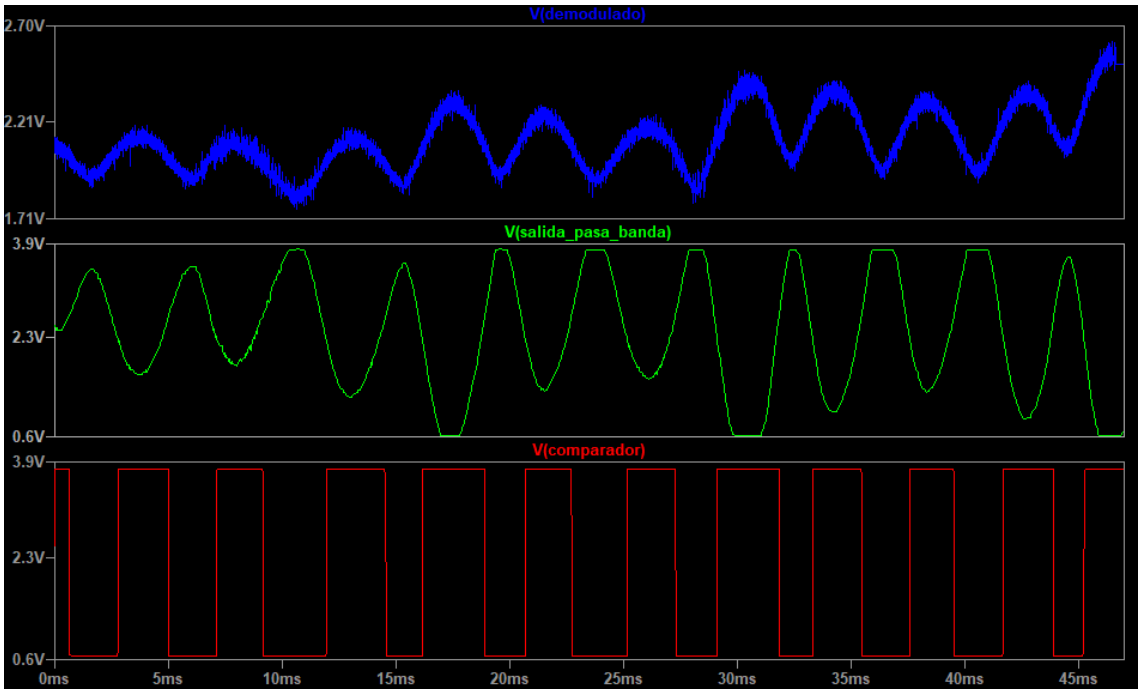


Figura 7.3.1. Simulación de las distintas etapas

El circuito se montará sobre una *protoboard*, y se usará el circuito LM324 el cual incluye 4 operacionales. (Figura 7.3.2)

Este circuito integrado se puede alimentar con una tensión de hasta 32 Voltios, y tiene un ancho de banda de 1 MHz para ganancia unitaria (GBW). Esto implica que, para una frecuencia de 1600 Hz, puede llegar a amplificar hasta 625. Cumpliendo con estas dos características el LM324 es válido para nuestro proyecto.

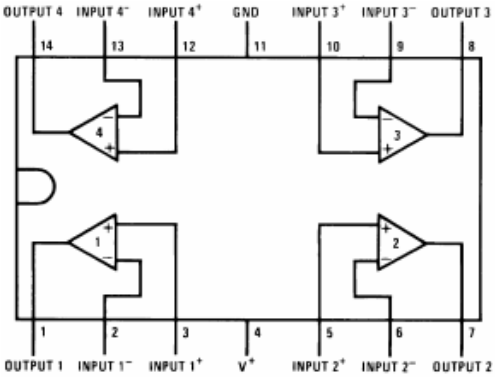


Figura 7.3.2. LM324 diagrama

Montado el circuito y realizando varias pruebas de medida con la lijadora los resultados se han obtenido y medido con el osciloscopio:

En la figura 7.3.3, se muestra la señal recibida y la señal demodulada simultáneamente. Como se puede comprobar, efectivamente el demodulador logra recoger la envolvente de la señal original. La señal resultante presenta algo de rizado, pero veamos cómo afecta esto a la siguiente etapa de filtrado.

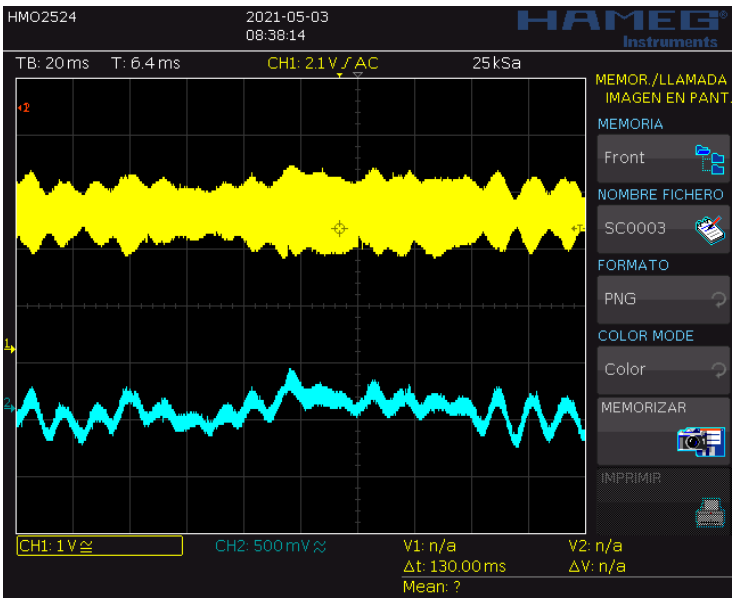


Figura 7.3.3. Señal recibida y señal demodulada

En la figura 7.3.4, se muestra la etapa de filtrado. Ambas escalas se encuentran en 1 V, y se observa como la señal ha sido amplificada en gran medida, al mismo tiempo que se ha invertido. La señal resultante presenta bajo ruido, lo que no deberá afectar al disparador de Schmitt.

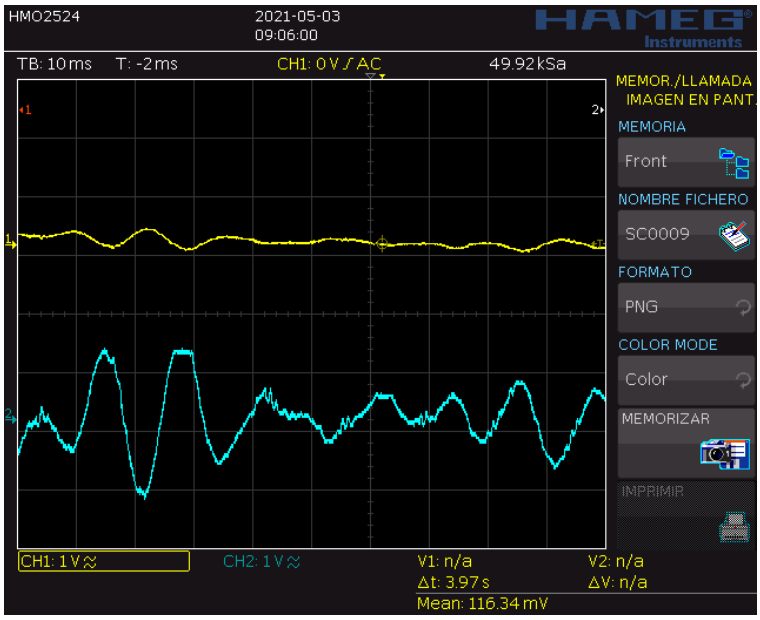


Figura 7.3.4. Señal demodulada y filtrada

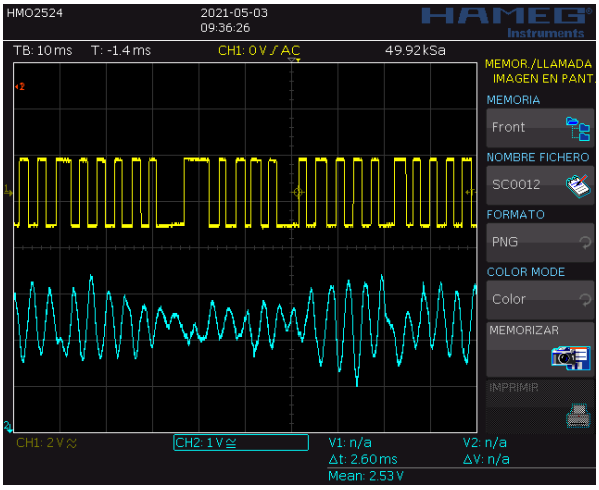


Figura 7.3.5. Salida comparador y filtro

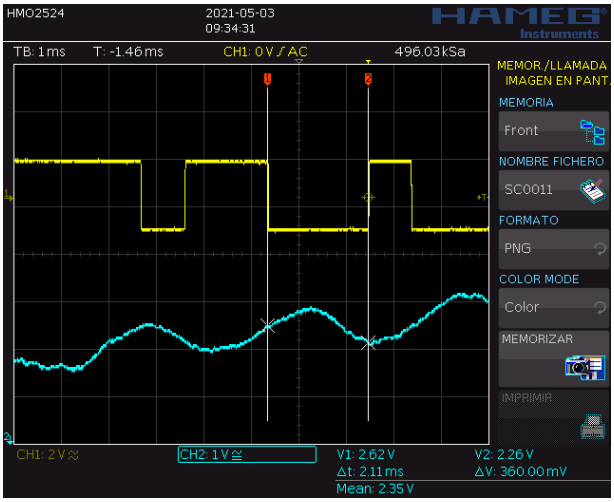


Figura 7.3.6. Umbrales del comparador real

En la figura 7.3.5 y 7.3.6 se muestra el resultado del comparado con histéresis y la salida del filtro pasa banda. La señal es convertida en su mayoría en una señal cuadrada. Los umbrales de disparo del circuito real son de 2,62 V y de 2,26 V, desviándose un poco de los teóricos a causa de las tolerancias de las resistencias, pero las desviaciones entran dentro de un margen tolerable.

8. Medición de la frecuencia.

La señal cuadrada generada en la última etapa de salida se conectará a Arduino a través del pin digital 2, que corresponde con la interrupción externa 0. Para las pruebas realizadas en este apartado utilizaremos otro Arduino para no tener que estar modificando el código del generador de pulsos de 40 kHz.

Para la medición de la frecuencia, se ha optado por usar dos métodos. El primer método es midiendo el periodo de la señal y realizando su inversa, y el segundo método consiste en medir el número de pulsos en un intervalo de tiempo fijo. Cada uno de estos métodos tienen sus ventajas y desventajas, y pueden ser mejor o peor opción según sea el caso en el que se aplican.

8.1 Método de pulsos.

Este método consiste en contar el número de pulsos en un intervalo de tiempo determinado. La forma de medir los pulsos se realizará cada vez que se produzca un cambio en la señal cuadrada, ya sea en un flanco de subida, en un flanco de bajada o en ambos a la vez (ténganse en cuenta que en este caso el número de pulsos es el doble).

Desarrollamos las ecuaciones necesarias para calcular la frecuencia con este método:

$$f_{señal} = \frac{N}{T_{medida}} \quad (29)$$

Donde:

- $f_{señal}$, es la frecuencia de la señal en Hz.
- T_{medida}, es el intervalo de medida de los pulsos en segundos, que la tomaremos fija.
- N, es el número de pulsos medidos en flanco de subidas o bajadas.

Ténganse en cuenta que, si se mide el pulso en los cambios de la señal tanto de subida como de bajada, la expresión anterior debe de dividirse entre 2. De hecho, usaremos esta forma de medir porque más puntos implica mayor resolución como veremos más adelante.

Por otro lado, el intervalo de medida también define la resolución de forma que si el intervalo aumenta la resolución también aumenta.

Para ver cómo afecta el intervalo de medida y el número de pulsos, estudiaremos en otro apartado el error sistemático por la precisión del sistema de medida. Calcularemos el error al usar la siguiente expresión:

$$f_{señal} = \frac{N}{2 \cdot T_{medida}} \quad (30)$$

8.1 Método del periodo.

Este método consiste en medir el periodo de la señal cuadrada. Para ello, se mide el tiempo que pasa entre los dos puntos a partir de que se repita la onda.

Al igual que con el método de pulsos, la señal se medirá en el intervalo en el que se produzcan dos flancos de subida o de bajada. También se puede medir el intervalo de medio periodo con

flancos en el cambio de la señal y multiplicar su valor por 2, pero esto solo es válido si el tiempo en alto y en bajo de la señal es el mismo.

Este método obtiene directamente la frecuencia de la onda con la inversa del periodo medido:

$$f_{señal} = \frac{1}{T_{medida}} \quad (31)$$

Ténganse en cuenta de que en caso de trabajar midiendo la mitad del periodo, su valor debe multiplicarse por 2. Esta será la forma que usaremos para calcular la frecuencia con este método, suponiendo que los anchos de pulsos son iguales, pues como veremos luego con la estimación del error y en las pruebas realizadas, esta manera nos conviene más y no será el problema principal de este método. La expresión que utilizaremos entonces es la siguiente:

$$f_{señal} = \frac{1}{2 \cdot T_{medida}} \quad (32)$$

8.3 Error sistemático de la frecuencia.

El error sistemático, es el error que se tiene por el sistema de medida utilizado, por la propia precisión del instrumento de medida o por el proceso de medición. [18]

Por ejemplo, en el caso de los instrumentos de medida el error de la precisión se toma como la unidad mínima medible (la resolución), y en ocasiones se toma la mitad de este valor en el caso de instrumentos analógicos.

Para calcular como se propaga el error en la medida de la frecuencia podemos aplicar la expresión de propagación de error para una variable:

$$\Delta Y = \left|\frac{\partial f(X)}{\partial X}\right| \cdot \Delta X \quad (33)$$

Donde:

- ΔY, es el error absoluto de la expresión calculada.
- ΔX, es el error de la variable independiente.
- $f(X)$, es la función que determina el valor de Y en función de X.

Comencemos primero con el método de pulsos. En este caso como nuestra medida es el número de pulsos, la resolución será de $\pm$ 1 pulso. El error de la frecuencia será, por lo tanto:

$$f_{señal}(N) = \frac{N}{2 \cdot T_{medida}} \rightarrow$$

$$\Delta f = \left|\frac{\partial f_{señal}(N)}{\partial N}\right| \cdot \Delta N = \frac{1}{2 \cdot T_{medida}} \quad (34)$$

Fíjese que el error es constante y es inversamente proporcional al periodo de medida. De hecho, si no midiéramos la señal en cada cambio, sino en solo flancos de subida o bajada el error sería el doble.

El caso es que el error absoluto no nos dice mucho, porque no es lo mismo un error de 10 Hz en una medida de 20 Hz, que en una de 200 Hz. Por eso, nos interesa estudiar más el error relativo y estudiar el porcentaje de error de medida.

La expresión del error relativo de este método es el siguiente:

$$E_r = \frac{E_{absoluto}}{Valor_{medido}} \rightarrow$$

$$E_r = \frac{\frac{1}{\cancel{2 \cdot T_{medida}}}}{\frac{N}{\cancel{2 \cdot T_{medida}}}} = \frac{1}{N} \rightarrow E_r = \frac{100}{N}\ [\%]\ \ (35)$$

Se observa que el error relativo es inversamente proporcional al número de pulsos (Figura 8.3.1), es decir cuantos más pulsos se midan menor el error cometido. Otro detalle es que el intervalo de medida no influye en el error, sin embargo, al usar la medida en el cambio implica mayor número de puntos en el mismo intervalo, por lo que es mucho mejor.

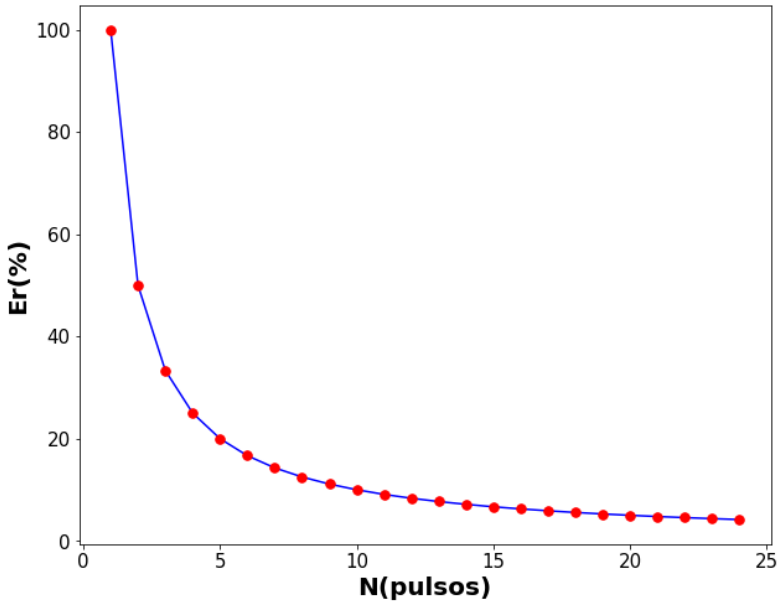


Figura 8.3.1. Error relativo de los pulsos

Ahora analicemos el error del método del periodo. Obtengamos primero el error absoluto. En este caso el error de medida por precisión del aparato viene por la función "*micros*" de Arduino, la cual tiene una resolución de 4 μs.

$$f_{señal}(T) = \frac{1}{2 \cdot T} \rightarrow$$

$$\Delta f = \left|\frac{\partial f_{señal}(T)}{\partial T}\right| \cdot \Delta T = \frac{1}{2 \cdot T^2} \cdot 4 \cdot 10^{-6}\ \ (36)$$

En este caso el erro absoluto aumenta su el periodo disminuye, es decir que, a mayor frecuencia, pero. Como en el caso anterior analizaremos el error relativo:

$$E_r = \frac{E_{absoluto}}{Valor_{medido}} \rightarrow$$

$$E_r = \frac{\frac{1}{2 \cdot T^2} \cdot 4 \cdot 10^{-6}}{\frac{1}{2 \cdot T}} = \frac{4 \cdot 10^{-6}}{T} \rightarrow E_r = \frac{4 \cdot 10^{-4}}{T} \ [\%] \quad (37)$$

El error relativo tiene un comportamiento inverso al método de pulsos. en cuanto a la medida de frecuencia (Figura 8.3.2).

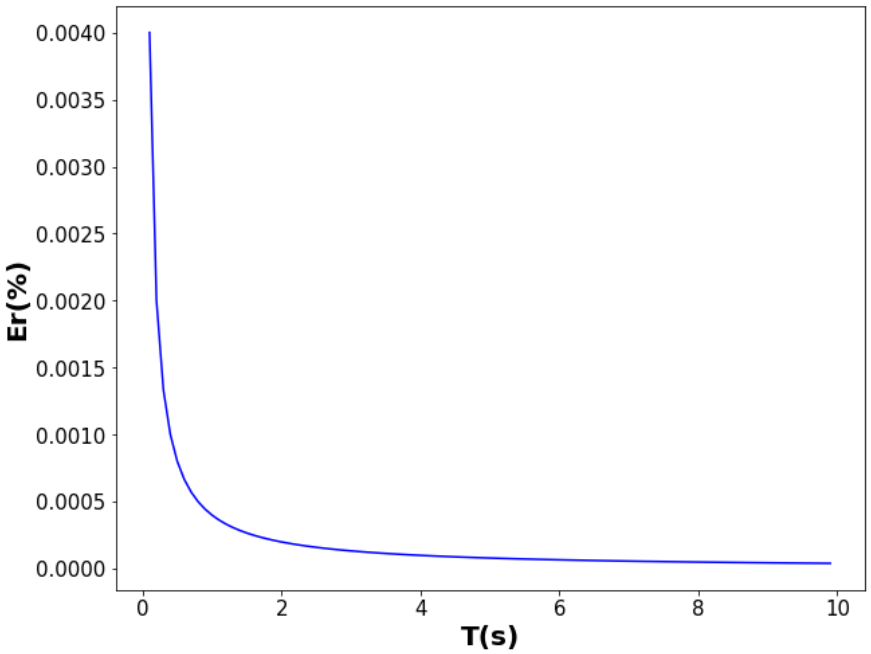


Figura 8.3.2. Error relativo del periodo

Como la resolución de la medida del periodo es tan alta, parece que el método del periodo tiene más precisión, pero hay que tener en cuenta que estamos suponiendo que el ancho de pulsos de un periodo es el mismo, lo cual en un sistema perfecto con velocidad constante debería cumplirse, pero todos sabemos que los sistemas reales poseen variables que no podemos controlar. Estas variables introducirán en un nuestro sistema componentes que nos produzcan un error aleatorio. Probaremos ambos métodos de medida para ver las ventajas y desventajas de cada uno y observar cuál de los dos métodos es más robusto al ruido.

9. Implementación del receptor

En la fase de pruebas del receptor estaremos usando un Arduino distinto al de emisión para no modificar su código. En este programa estaremos agregando además de los métodos descritos, un enconder que basado en el método de los pulsos nos determinará la velocidad real de la cinta para comprobar la eficacia del sensor. El *encoder* hace contacto con la cinta por medio de una rueda de 3 cm de diámetro y estará conectado a Arduino a través de su segundo pin de interrupción que estará programada para detectar los flancos de bajada.

9.1 Código del método de pulsos.

Pasamos a realizar el código del método de pulsos. Primero inicializamos las variables, y en el *Void Setup* inicializamos la comunicación serial a 115200 baudios para mostrar los datos recogido. También definimos los pines como entrada con *Pull-Up*, y declaramos las interrupciones externas. (Figura 9.1.1)

```
//Definimos estas variables globales, que se deben ejecutar en varias funciones
unsigned long pulso = 0;
unsigned int pulsoencoder = 0;

//En este caso, está variable es necesaria inicializarla antes de la funcion loop
unsigned long t = 0;


void setup() {

 Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);
}
```

Figura 9.1.1. Código inicialización método de pulsos

Para que esté código funcione debemos de definir las funciones de las interrupciones. En estas funciones incrementaremos el contador de los pulsos cada vez que sean detectados. (Figura 9.1.2)

```
//Definimos la interrupcion de los pulsos del sensor
void interrupcion() {
  //incrementamos el contador
  pulso++;


}

//Definimos la interrupcion de los pulsos del encoder
void encoder() {
  //incrementamos el contador
  pulsoencoder++;
}
```

Figura 9.1.2. Código interrupción método de pulsos

Por último, realizamos el código del *Void Loop*. (Figura 9.1.3)

```
void loop() {
  //Tomamos el tiempo actual en microsegundos
  unsigned long tl = micros();

  //Calculamos la velocidad cada 200 ms
  if((tl - t) > 200E3){
    unsigned int pulso_medido = 0;
    unsigned int pulsoencoder_medido = 0;
    //Desactivamos las interupciones
    noInterrupts();
      pulso_medido= pulso;
      pulsoencoder_medido = pulsoencoder;
      pulso = 0;
      pulsoencoder = 0;
      //Copiamos las variables y las reseteamos para que siga contando
    interrupts();
    //Definimos el intervalo de medida y reseteamos el intervalo de tiempo
    unsigned long    Tmedida = tl -t;
    t = micros();
    float   f = ((float)pulso_medido*1E6)/(2*(float)Tmedida);
    //Calculamos la frecuencia del sensor

    float fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
    float v = 3.6*340*f/(2*40E3*0.707);
    float vencoder = 3.6*fencoder*0.03*3.14159;
    //Calculamos la frecuencia del encoder

    Serial.print(pulso_medido);
    Serial.print(",");
    Serial.print(v);
    Serial.print(",");
    Serial.println(vencoder);

  }

}
```

Figura 9.1.2. *Void Loop* método de pulsos

Para medir el intervalo de tiempo usamos la función “micros”, que devuelve el número de microsegundos desde el comienzo del programa. Para hacer esta prueba hemos establecido un intervalo de 200 ms. El cálculo de la frecuencia será conforme la ecuación que hemos decidido. El ángulo establecido es de 45°, por lo que en la expresión hemos realizado la siguiente aproximación porque es más fácil de operar:

$$\cos(45°) \approx 0{,}707$$

La velocidad del *encoder* se determina, teniendo en cuenta que una revolución son 200 pulsos, y sabiendo que el diámetro de la rueda es de 3 cm, la velocidad se obtiene como el producto del número de revoluciones por la circunferencia de la rueda.

Tanto la velocidad del *encoder* como la de pulsos se han mostrado en este caso en $\frac{km}{h}$, que, a pesar de no ser la unidad del sistema internacional, la hemos usado para las pruebas por dar un valor más grande que los $\frac{m}{s}$.

Se mostrará a través del monitor serial de Arduino los datos de los pulsos, la velocidad medida del sensor y del *encoder.*

Tras realizar el código realizamos varias pruebas y guardaremos los datos para su posterior análisis.

9.2 Código del método de pulsos.

Este método es similar al anterior en cuanto al código de inicialización. Pero en este caso hemos añadido un par de variables globales más que nos servirán más adelante. (Figura 9.2.1)

```
#define N 20
//Definimos las variables globales
byte pulsorecibido = 0;
unsigned int pulsoencoder = 0;

//Se inicializan las variables que deben tener un valor definido antes de la funcion void loop
unsigned long t = 0;
unsigned long t2 = 0;
unsigned long T[N];
byte j = 0;
float fencoder = 0;

void setup() {

  Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);
}
```

Figura 9.2.1. Inicialización método del periodo

Hemos definido en este caso un *buffer* de datos de N valores. Este *buffer* se irá escribiendo con los nuevos datos del periodo medido, luego se realizará la media aritmética de estos valores y el resultado es el periodo que utilizaremos para el cálculo de la velocidad. El motivo de esto, los veremos más adelante, pero en resumen se debe a la naturaleza ruidosa de este método.

Definimos las funciones de la interrupción (Figura 9.2.2). Ahora la función de interrupción del sensor estableceremos una variable en 1 para saber que la interrupción ha ocurrido, y en función de este valor calcularemos el periodo en el *Void Loop*.

```
//Función de interrupción del sensor
void interrupcion() {
  pulsorecibido = 1;

}

//Función de interrupciones del encoder
void encoder() {
  pulsoencoder++;

}
```

Figura 9.2.2. Interrupciones método del periodo

Para el *Void Loop* hay que tener en cuenta que ahora tenemos por un lado el método de pulsos por el *encoder* y el método del periodo. El *encoder* tiene el mismo código, pero cambiando el intervalo de tiempo. (Figura 9.2.3)

```
unsigned long t1 = micros();
//Tomamos las velocidad del encoder cada 100 ms
if(t1-t2>100E3){
  float pulsoencoder_medido = 0;

  noInterrupts();
   pulsoencoder_medido = pulsoencoder;
   pulsoencoder = 0;
   //Copiamos el numero de pulsos y restablecemos la variable para que pueda seguir contando
  interrupts();
  unsigned long    Tmedida =( t1 - t2);
  t2 = micros();
  fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
  //Calculamos la frecuencia del encoder (revoluciones por segundo)
```

Figura 9.2.3. *Encoder* método del periodo

El cálculo de la frecuencia por el método del periodo se realizará cuando se detecte la interrupción. Se calcula el nuevo valor del periodo y se guarda en el *buffer* de forma cíclica sobrescribiendo siempre el valor más antiguo. Después se realiza la media de los N valores y con ese valor se calcula la frecuencia real. (Figura 9.2.4)

```
if(pulsorecibido){

   unsigned long    Tmedida =( t1 - t);
   //Calculamos el periódo

   t = micros();
   //Como estamos midiendo en los cambios el periódo medido es la mitad
   //Guardamos el dato en un array que dispone de los N valores anteriores
   //Guardamos de froma ciclica segun el indice j, de modo que siempre guardamos
   //el nuevo dato, en la posición del dato más antiguo
   if(j <= (N-1)) T[j] = 2*Tmedida;

   unsigned long Ttotal = 0;

   for(int i = 0; i < N; i++){
    Ttotal += T[i];
   }
   //Calculamos la media de todos los periódos

   float Tmedia = (float)(Ttotal)/N;
   float f = ((float)1E6)/(Tmedia);
   float v = 3.6*340*f/(2*40E3*0.707);
   j++;
   if(j>N){
    //Cuando el valor de conteo supera el valor de N, se muestran los datos
    //Hay que tener en cuenta que solo se mostraran los datos al tomar N muestras
    //Además, las función Serial.print lleva un tiempo de ejecucion más alto,
    //por lo que las medidas en alta frecuencia se van a ver más afectadas
   float vencoder = 3.6*fencoder*0.03*3.14159;
    Serial.print(v);
     Serial.print(",");
    Serial.println(vencoder);
    j = 0;
   }
    pulsorecibido = 0;


}
```

Figura 9.2.4. Calculo con el método del periodo

En este caso hemos mostrado los datos cada vez que se obtengan N valores nuevos, el motivo de esto es porque la función “Serial”, lleva cierto tiempo de ejecución en función de la longitud de los datos, por lo que llamarla puede afectar al periodo medido si este es menor que el tiempo de ejecución.

9.3 Análisis y resultado de los dos métodos.

Ahora analizaremos los datos obtenidos con ambos métodos para estudiar las ventajas y desventajas de cada uno, y valorar cual nos dará mejores medidas.

Empecemos pues con el método de los pulsos. En este apartado usaremos el Excel por disponer de algunas herramientas estadísticas.

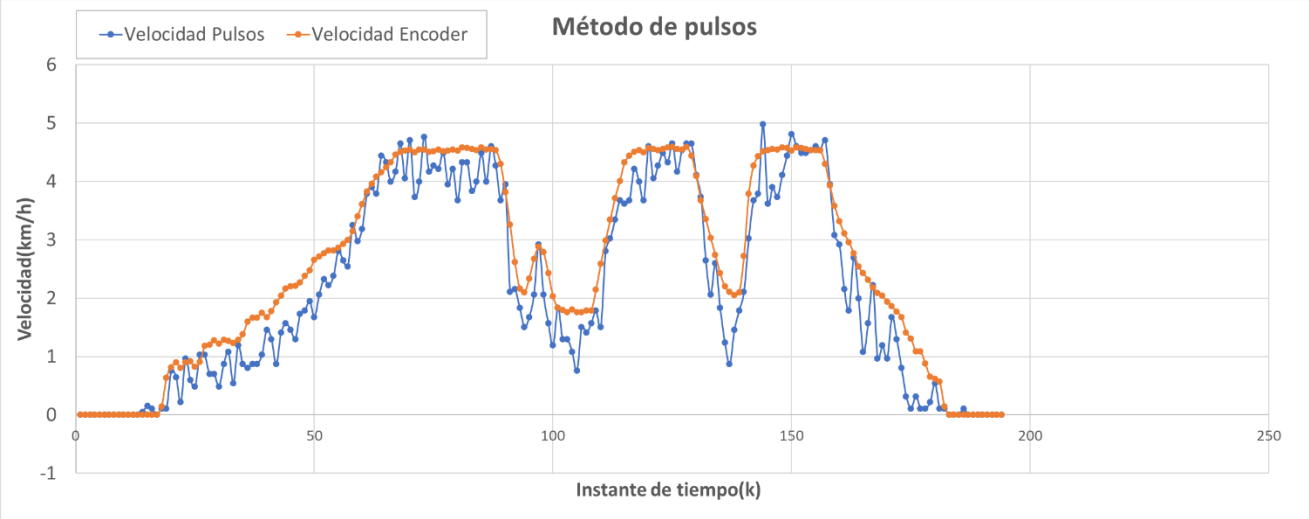


Figura 9.3.1. Datos métodos de pulsos

En la figura 9.3.1, tenemos una representación de los datos capturados con el método de los pulsos. Se puede observar que el sensor de ultrasonidos y el *encoder* miden la velocidad aproximadamente con los mismos valores. Ambas medidas pueden presentar diferencias entre el valor de la velocidad real de la cinta, pues es posible que existan variaciones en la instalación del banco de prueba como que el sensor no esté a 45° o que la rueda del *encoder* no sea exactamente de 3 cm. Por este motivo más que estudiar la gráfica de la velocidad en sí mismo, nos interesa más saber si la proporcionalidad entre ambos valores es siempre constante, si la gráfica resultante es plana entonces implica que existe una proporción perfecta entre los dos valores.

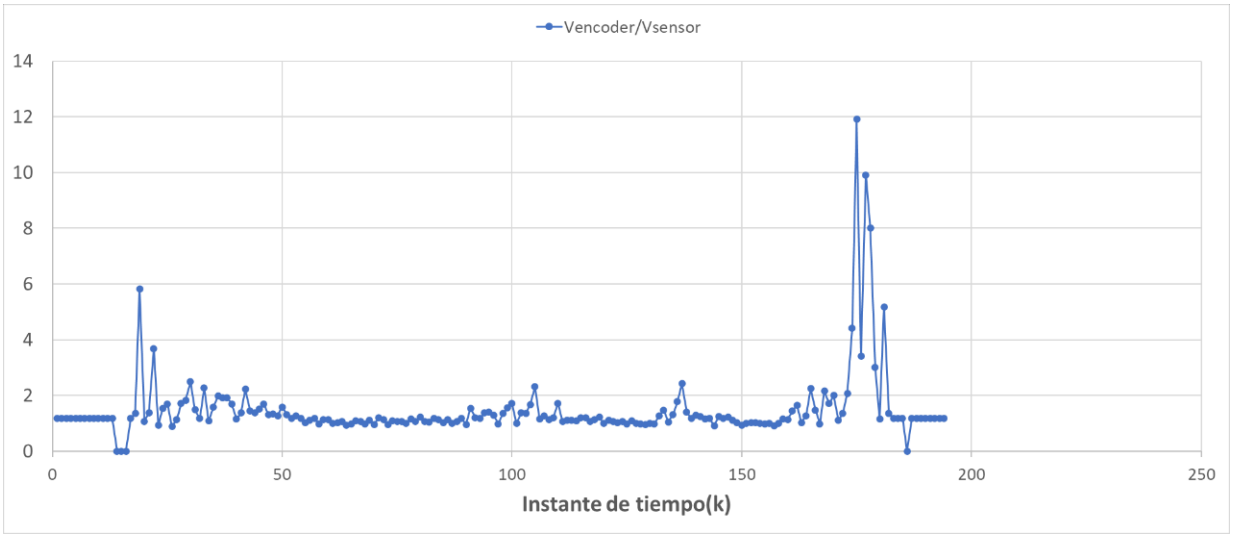


Figura 9.3.2. Relación velocidad encoder/sensor

En la figura 9.3.2 se muestra la relación de las velocidades del encoder entre las velocidades del sensor. En general se mantiene una proporcionalidad constante, pero en los extremos se observa una desviación mucho mayor, que coincide con las regiones de menor velocidad de la figura 9.3.1. Podemos representar de nuevo esta gráfica, pero cambiando el instante de tiempo, por el número de pulsos. Si realizamos este cambio y lo representamos con un diagrama de dispersión el resultado es el siguiente (Figura 9.3.3):

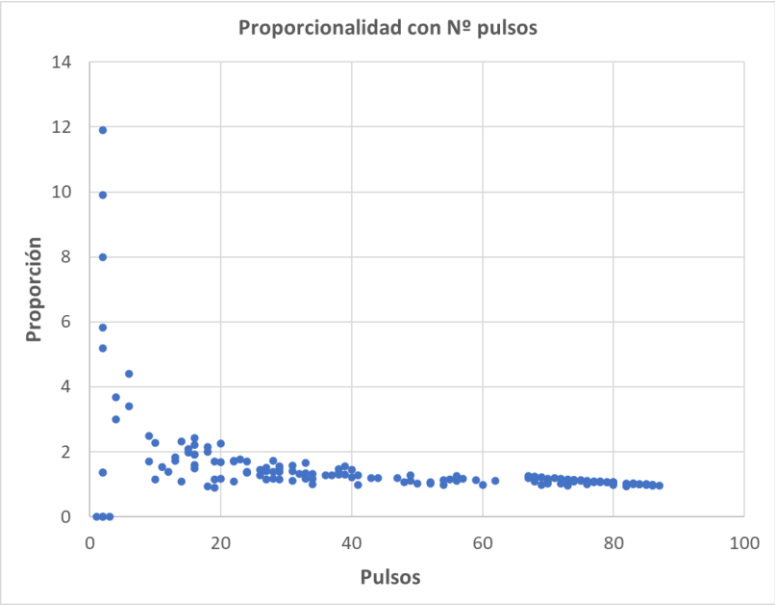


Figura 9.3.3. Relación velocidad encoder/sensor en función del número de pulsos

El número de pulsos es determinante para poder obtener un buen resultado de medida. A menor número de pulsos existe una mayor dispersión de los datos, lo cual concuerda con el error sistemático relativo que habíamos calculado (Figura 8.3.1). Podemos establecer un punto a partir de N puntos, donde el resultado lo consideremos aceptable. En este caso optamos por establecer este punto a partir de 20 pulsos, por debajo de eso las medidas serían demasiado ruidosas o poco precisas.

Estas gráficas nos dan una idea del comportamiento de este método, pero sería mejor si podemos cuantizar los datos para comparar de una forma más analítica los diferentes resultados. Para ello, disponemos de una herramienta estadística, el coeficiente de correlación de Pearson, el cual nos mide la correlación lineal entre dos variables independientemente de la escala de medida y se suele denotar con la letra ρ o r. Este coeficiente tiene su valor acotado entre [-1, 1], y dependiendo del valor tenemos varias interpretaciones:

- Si $0 < r < 1$, la correlación entre las dos variables es positiva, si una incrementa, la otra tambien lo hace.
- Si $-1 > r > 0$, la correlación entre las dos variables es negativa, si una incrementa, la otra disminuye.
- Si $r = 0$, implica que no existe una correlación lineal entre las dos variables. Esto no implica que no existan otro tipo de correlaciones no lineales.
- Si $r = -1$, existe una correlación negativa perfecta, si una variable incrementa la otra disminuye en la misma proporción.
- Si $r = 1$, existe una correlación positiva perfecta, ambas variables incrementan o disminuyen en la misma proporción.

Es decir, cuanto más cercano a 1 este coeficiente mejor será el resultado.

Para el método de pulsos y los datos de la figura 9.3.1, el coeficiente obtenido es $r = 0{,}9738$, un resultado que demuestra que este método nos da resultados con una muy buena precisión.

Ahora analicemos los resultados del método del periodo. Empecemos mostrando los datos recogidos sin realizar la media de los datos. (Figura 9.3.4)

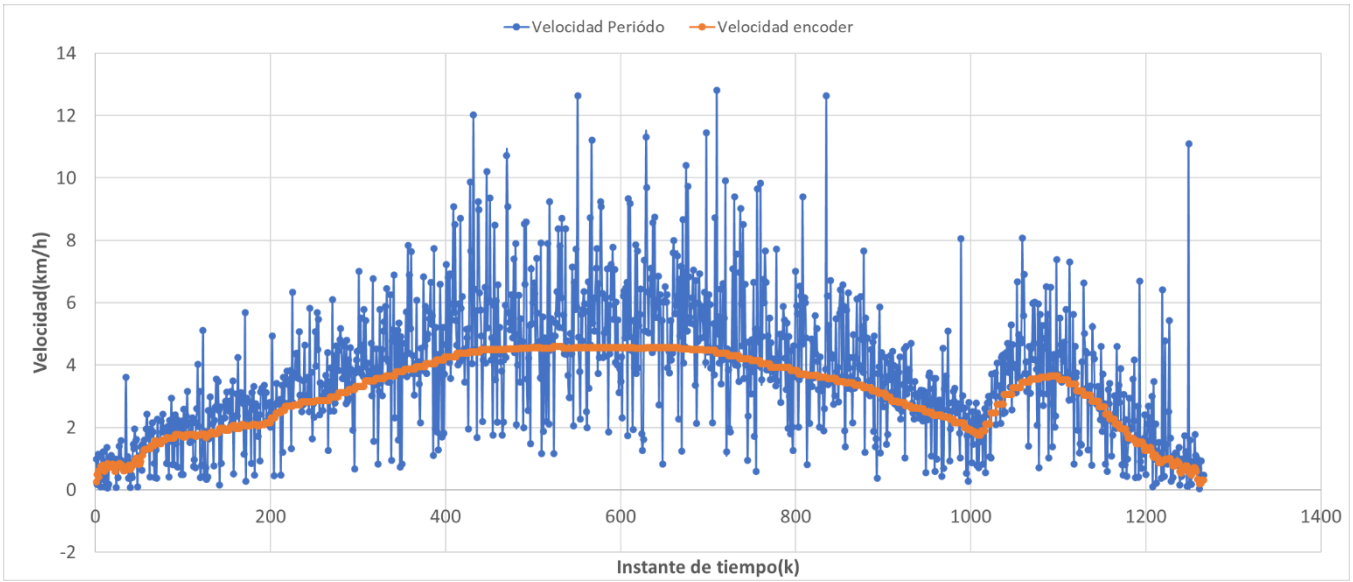


Figura 9.3.4. Datos método del periodo

Los datos obtenidos con este método son mucho más ruidosos que el método de pulsos. Esto contrasta con el error sistemático calculado y se debe principalmente a que en un sistema real existen cambio en la velocidad y aceleraciones por el movimiento del robot o las imperfecciones del terreno. Esto en combinación con el posible ruido eléctrico, conlleva a que la medida entre un periodo y el siguiente cambien de forma significativa. ¿Pero cómo se puede evitar o reducir este ruido?

La respuesta es simple, y en lugar de calcular la velocidad con el último periodo medido, realizarlo con la media de los N periodos anteriores.

La media móvil, es un indicador de la tendencia de los datos, y existen multitud de ellas. Tenemos la media móvil simple, la exponencial y la ponderada, en este caso usaremos la media móvil simple, que consiste en realizar la media aritmética de los N valores. [19]

El valor ideal de los N valores es importante para obtener una tendencia correcta de los datos, una media de pocos valores tendrá una respuesta más rápida, pero también más ruidosa, en cambio una media más grande será menos ruidosa, pero será más lenta y tardará más en responder. En la teoría es complicado calcular un valor de N que sea optimo pues dependerá de la naturaleza de los datos, así que para este proyecto obtendremos el valor de N de forma práctica, que mejor se adapte a los datos obtenidos.

Para ello nos apoyaremos de nuevo en el coeficiente de correlación de Pearson y realizaremos la media móvil de los datos de la figura 9.3.4, con distintos valores de N hasta encontrar el que nos del coeficiente más alto. Hemos realizado el cálculo con los siguientes valores:

Tabla 4

Coeficiente de correlación en función de N

N	Coef correlación
1	0,670706386
5	0,886106899
10	0,94015734
20	0,969494702
50	0,959253183
100	0,890169521

En la tabla mostrada se ha calculado el coeficiente correlación para los valores de N de, 1, 5, 10, 20, 50 y 100. Hay una buena mejoría de los valores hasta alcanzar un máximo en aproximadamente 20 muestras, pasando de $0{,}67$ a $0{,}97$ aproximadamente. Si se sigue incrementando las muestras, llegará el punto que el resultado pierda la correlación por el desfase introducido. Esto se muestra mejor en la siguiente figura:

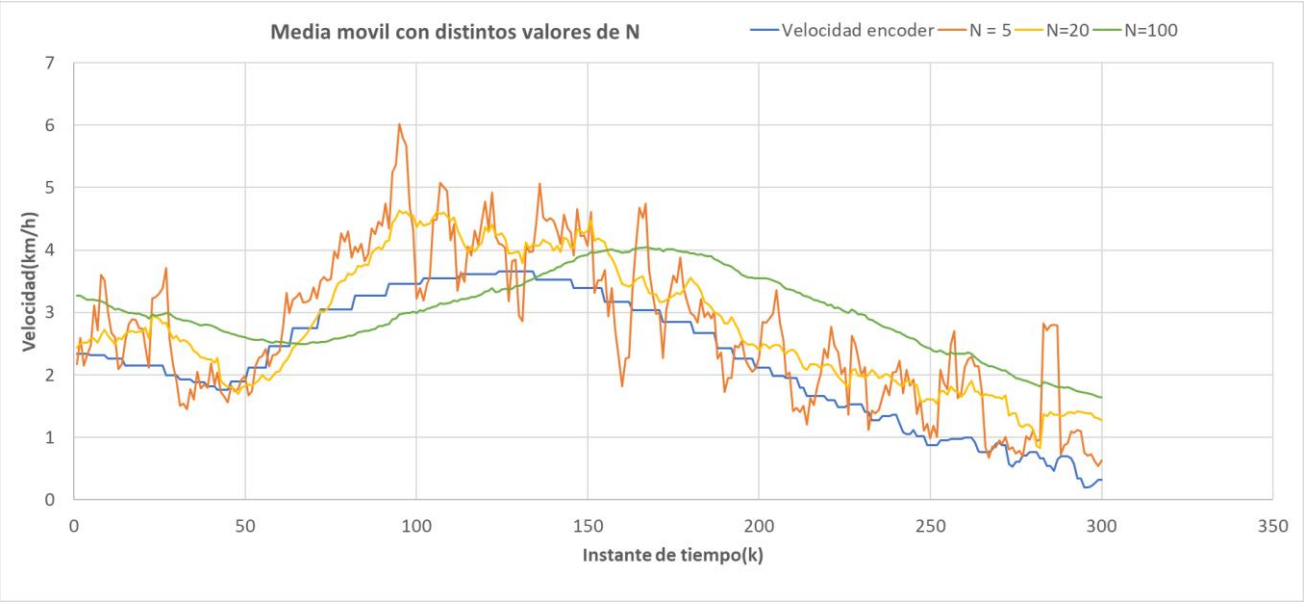


Figura 9.3.5. Media móvil con distintos valores de N

En la figura 9.3.5 se han representado los datos de la media móvil para 5, 20 y 100 muestras, y efectivamente cuanto mayor número de muestras mayor retardo en la señal, por eso a pesar de tener menor ruido no tiene tan buena correlación. Por lo tanto, eligiéremos la opción de 20 muestras que se adapta bastante bien a la señal original llegando a un compromiso entre precisión y respuesta.

Por último, para determinar cómo cambia el error relativo con la frecuencia a causa del error aleatorio, podemos realizar algo similar al método de pulsos. Podemos calcular el error, suponiendo el valor del *encoder* como el valor real de la velocidad:

$$E_r = \frac{|\Delta X|}{X_{real}} = \frac{|X_{real} - X_{medida}|}{X_{real}} \rightarrow X_{real} > 0 \rightarrow E_r = \left|1 - \frac{X_{medido}}{X_{real}}\right| \quad (38)$$

Sabemos que el dato del *encoder,* puede no ser el real, pero lo que nos interesa más bien es estudiar la dispersión del error. En este sentido, el único térmico que determina ese grado de dispersión del error relativo es la relación entre el valor medido y el real. Si calculamos dicha expresión y representamos los datos el resultado es el siguiente. (Figura 9.3.6)

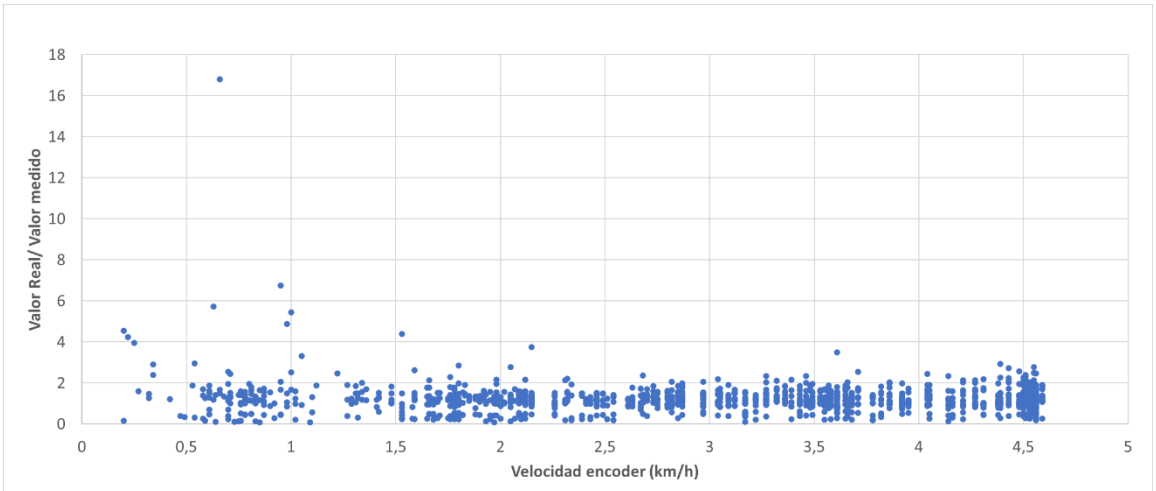


Figura 9.3.6. Proporcionalidad con el método del periodo

Los resultados obtenidos presentan una particularidad y es que a pesar de ser un método más ruidoso que el de pulsos, la dispersión de los datos frente a la medida real se mantiene prácticamente constante con la frecuencia.

10. Diseño final

10.1 Combinado ambos métodos.

Ya analizamos cada uno de los métodos y hemos podido sacar varias conclusiones de ambos, extrayendo las siguientes diferencias:

- El método de pulsos funciona mejor que el del periodo, pero se comete mucho error si no se leen suficientes pulsos.
- El método de pulsos no puede medir por debajo del intervalo de tiempo, mientras que con el periodo no tiene esta limitación en baja frecuencia.
- La medida del periodo es muy ruidosa, pero presenta un error relativo casi constante con la frecuencia.
- El método del periodo no puede determinar por si solo frecuencia 0 pues esto implica medir un periodo con un tiempo infinito.

Como ambos tienen sus ventajas y desventajas, vamos a utilizar lo mejor de cada uno y realizar una combinación para mejorar los resultados.

La idea es realzar la mayor parte de las medidas con los pulsos, pero en los casos que se quiera medir baja velocidad (baja frecuencia), utilizaremos la medida del periodo.

Como no podemos determinar si debemos medir con pulsos o con el periodo sin saber la velocidad, lo que realizaremos será medir siempre la velocidad con el método del periodo, si luego pasado el intervalo de tiempo establecido el número de pulsos contados supera o es igual a 20 descartamos la medida del periodo y calculamos una nueva con los pulsos obtenido.

Para la creación del código cogemos como base el usado para medir los pulsos y añadimos el cálculo con el periodo como una función.

```
#define N 20
//Definimos estas variables globales, que se deben ejecutar en varias funciones
volatile unsigned long pulso = 0;
volatile unsigned int pulsoencoder = 0;
volatile byte pulsorecibido = 0;
//En este caso, está variable es necesaria inicializarla antes de la funcion loop
unsigned long t = 0;
unsigned long T[N];
byte j = 0;

void setup() {

 Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);

  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);

}
```

Figura 10.1.1. Inicialización método combinado

La figura 10.1.1, es el código de la inicialización, la cual no tiene demasiados cambios. Si se ha añadido ahora una salida más, el pin 13 que está conectado al led integrado de Arduino Uno, que usaremos como indicador de que estamos midiendo con pulsos o con el periodo. Ahora en la interrupción del sensor usaremos la combinación de ambos códigos. (Figura 10.1.2)

```
void interrupcion() {
  pulsorecibido = 1;
  pulso++;
}
```

Figura 10.1.2. Interrupción combinada

Definimos el método de pulsos como una función de tipo “float” que llamaremos “vperiodo” y la llamamos al inicio del *Void Loop*.

```
void loop() {
  //Calculamos la velocidad con el periodo
  float v_periodo = vperiodo();
  //Tomamos el tiempo actual en microsegundos
  unsigned long tl = micros();
  //Calculamos la velocidad cada 250 ms
  if((tl - t) > 250E3){
    unsigned int pulso_medido = 0;
    unsigned int pulsoencoder_medido = 0;
    noInterrupts();
      pulso_medido= pulso;
      pulsoencoder_medido = pulsoencoder;
      pulso = 0;
      pulsoencoder = 0;
      //Copiamos las variables y las reseteamos para que siga contando
    interrupts();
    unsigned long    Tmedida = tl -t;
    t = micros();
    float v;
    //Determinamos si tenemos el número suficiente de pulsos
    if(pulso_medido>=20){
      digitalWrite(13,HIGH);
      float    f = ((float)pulso_medido*1E6)/(2*(float)Tmedida);
       //Calculamos la frecuencia del sensor

       v = 3.6*340*f/(2*40E3*0.707);
      for(int i = 0; i < N; i++){
       T[i] = 2*Tmedida/pulso_medido;
      }
    }
    else{
      digitalWrite(13,LOW);
      if(pulso_medido < 1){
       v = 0;

      }
      else{
      v = v_periodo;
      }
    }
    float fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
    float vencoder = 3.6*fencoder*0.03*3.14159;
```

Figura 10.1.3. *Void Loop* combinado

La parte más importante del código radica en el *Void Loop* (figura 10.1.3). Tras calcular la velocidad, estimaremos el número de pulsos con un intervalo de 250 ms, es decir a 4 datos por segundo que para los propósitos requeridos es suficiente.

Si en este periodo de tiempo se miden 20 pulsos o más activamos el led y calculamos la velocidad con los pulsos. Además, como este método es más preciso si se cumple rellenamos el *buffer* del periodo con este valor, para que en caso de medir menos de 20 pulsos la medida del periodo parta del último valor obtenido.

Para el caso que no se cumpla el requisito de pulsos apagamos el led y usaremos la velocidad del periodo, siempre y cuando se mida al menos un pulso. Esto es básicamente para evitar esperar de forma infinita para indicar que la velocidad del robot es 0. Por esto, indicamos que la velocidad es 0, y en este caso no cambiamos el *buffer* del periodo pues implicaría poner valores infinitos cosa que no es posible. Si se podría introducir un valor lo suficientemente grande como para que su cálculo de aproximadamente 0, pero de todas formas tarde o temprano se recibirá un pulso y su valor será relativamente grande. En función de las condiciones se podría realizar el cambio mencionado en el código y estudiar su efecto.

10.2 Mejorando el filtro

Además de realizar la combinación de los métodos, probaremos a ver si es posible cambiar el filtro para mejor la precisión de la medida a bajas frecuencia. Cambiaremos el condensador de la frecuencia de corte inferior del filtro pasa banda de $2{,}2\ \mu F$ a $3{,}3\ \mu F$. Este cambio llevara la frecuencia de corte desde unos $72\ Hz$ a $48\ Hz$, obteniendo el siguiente resultado: (Figura 10.2.1)

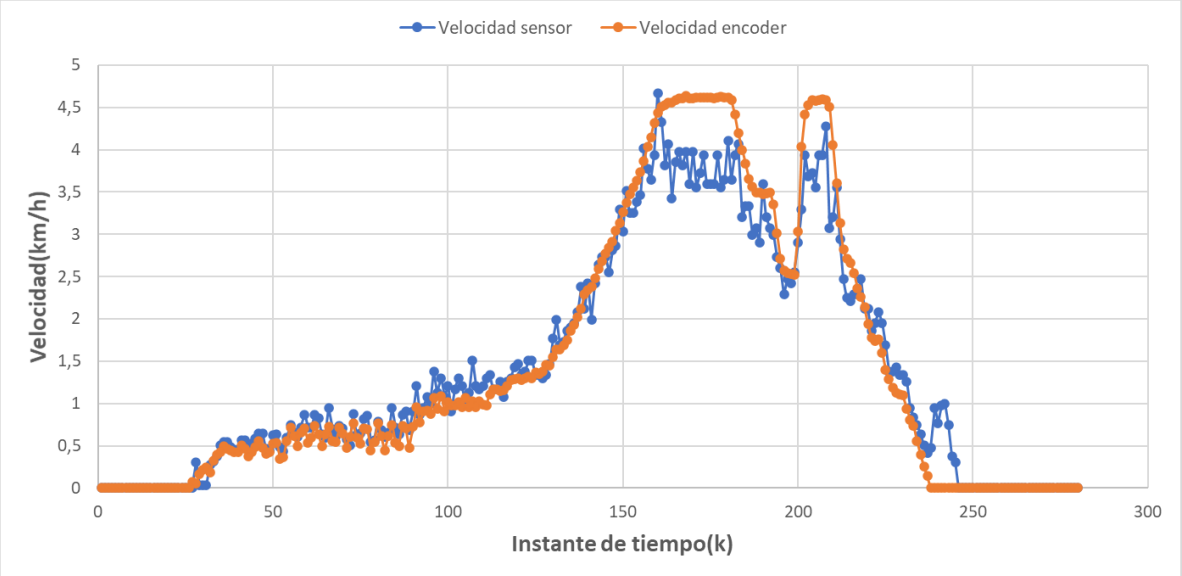


Figura 10.2.1. Datos combinados con $3,3\ \mu F$

Aunque los resultados obtenidos son bastante buenos, se observa que existe un recorte en las altas frecuencias. Para encontrar el motivo de este problema analizamos la salida del filtro y el comparador con el osciloscopio. (Figura 10.2.2)

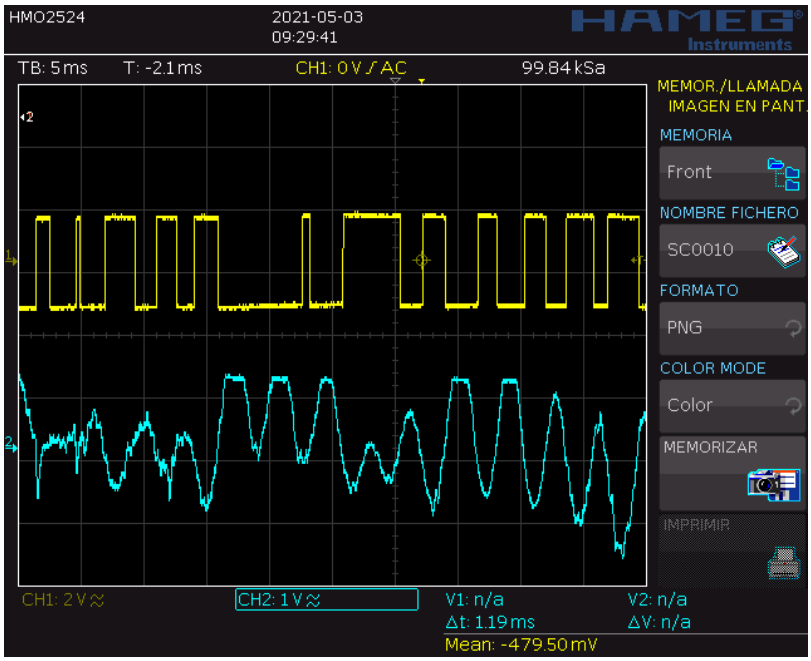


Figura 10.2.2. Comportamiento del filtro con $3,3\ \mu F$

Con el osciloscopio se observa claramente el problema, al bajar la frecuencia de corte inferior, las bajas frecuencia se atenúan menos, y produce en el caso de medir alta frecuencia que se cuele mayor cantidad de ruido. El ruido de baja frecuencia cambia el nivel de continua de la señal, haciendo que el comparador con histéresis no funcione correctamente.

El problema radica entonces en no encontrar un valor óptimo para la frecuencia inferior del filtro. Lo ideal sería ajustar el ancho de banda en función de la frecuencia medida y para ello podemos realizar este ajuste utilizando dos condensadores distintos y un multiplexor analógico.

En un multiplexor demultiplexor analógico, tenemos una entrada/salida común que puede conectar con una salida/entrada en función de sus bits de selección. Con esto, podemos conectar el común del multiplexor en la posición del condensador C1 (Figura 7.1.2), y luego seleccionar la salida con el condensador deseado.

La salida de control será la misma que usábamos para operar el led del pin 13, de modo que si estamos midiendo el periodo (baja frecuencia) usaremos un condensador más grande, mientras que si medimos los pulsos (alta frecuencia) utilizaremos un condensador más pequeño.

El multiplexor que se utilizó fue el CD4051b, que es un multiplexor analógico 8 a 1. Su esquema de funcionamiento es el siguiente. (Figura 10.2.3)

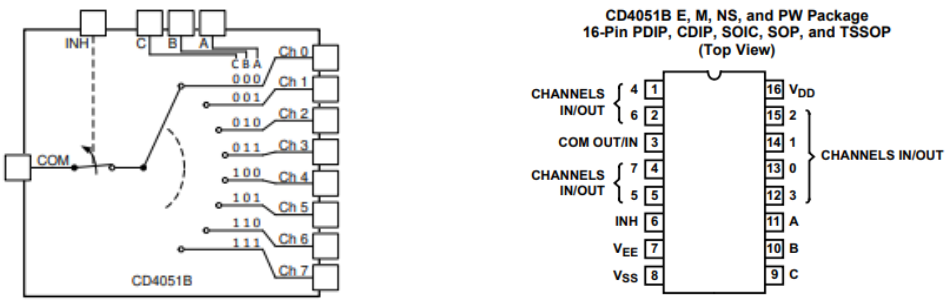


Figura 10.2.3. Diagrama CD4051b

Como solo usaremos dos condensadores usaremos dos canales que serán controlador por la entrada de selección A, y la entrada B y C irán a tierra junto con V_{EE}, Vss y INH (desactiva todos los canales si está a 1).

Aprovechando ahora que podemos ajustar mejor el filtro, en alta frecuencia usaremos un condensador de 1 μF con una frecuencia de corte inferior a 159 Hz.

De esta manera el esquema del circuito final de este proyecto nos queda como en la figura 10.2.4 y el montaje del prototipo en la figura 10.2.5.

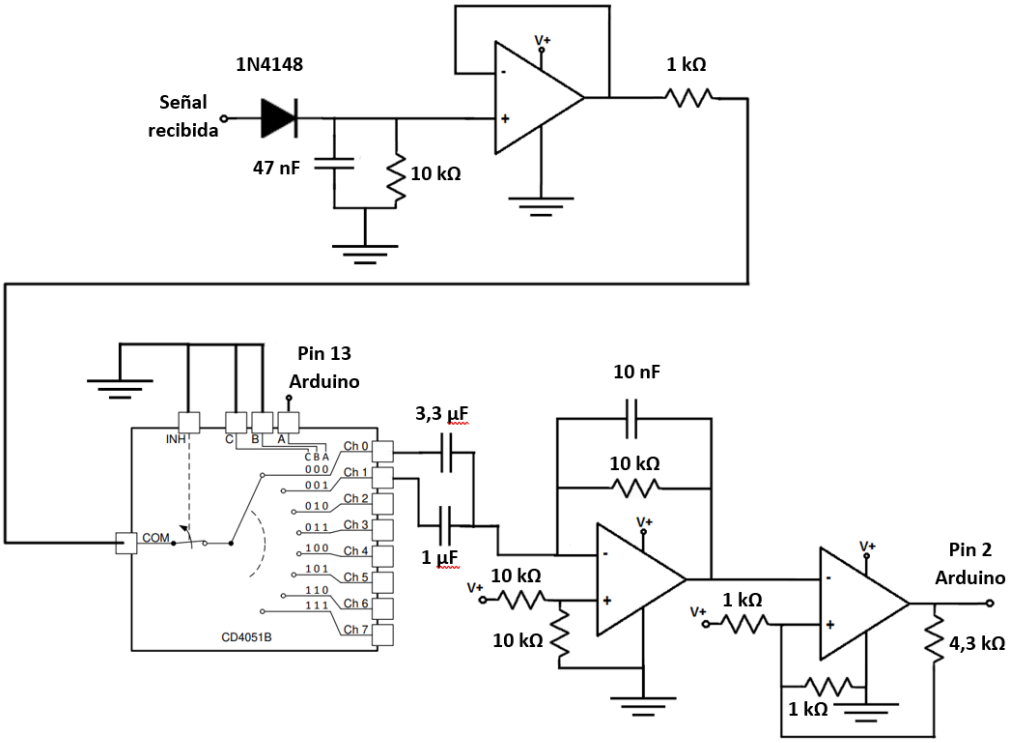


Figura 10.2.4. Esquema del circuito

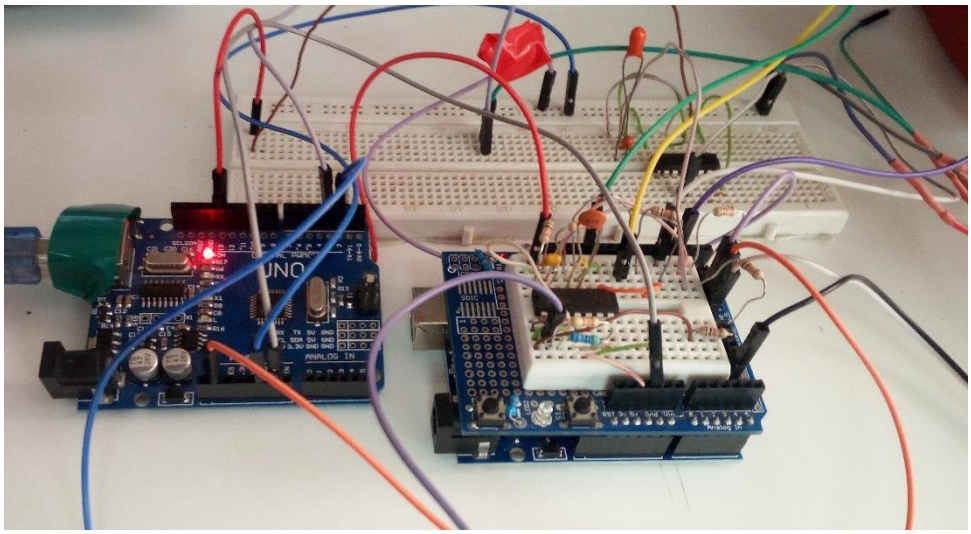


Figura 10.2.5. Prototipo final del circuito

Tras realizar la nueva configuración del circuito, probamos obteniendo nuevos datos. (Figura 10.2.6)

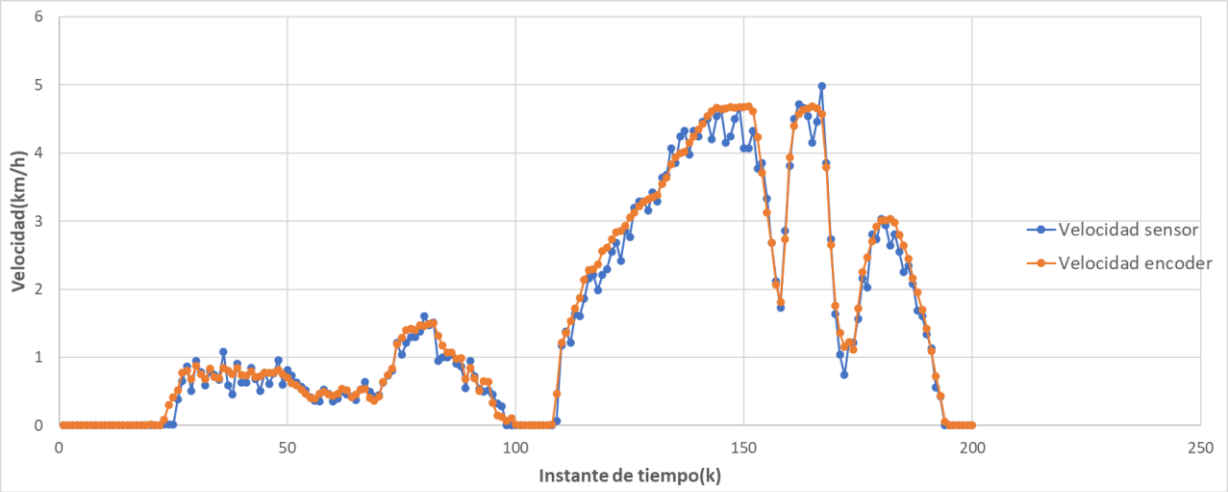


Figura 10.2.6. Datos con el multiplexor

La mejora con el nuevo circuito es bastante notable, dando el mejor resultado hasta ahora. La correlación de las variables es casi perfecta, con un coeficiente de 0,99369825, lo que demuestra que el sensor es capaz de tener una medida precisa de la velocidad con las condiciones experimentadas en el banco de pruebas.

10.3 Resultados finales.

Tras comprobar que el sensor es capaz de medir la velocidad de forma precisa si se cumplen las condiciones ideales, es el momento de probar el sensor en una situación más real para medir la velocidad de un robot. Instalaremos el sensor en una silla de ruedas eléctrica, un robot con dirección diferencial.

Procederemos de la misma forma que con la fase de pruebas, es decir, instalaremos el sensor en uno de los laterales de la silla y contrastaremos la velocidad medida del sensor con la velocidad del *encoder* de la rueda del mismo lado. La silla se comunica con los sensores a través del sistema operativo ROS (*Robot Operating System*), pensado para simplificar y manejar la programación del comportamiento complejo de los robots con la incorporación de herramientas y bibliotecas.

Para el caso del sensor deberemos modificar el código para que pueda enviar los datos de la velocidad por puerto Serial, pero siguiendo los estándares de comunicación de ROS. Esto lo realizaremos con librería “ros.h”. [20]

Por otro lado, en las pruebas teníamos dos Arduino para facilitar la tarea de cambiar el código, pero para la implementación final no es necesario, porque un solo Arduino tiene capacidad suficiente. Es por esto por lo que combinaremos el código del envió, con el de recepción con el multiplexor. Los cambios principales a realizar es eliminar del código relacionado con la medida del *encoder* de la cinta, incluir las interrupciones del emisor y declarar la clase de la comunicación ROS. También cambiaremos las unidades de la velocidad para mostrarla ahora en m/s.

Definimos la velocidad como un dato en float32. “velocidad_msg” guardará el mensaje a transmitir y con “pub_velocidad” se establece el nombre del dato para llamarlo luego en el sistema ROS. (Figura 10.3.1)

```
#include <ros.h>
#include <std_msgs/Float32.h>
//Definimos la clase de ROS
ros::NodeHandle  nh;
//Los mensajes tendran formato de Float32
std_msgs::Float32 velocidad_msg;
ros::Publisher pub_velocidad("Velocidad", &velocidad_msg);
```

Figura 10.3.1. Declaración ROS

Cuando se obtenga la medida de la velocidad se guarda en el mensaje de la clase y se transmitirá a través de la función “pub_velocidad.publish” y la función “nh.spinOnce”. (Figura 10.3.2)

```
velocidad_msg.data = v;
pub_velocidad.publish(&velocidad_msg);
nh.spinOnce();
```

Figura 10.3.2. Envío del mensaje en ROS

Con el código modificado, instalamos el sensor en la silla cerca del suelo y con un ángulo aproximado de 45°. Mostraremos dos pruebas, que se realizaron, una en el pasillo y otra en los aparcamientos de las instalaciones de la Universidad de La Laguna. Estas pruebas se guardan el robot como dos ficheros CSV, en uno se guarda los datos del sensor tiempo y velocidad, y en el otro fichero se guardan los datos del *encoder* de la rueda con la cuenta de sus pulsos.

El *encoder* de la rueda es de 8800 pulsos por revolución y su diámetro es de 0,32 metros. Con esto en cuenta podemos leer los ficheros en Python con el módulo “pandas” y con “matplotlib” y “numpy” podemos representar los datos. Los datos de la velocidad del *encoder* estarán representados en valor absoluto, pues el sensor de ultrasonidos no es capaz de distinguir la dirección de la velocidad.

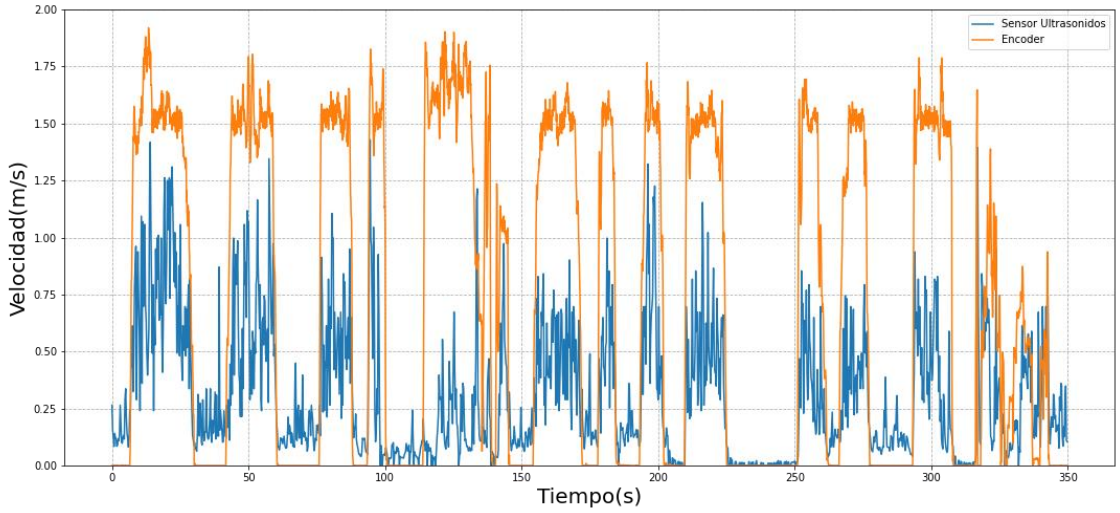


Figura 10.3.4. Datos de velocidad en el pasillo

La figura 10.3.4 son los datos del pasillo y muestran que existe una correlación con los datos del sensor de ultrasonidos y la velocidad del *encoder.* El coeficiente de correlación de Pearson existente es de 0,64977211, no tan bueno como los datos obtenidos con la cinta del banco de pruebas.

Esta diferencia en la medida se debe principalmente a que el suelo del pasillo está compuesto por baldosas lisas que no producen una buena reflexión de la onda, por lo que se pierden parte

de los pulsos de señal. Las reflexiones medidas serán en parte a las irregularidades del suelo como por ejemplo las juntas de las baldosas, produciendo una medida más ruidosa.

Por otro lado, hay ciertos puntos donde el robot este detenido, pero el sensor mide un ruido de fondo. El origen de este ruido podría ser a causa de ruido eléctrico, podría ser algún tipo de ruido que se sume a la onda recibida o incluso puede ser que el sensor haya medido la velocidad de algún objeto que tenga delante. Pero este aspecto no es tan preocupante pues es posible encontrar una solución, por ejemplo, cambiando los umbrales o estableciendo un mínimo de velocidad.

Si se estudia en más detenimiento la onda recibida se podría ajustar mejor el circuito para amplificar o filtrar mejor la onda, pudiendo mejorar un poco más los resultados, pero incluso con esto el sensor tendrá un límite impuesto por la cantidad de onda reflejada.

Viendo este resultado probamos en otra superficie para observar los resultados.

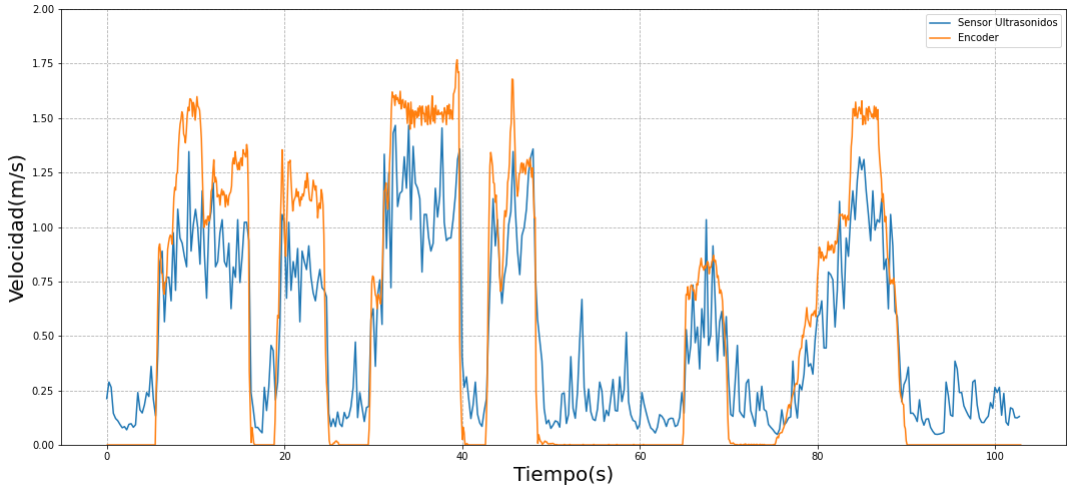


Figura 10.3.5. Datos de velocidad en los aparcamientos

La figura 10.3.5, son los datos capturados en los aparcamientos. La diferencia es notable, y la correlación es mucho mejor de 0,92983878 un valor más ajustado a las pruebas hecha en la lijadora.

La superficie de los aparcamientos es de asfalto y con pocas variaciones en el nivel. Esta superficie favorece la reflexión de la onda en todas las direcciones y mejora la recepción captando un mayor número de pulsos. Por este motivo la velocidad medida es mucho mayor que en el caso del pasillo, y también posee menos ruido.

Esto demuestra que el sensor si es capaz de medir la velocidad, pero su valor depende de la superficie, siendo mejor superficies porosas como el asfalto e incluso suelos arcillosos. Que el sensor opere mejor en superficies porosa no es una limitación para su uso, se puede utilizar perfectamente en robots que operen siempre en exteriores, como por ejemplos los coches autónomos, en este caso simplemente hay que ajustar el filtro pasa banda y se podrán medir velocidades mayores.

En contraposición en zonas de interior con suelos lisos las medidas carecerán de la precisión necesaria. Para medir de mejor forma en este tipo de suelos con el efecto *Doppler,* habría que cambiar algún parámetro del sensor o la tipología. Por ejemplo, si se aumenta la frecuencia de la onda transmitida, bajará su longitud de onda y será más fácil su dispersión, además la

frecuencia *Doppler* también aumenta, y favorece el método de pulsos. Otra mejora posible es cambiar el transmisor y receptor de posición, de forma que el receptor se encuentre delante del transmisor para captura la reflexión con el suelo teniendo en cuenta el ángulo de reflexión. Así se dejaría de depender tanto de la superficie, pero también produce una instalación un poco más compleja.

11. Conclusión

En la elaboración de este proyecto se ha expuesto la importancia de tomar medidas que sean precisas y con baja incertidumbre para la resolución de los problemas típicos encontrados en la odometría de robots móviles.

Se ha logrado demostrar que existen métodos alternativos para la medida de la velocidad de un robot terrestre, con la aplicación, diseño e implementación de un sensor de ultrasonidos. Mediante el uso del efecto *Doppler* y otras técnicas utilizadas como la demodulación AM, ha sido posible crear un dispositivo capaz de medir la velocidad con una resolución y precisión superior a las expectativas iniciales.

Los datos obtenidos con las herramientas y programas utilizados, junto con los conocimientos del grado de diversas disciplinas, son en parte la que han producido poder analizar cada uno de los problemas encontrados en cada una de las etapas, pudiendo llegar a diseñar un circuito y código más optimizado y con mejores resultados.

Es cierto que el sensor posee mejores resultados según el tipo de entorno al que este expuesto, siendo el caso más favorable de aplicación las superficies porosas por tener una mayor capacidad de reflexión en todos los sentidos. Pero esto no implica que no pueda ser usado en un robot real, de hecho, un robot de exterior que se mueva a una velocidad media sobre el asfalto logrará medidas precisas. Todo esto teniendo en cuenta un sensor que es de fácil instalación y con un coste económico. Además, el proyecto expuesto da la opción a múltiples acciones de mejora, como probar distintos modos de instalación o cambiar los valores del filtro para adaptarse mejor al entorno.

En conclusión, los resultados y la elaboración de este proyecto han logrado cumplir con los objetivos planteados, dando solución al problema de odometría típico al usar los sensores *encoder* que introducen errores de incertidumbre por el deslizamiento de las ruedas. El sensor diseñado aplaca este problema midiendo la velocidad sin depender del contacto físico con suelo.

12. Conclusion

In the development of this project, we have exposed the importance of taking measurements that are accurate and with low uncertainty for the resolution of typical problems encountered in the odometry of mobile robots.

It has been demonstrated that there are alternative methods for measuring the speed of a land robot, with the application, design and implementation of an ultrasonic sensor. Through the use of the Doppler effect and other techniques such as AM demodulation, it has been possible to create a device capable of measuring velocity with a resolution and accuracy higher than initial expectations.

The data obtained with the tools and programmes used, together with the knowledge of the degree of various disciplines, is partly what has made it possible to analyse each of the problems encountered in each of the stages, enabling us to design a more optimised circuit and code with better results.

It is true that the sensor has better results depending on the type of environment to which it is exposed, the most favourable case of application being porous surfaces as they have a greater capacity for reflection in all senses. But this does not imply that it cannot be used in a real robot; in fact, an outdoor robot moving at an average speed on asphalt will achieve accurate measurements. All this considering a sensor that is easy to install and economic in cost. In addition, the exposed project gives the option of multiple improvement actions, such as testing different installation modes or changing the filter values to better adapt to the environment.

In conclusion, the results and the development of this project have achieved the objectives set out, providing a solution to the typical odometry problem when using encoder sensors that introduce uncertainty errors due to wheel slippage. The sensor designed alleviates this problem by measuring speed without depending on physical contact with the ground.

13. Bibliografía

[1] J. L. Vargas Muñoz y P. E. Perez Salazar, «DISEÑO DE UN ROBOT MÓVIL CON TRACCIÓN,» 2020.

[2] «Cinemática de Mecanismos,» Universidad Carlos III, [En línea]. Disponible en: http://ocw.uc3m.es/ingenieria-mecanica/teoria-de-maquinas/material-de-clase-1/tema3-Cinematica.ppt. [Último acceso: 05 Julio 2021].

[3] J. L. Fernández, «Efecto Doppler,» FisicaLab, [En línea]. Disponible en: https://www.fisicalab.com/apartado/efecto-doppler. [Último acceso: 05 Julio 2021].

[4] «Fénomenos del sonido,» Univeso del Sonido, [En línea]. Disponible en: https://www.universodelsonido.cl/el-sonido/fenomenos/. [Último acceso: 05 Julio 2021].

[5] «Reflexión,» MUSIKI, [En línea]. Disponible en: http://musiki.org.ar/Reflexi%C3%B3n. [Último acceso: 05 Julio 2021].

[6] «Superposición de ondas,» Universidad del País Vasco, [En línea]. Disponible en: http://www.ehu.eus/acustica/espanol/basico/suones/suones.html. [Último acceso: 05 Julio 2021].

[7] F. Rey Micolau y F. Tarrés Ruiz, «Comunicaciones analógicas: modulaciones AM y FM,» [En línea]. Disponible en: http://openaccess.uoc.edu. [Último acceso: 05 Julio 2021].

[8] «Encoder ¿como funciona? y sus tipos,» Ingeniería Mecafenix, 28 Abril 2017. [En línea]. Disponible en: https://www.ingmecafenix.com/automatizacion/encoder/. [Último acceso: 05 Julio 2021].

[9] «Software de Arduino,» Arduino.cl, [En línea]. Disponible en: https://arduino.cl/programacion/. [Último acceso: 05 Julio 2021].

[10] «LTspice,» Analog Devices, [En línea]. Disponible en: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html. [Último acceso: 05 Julio 2021].

[11] «Maxima, a Computer Algebra System,» [En línea]. Disponible en: https://maxima.sourceforge.io/. [Último acceso: 05 Julio 2021].

[12] «STM32CubeIDE,» STMicroelectronics, [En línea]. Disponible en: https://www.st.com/en/development-tools/stm32cubeide.html. [Último acceso: 05 Julio 2021].

[13] «Processing,» [En línea]. Disponible en: https://processing.org/. [Último acceso: 05 Julio 2021].

[14] «Anaconda,» [En línea]. Disponible en: https://www.anaconda.com/products/individual. [Último acceso: 05 Julio 2021].

[15] «Making a better HC-SR04 Echo Locator,» Emil's Projects & Reviews, 22 Enero 2014. [En línea]. Disponible en: https://uglyduck.vajn.icu/ep/tags/hc-sr04/. [Último acceso: 05 Julio 2021].

[16] « Description of STM32F1 HAL and low-layer drivers,» STMicroelectronics, Febrero 2020. [En línea]. Disponible en: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf. [Último acceso: 05 Julio 2021].

[17] F. G. STREMLER, «INTRODUCCION A LOS SISTEMAS DE COMUNICACION,» de *Edicion Especial*, Addison wesley Longman, 2006, p. 20.

[18] «Incierto,» Universidad Complutense de Madrid, 10 Febrero 2011. [En línea]. Disponible en: https://fisicas.ucm.es/data/cont/media/www/pag-39682/incierto.pdf. [Último acceso: 05 Julio 2021].

[19] «Medias móvil simple, exponencial y ponderada: formulas y ejemplos,» Rankia, 29 Abril 2020. [En línea]. Disponible en: https://www.rankia.cl/blog/analisis-ipsa/2039072-medias-movil-simple-exponencial-ponderada-formulas-ejemplos. [Último acceso: 05 Julio 2021].

[20] «Rosserial,» 18 Marzo 2020. [En línea]. Disponible en: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup. [Último acceso: 05 Julio 2021].

[21] O. Baturone, *Robótica: Manipuladores y robots móviles,* Aníbal, 2001.

[22] «HC-SR04,» Hardwarelibre, [En línea]. Disponible en: https://www.hwlibre.com/hc-sr04/. [Último acceso: 05 Julio 2021].

[23] «Proyecto fotovoltaico,» [En línea]. Disponible en: https://www2.ineel.mx/proyectofotovoltaico/preg_20.html. [Último acceso: 05 Julio 2021].

14.Anexo

14.1 Código Arduino

1. Generador de pulsos

```
//HA es la salida para la alimentacion positiva del altavoz
//HB es la salida para la alimentacion negativa del altavoz

#define HA 11
#define HB 3
#define A 99 // 99 es el valor donde B = A valor más estable

uint8_t a = 0;

void setup() {
  pinMode(HA, OUTPUT);
  pinMode(HB, OUTPUT);

 SREG = (SREG & 0b01111111); //Desabilitar interrupciones
 TIMSK2 = TIMSK2|0b00000010; //Habilita la interrupcion por el comparador A
 TCCR2A = 0b01010010; //Habilita el modo ctc
 TCCR2B = 0b00000001; //Preescaler 1
 //Establecemos los valores de los comparadores
 OCR2A = A;
 OCR2B = A;
 TCNT2 = 0; //Resetea contador

 SREG = (SREG & 0b01111111) | 0b10000000; //Habilitar interrupciones //Desabilitar interrupciones


}

void loop() {

}

ISR(TIMER2_COMPA_vect){
   a++;
   if(a > 3) a = 0;
   switch(a){
    case 0:
     TCCR2A = 0b01100010; //Establece COMPA en toggle y COMPB en clear
    break;

    case 1:
     TCCR2A = 0b10100010; //Establece COMPA en clear y COMPB en clear
    break;

    case 2:
      TCCR2A = 0b10010010; //Establece COMPB en toggle y COMPA en clear
    break;

    case 3:
      TCCR2A = 0b10100010;  //Establece COMPA en clear y COMPB en clear
    break;
   }

}
```

2. Receptor con método de pulsos.

```
//Definimos estas variables globales, que se deben ejecutar en varias funciones
unsigned long pulso = 0;
unsigned int pulsoencoder = 0;

//En este caso, está variable es necesaria inicializarla antes de la funcion loop
unsigned long t = 0;


void setup() {

 Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);
}


void loop() {
  //Tomamos el tiempo actual en microsegundos
  unsigned long tl = micros();

  //Calculamos la velocidad cada 200 ms
  if((tl - t) > 200E3){
    unsigned int pulso_medido = 0;
    unsigned int pulsoencoder_medido = 0;
    //Desactivamos las interupciones
    noInterrupts();
      pulso_medido= pulso;
      pulsoencoder_medido = pulsoencoder;
      pulso = 0;
      pulsoencoder = 0;
      //Copiamos las variables y las reseteamos para que siga contando
    interrupts();
    //Definimos el intervalo de medida y reseteamos el intervalo de tiempo
    unsigned long    Tmedida = tl -t;
    t = micros();
    float   f = ((float)pulso_medido*1E6)/(2*(float)Tmedida);
    //Calculamos la frecuencia del sensor

    float fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
    float v = 3.6*340*f/(2*40E3*0.707);
    float vencoder = 3.6*fencoder*0.03*3.14159;
    //Calculamos la frecuencia del encoder

    Serial.print(pulso_medido);
    Serial.print(",");
    Serial.print(v);
    Serial.print(",");
    Serial.println(vencoder);

  }

}

//Definimos la interrupcion de los pulsos del sensor
void interrupcion() {
  //incrementamos el contador
  pulso++;


}

//Definimos la interrupcion de los pulsos del encoder
void encoder() {
  //incrementamos el contador
  pulsoencoder++;
}
```

3. Receptor con método del periodo.

```
#define N 20
//Definimos las variables globales
byte pulsorecibido = 0;
unsigned int pulsoencoder = 0;

//Se inicializan las variables que deben tener un valor definido antes de la funcion void loop
unsigned long t = 0;
unsigned long t2 = 0;
unsigned long T[N];
byte j = 0;
float fencoder = 0;

void setup() {

  Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);
}


void loop() {
 //Varaible para contar el tiempo en microsegundos
  unsigned long t1 = micros();
  //Tomamos las velocidad del encoder cada 100 ms
  if(t1-t2>100E3){
    float pulsoencoder_medido = 0;

    noInterrupts();
     pulsoencoder_medido = pulsoencoder;
     pulsoencoder = 0;
     //Copiamos el numero de pulsos y restablecemos la variable para que pueda seguir contando
    interrupts();
    unsigned long    Tmedida =( t1 - t2);
    t2 = micros();
    fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
    //Calculamos la frecuencia del encoder (revoluciones por segundo)
  }
  if(pulsorecibido){

     unsigned long    Tmedida =( t1 - t);
     //Calculamos el periódo

     t = micros();
     //Como estamos midiendo en los cambios el periódo medido es la mitad
     //Guardamos el dato en un array que dispone de los N valores anteriores
     //Guardamos de froma ciclica segun el indice j, de modo que siempre guardamos
     //el nuevo dato, en la posición del dato más antiguo
     if(j <= (N-1)) T[j] = 2*Tmedida;


     unsigned long Ttotal = 0;

     for(int i = 0; i < N; i++){
      Ttotal += T[i];
     }
     //Calculamos la media de todos los periódos

     float Tmedia = (float)(Ttotal)/N;
     float f = ((float)1E6)/(Tmedia);
     float v = 3.6*340*f/(2*40E3*0.707);
     j++;
     if(j>N){
      //Cuando el valor de conteo supera el valor de N, se muestran los datos
      //Hay que tener en cuenta que solo se mostraran los datos al tomar N muestras
      //Además, las función Serial.print lleva un tiempo de ejecucion más alto,
      //por lo que las medidas en alta frecuencia se van a ver más afectadas
```

```
      float vencoder = 3.6*fencoder*0.03*3.14159;
       Serial.print(v);
        Serial.print(",");
       Serial.println(vencoder);
       j = 0;
      }
       pulsorecibido = 0;


  }

}
//Función de interrupción del sensor
void interrupcion() {
  pulsorecibido = 1;

}

//Función de interrupciones del encoder
void encoder() {
  pulsoencoder++;

}
```

4. Receptor con método combinado.

```
#define N 20
//Definimos estas variables globales, que se deben ejecutar en varias funciones
volatile unsigned long pulso = 0;
volatile unsigned int pulsoencoder = 0;
volatile byte pulsorecibido = 0;
//En este caso, está variable es necesaria inicializarla antes de la funcion loop
unsigned long t = 0;
unsigned long T[N];
byte j = 0;

void setup() {

 Serial.begin(115200); //Inicializamos la comunicación

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);

  //Establecemos el pin 3 como entrada (encoder) y habilitamos las interrrupciones en los flancos negativos
  pinMode(3, INPUT_PULLUP);
  attachInterrupt(1, encoder, FALLING);

  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);

}
void loop() {
  //Calculamos la velocidad con el periodo
  float v_periodo = vperiodo();
  //Tomamos el tiempo actual en microsegundos
  unsigned long tl = micros();
  //Calculamos la velocidad cada 250 ms
  if((tl - t) > 250E3){
    unsigned int pulso_medido = 0;
    unsigned int pulsoencoder_medido = 0;
    noInterrupts();
      pulso_medido= pulso;
      pulsoencoder_medido = pulsoencoder;
      pulso = 0;
      pulsoencoder = 0;
      //Copiamos las variables y las reseteamos para que siga contando
    interrupts();
    unsigned long    Tmedida = tl -t;
    t = micros();
    float v;
    //Determinamos si tenemos el número suficiente de pulsos
    if(pulso_medido>=20){
      digitalWrite(13,HIGH);
      float   f = ((float)pulso_medido*1E6)/(2*(float)Tmedida);
       //Calculamos la frecuencia del sensor
```

```
        v = 3.6*340*f/(2*40E3*0.707);
       for(int i = 0; i < N; i++){
        T[i] = 2*Tmedida/pulso_medido;
       }
     }
     else{
       digitalWrite(13,LOW);
       if(pulso_medido < 1){
        v = 0;

       }
       else{
       v = v_periodo;
       }
     }
     float fencoder = ((float)pulsoencoder_medido*1E6)/(200*(float)Tmedida);
     float vencoder = 3.6*fencoder*0.03*3.14159;
     //Calculamos la frecuencia del encoder

  Serial.print(pulso_medido);
    Serial.print(",");
    Serial.print(v);
    Serial.print(",");
    Serial.println(vencoder);

  }

}

void interrupcion() {
  pulsorecibido = 1;
  pulso++;
}
void encoder() {
  pulsoencoder++;

}

//Definimos una funcion para determinar la velocidad con el periodo
float vperiodo(){
  unsigned long t1 = micros();
  //Definimos el intervalo de tiempo y la velocidad con variable
  //static para conservar su valor al salir de la función

  static unsigned long tant = 0;
  static float v = 0;
  if(pulsorecibido){
     unsigned long    Tmedida =( t1 - tant);
     //Calculamos el periódo

     tant = micros();
     //Como estamos midiendo en los cambios el periódo medido es la mitad
     //Guardamos el dato en un array que dispone de los N valores anteriores
     //Guardamos de froma ciclica segun el indice j, de modo que siempre guardamos
     //el nuevo dato, en la posición del dato más antiguo
     if(j <= (N-1)) T[j] = 2*Tmedida;

     unsigned long Ttotal = 0;

     for(int i = 0; i < N; i++){
      Ttotal += T[i];
     }
     //Calculamos la media de todos los periódos

     float Tmedia = (float)(Ttotal)/N;
     float f = ((float)1E6)/(Tmedia);
      v = 3.6*340*f/(2*40E3*0.707);
     j++;
     if(j>=N) j = 0;
      pulsorecibido = 0;


  }
  return v;
}
```

5. Código de receptor y emisor combinado (implementación final).

```
#include <ros.h>
#include <std_msgs/Float32.h>
//Definimos la clase de ROS
ros::NodeHandle  nh;
//Los mensajes tendran formato de Float32
std_msgs::Float32 velocidad_msg;
ros::Publisher pub_velocidad("Velocidad", &velocidad_msg);

//HA es la salida para la alimentacion positiva del altavoz
//HB es la salida para la alimentacion negativa del altavoz


#define N 20
#define HA 11
#define HB 3
#define cte 198 //Fclk/(2 * Preescaler *40kHz) - 2 -> Fclk = 16 MHz
#define A 99
//unsigned long  t2 = 0;
uint8_t a = 0;
//int t =0;
//Definimos estas variables globales, que se deben ejecutar en varias funciones
volatile unsigned long pulso = 0;
volatile unsigned int pulsoencoder = 0;
volatile byte pulsorecibido = 0;
//En este caso, está variable es necesaria inicializarla antes de la funcion loop
unsigned long t = 0;
unsigned long T[N];
byte j = 0;

void setup() {
  //Inicializamos la comunicación ROS
  nh.initNode();
  nh.advertise(pub_velocidad);

  //Establecemos el pin 2 como entrada y habilitamos las interrrupciones en el cambio de estado
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(0, interrupcion, CHANGE);


  pinMode(13,OUTPUT);
  digitalWrite(13,LOW);




  pinMode(HA, OUTPUT);
  pinMode(HB, OUTPUT);
//  pinMode(13, OUTPUT);
 SREG = (SREG & 0b01111111); //Desabilitar interrupciones
 TIMSK2 = TIMSK2|0b00000010; //Habilita la interrupcion por el comparador A
 TCCR2A = 0b01010010; //Habilita el modo ctc
 TCCR2B = 0b00000001; //Preescaler 1
 OCR2A = A;
 OCR2B = A;
 TCNT2 = 0; //Resetea contador

 SREG = (SREG & 0b01111111) | 0bl0000000; //Habilitar interrupciones //Desabilitar interrupciones

  //Serial.begin(115200);
}

void loop() {
  float v_periodo = vperiodo();
  //Tomamos el tiempo actual en microsegundos
  unsigned long tl = micros();
  //Calculamos la velocidad cada 250 ms
  if((tl - t) > 250E3){
    unsigned int pulso_medido = 0;
    unsigned int pulsoencoder_medido = 0;
    noInterrupts();
      pulso_medido= pulso;
      pulsoencoder_medido = pulsoencoder;
      pulso = 0;
      pulsoencoder = 0;
      //Copiamos las variables y las reseteamos para que siga contando
    interrupts();
    unsigned long    Tmedida = tl -t;
    t = micros();
    float v;
```

```
    if(pulso_medido>=20){
      digitalWrite(13,HIGH);
      float    f = ((float)pulso_medido*1E6)/(2*(float)Tmedida);
       //Calculamos la frecuencia del sensor


       v = 340*f/(2*40E3*0.707);
      for(int i = 0; i < N; i++){
       T[i] = 2*Tmedida/pulso_medido;
      }
    }
    else{
      digitalWrite(13,LOW);
      if(pulso_medido < 1){
       v = 0;

      }
      else{
      v = v_periodo;
      }
    }

      velocidad_msg.data = v;
      pub_velocidad.publish(&velocidad_msg);
      nh.spinOnce();

  }
}

void interrupcion() {
  pulsorecibido = 1;
  pulso++;


}


 float vperiodo(){
   unsigned long tl = micros();
   static unsigned long tant = 0;
   static float v = 0;
   if(pulsorecibido){

      unsigned long    Tmedida =( tl - tant);
      //Calculamos el periódo

      tant = micros();
      //Como estamos midiendo en los cambios el periódo medido es la mitad
      //Guardamos el dato en un array que dispone de los N valores anteriores
      //Guardamos de froma ciclica segun el indice j, de modo que siempre guardamos
      //el nuevo dato, en la posición del dato más antiguo
      if(j <= (N-1)) T[j] = 2*Tmedida;

      unsigned long Ttotal = 0;

      for(int i = 0; i < N; i++){
       Ttotal += T[i];
      }
      //Calculamos la media de todos los periódos

      float Tmedia = (float)(Ttotal)/N;
      float f = ((float)1E6)/(Tmedia);
       v = 340*f/(2*40E3*0.707);
      j++;
      if(j>=N) j = 0;
       pulsorecibido = 0;


   }
   return v;
}
```

```
ISR(TIMER2_COMPA_vect){
 // PORTB = (PORTB & 0b011111) | ((PORTB & 0b100000) ^ (1 << 5));

   a++;
   if(a > 3) a = 0;
   switch(a){
    case 0:
     TCCR2A = 0b01100010; //Establece COMPA en toggle y COMPB en clear

    break;
    case 1:
     TCCR2A = 0b10100010; //Establece COMPA en clear y COMPB en clear

    break;
    case 2:
      TCCR2A = 0b10010010; //Establece COMPB en toggle y COMPA en clear

    break;
    case 3:
      TCCR2A = 0b10100010;  //Establece COMPA en clear y COMPB en clear

    break;
   }


}
```

14.2 Código STM32 (main.c)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "usb_device.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "usbd_cdc_if.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

#define LEN 5000 //Tamaño del buffer de los datos recogidos
#define BUFFERLEN 2001 //Añado 1 más como identificador de la cadena
#define CHAINNUM LEN/((BUFFERLEN-1)/2) //Determina en cuantos paquetes dividir el
buffer

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint16_t datos[LEN];
uint8_t buffer[BUFFERLEN];
uint8_t datosListos = 0;

//Cuando los datos han finalizado se ejecuta la siguiente interrupción
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
	//Establecemos que los nuevos datos ya están listos
	datosListos = 1;

}


void crearBuffer(uint16_t dato, uint8_t *Hbyte, uint8_t *Lbyte ){
	//Pasamos el vector por referencia y separamos los datos originales
	// de 16 bits en dos de 8 bits

	*Lbyte = dato & 0xFF;
	*Hbyte = dato >> 8;

}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */
  uint8_t flancoP = 0;
  uint8_t bufferListo = 0;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
```

```
  {           //Obtenemos el estado del pulsador
        uint8_t state = HAL_GPIO_ReadPin(pulsador_GPIO_Port,pulsador_Pin);

        //Verificamos el flanco de subida del pulsador
        if(!state && !flancoP) flancoP = 1;
        else if (state && flancoP) flancoP = 0;
        else state = 1;

        if(datosListos){
              //Si los datos ya estan listos generamos el buffer y lo enviamos
               if(!bufferListo){
                      for(int j = 0; j < CHAINNUM; j++) {
                             for(int i = 0; i < (BUFFERLEN - 1)/2; i++) {
                                    crearBuffer(datos[i+j*(BUFFERLEN - 1)/2],
&buffer[2*i], &buffer[2*i + 1]);

                             }
                             //Añadimos a paquete su número para identificarlo en la
recepción
                             buffer[BUFFERLEN-1] = j;

                             for(int k = 0; k < 5; k++){
                                    //Trasmitimos y dejamos un tiempo entre envio
                             CDC_Transmit_FS(buffer, BUFFERLEN);
                             HAL_Delay(50); // 50 ms
                             }

                      }
                      //Encendemos el Led y establecemos que el envio ha finalizado
                      bufferListo = 1;
                      HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 0);

               }

        }
        if(!state) {
               //Si se pulsa el boton y se solicitan nuevos datos
               datosListos = 0;
               bufferListo = 0;
               //Empezamos la conversion por DMA
               HAL_ADC_Start_DMA(&hadc1, (uint32_t *) datos, LEN);
               //Apagamos el LED
               HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, 1);

        }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
```

```
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC|RCC_PERIPHCLK_USB;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV8;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLL_DIV1_5;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_5;
  sConfig.Rank = ADC_REGULAR_RANK_1;
```

```
  sConfig.SamplingTime = ADC_SAMPLETIME_71CYCLES_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : LED_Pin */
  GPIO_InitStruct.Pin = LED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : pulsador_Pin */
  GPIO_InitStruct.Pin = pulsador_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(pulsador_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
```

```
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

14.3 Código Processing

```
//Importamos la librería de comunicación serial y llamamos a la clase
import processing.serial.*;
Serial puerto;
//Definimos las constantes de tamaño de los buffer
final int buffersize = 2001;
final int datossize = 5000;
//Generamos el buffer de todos los datos
int[] datos = new int[datossize];

//Creamos una variable para determinar si hay dispositivo conectado
boolean comLista = false;

//Generamos el buffer de la recepción de los paquetes
 byte[] Buffer = new byte[buffersize];
 boolean evento = false;

//Generamos la clase del fichero
 PrintWriter fichero;
 boolean escrito = false;

void setup() {
  //Inicializamos la comunicacion y miramos los elementos conectados en el puerto COM
  String[] lista = Serial.list();

   if(lista.length != 0){
    for(int i = 0; i < lista.length; i++){
    println("Lista[" + i + "]= " + lista[i]);
    }
    //Elegimos el primer elemento de la lista
    // (puede ser otro si tiene mas de un dispositivo conectado)
  puerto = new Serial(this,lista[0], 115200); //Los baudios en el caso del STM32 no son relevantes
  comLista = true;

  puerto.buffer(buffersize);
   }

   else
   //Generamos el fondo
   size(800,600);
   background(0);
  frameRate(30);

}


void draw() {
  //Si se ha producido un evento Serial
  if(evento){
    if(Buffer[buffersize -1] == 0){
     //Creamos un archivo donde guardar los datos
     escrito = false;

    }

    int j = Buffer[buffersize -1];
    if(j >= 0 && j <=4){
      j = j*(buffersize-1)/2;

      for(int i = 0; i < (buffersize-1)/2; i ++){
   //Reconstruimos los datos de nuevo a 16 bits y vamos guardando segun
   // lleguen los nuevos paquetes

        datos[i + j] = (Buffer[2*i] << 8) + (Buffer[2*i +1] & 0xFF);

      }
    }
    if(Buffer[buffersize -1] == 4 && !escrito){
    // Si ha llegado el último paquete
    // Escribimos el fichero con los datos
       fichero = createWriter("datos.txt");
       for(int i = 0; i < datossize; i++) fichero.println(datos[i]);
       escrito = true;
```

```
        fichero.flush();
        fichero.close();
    }
    evento = false;


//Muestra el primer valor de los 3 primeros paquetes para verificar
   println(datos[0 + j] + ", " + datos[1+j] + ", " +datos[2+j] + ", " + Buffer[buffersize -1]);
   }

   background(0);
   float largo = 800;
   float ancho = 600;
   float rlargo = largo - largo*0.1;
   float rancho = ancho - ancho*0.04;
    //Pintamos la lineas de fondo
    grid(largo,ancho, rlargo, rancho);

    //Dibujamos lineas entre cada punto de los datos
    lineas(largo,ancho,rlargo, rancho, datos);

  //Si no hay dispositivo conectado notificar
    if(!comLista){
      textSize(32);
      fill(255, 0, 0);
      textAlign(CENTER);
      text("No hay dispositivo conectado",largo/2, ancho/2 - 10);
    }

}
//Si se produce una llegada de los datos
void serialEvent (Serial c){
  //Leemos los bytes y los guardamos en el Buffer
   Buffer = puerto.readBytes();
   evento = true;

}
//Funcion para generar las lineas del fondo
void grid(float largo, float ancho, float rlargo, float rancho){
  rectMode(CENTER);
  noFill();
  stroke(255);
  strokeWeight(2);

  rect(largo/2, ancho/2, rlargo, rancho);

  for(int i = 0; i < 7; i++){
    float cancho = (ancho - rancho)/2 +(1+i) *rancho/8;
    if(i == 3)    strokeWeight(1.5);
    else    strokeWeight(0.5);
    line((largo - rlargo)/2, cancho ,(largo + rlargo)/2, cancho);
  }
  for(int i = 0; i < 9; i++){
    float clargo = (largo - rlargo)/2 +(1+i) *rlargo/10;
    if(i == 4) strokeWeight(1.5);
    else    strokeWeight(0.5);
    line(clargo, (ancho + rancho)/2, clargo,(ancho - rancho)/2);
   }


       // puntos(largo,ancho,rlargo, rancho, datos);

}
//Funcion para dibujar los puntos tomados
void puntos(float xmax, float ymax, float rx, float ry, int d[]){
  stroke(255,0,0);
  strokeWeight(2);
  int N = d.length;

  for(int i = 0; i < N; i++){
    float x = i * rx/(N - 1) + (xmax - rx)/2;
    float y =  ry * (1 - ((float)d[i]/(0xFFF))) +  (ymax - ry)/2;
    point(x,y);
  }

}
```

```
149 //Por si el valor del numero de datos no se especifica, se dibujaran todos los puntos
150 void lineas(float xmax, float ymax, float rx, float ry, int d[]){
151   lineas(xmax, ymax, rx, ry, d, d.length);
152 }
153 
154 //Funcion para generar las lineas entre los puntos
155 void lineas(float xmax, float ymax, float rx, float ry, int d[], int N){
156   stroke(0,255,0);
157   strokeWeight(1);
158   if(N > d.length) N = d.length;
159   //println(N);
160   for(int i = 0; i < (N - 1); i++){
161     float x1 = i * rx/(N - 1) + (xmax - rx)/2;
162     float y1 =  ry * (1 - ((float)d[i]/(0xFFF))) +  (ymax - ry)/2;
163     float x2 = (i + 1) * rx/(N - 1) + (xmax - rx)/2;
164     float y2 =  ry * (1 - ((float)d[i + 1]/(0xFFF))) + (ymax - ry)/2;
165     line(x1,y1,x2,y2);
166   }
167 
168 }
```

14.4 Código Python

1. Lectura datos STM32

```
#Importamos los modulos que usaremos
import numpy as np
import matplotlib.pyplot as plt

#Leemos el fichero y generamos un array con los datos
fichero = 'datosdemodulados.txt'
with open(fichero) as f:
  lineas = f.readlines()
datos = []

for i in range(len(lineas)):
  datos.append(int(lineas[i]));

datos = np.array(datos)
print(datos[0:3], datos.size)

#Establecemos la frecuancia de muestreo
fm= 9E6/(12.5+71.5)
Tm = 1/ fm
N = datos.size #numero de muestras
Tmax = Tm * N
t = np.arange(0,Tmax,Tm)



#Representamos los datos para comprobar
%matplotlib inline
plt.figure(figsize=(15, 5))
plt.plot(t,datos)
plt.legend(['Datos fichero'])
plt.xlabel("t(s)")
plt.ylabel("Dato")

#Habilitamos la interacción con la gráfica
%matplotlib notebook
#Generamos un array con las frecuencias a representar
f = np.arange(0, fm/2, fm/N)
FFT = (np.abs(np.fft.fft(datos/4096))[0:f.size])
plt.figure(figsize=(9, 6))
plt.loglog(f, FFT)
plt.xlabel("F(Hz)")
plt.ylabel("Datos FFT")

#Escribimos los datos en un fichero en formato
#Convertimos los datos con formato PWL de LTSPICE para usarlo luego
with open('datosdemoduladosPWL.txt', 'w') as writefile:
```

```
    for i in range(len(datos)):
      voltage = 2*3.3*(datos[i]/4096)
      writefile.write(str(t[i]) + " " + str(voltage) + "\n")
```

2. Lectura datos del sensor silla

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Establecemos la ruta de los ficheros a leer
velocidad1_fichero=r"C:\Users\Alexis\Desktop\4° GIEIA\TFG\Datos sil
la\24 junio\vel_asfl.csv"
encoder1_fichero=r"C:\Users\Alexis\Desktop\4° GIEIA\TFG\Datos silla
\24 junio\enc_asfl.csv"

#Generamos un array con los datos de las columnas de interes
df1V = pd.read_csv (velocidad1_fichero)
print(df1V.head())
col_list = ["%time", "field.measured_position"]
df1E = pd.read_csv(encoder1_fichero, usecols=col_list)
print(df1E.head())

# Generamos el array con los datos

velocidad = np.array(df1V)
encoder = np.array(df1E)
deltat_encoder = []
delta_pulsos = []
#Calculamos la variación de los pulsos
for i in range(len(encoder) - 1):
    deltat_encoder.append(encoder[i,0]-encoder[i + 1, 0])
    delta_pulsos.append(encoder[i,1]-encoder[i + 1, 1])
deltat_encoder = np.array(deltat_encoder)
delta_pulsos = np.array(delta_pulsos)
encoder = np.delete(encoder,0,0)
#Rescalamos los datos y nos quedamos con su valor absoluto
# porque el sensor de ultrasonidos no distingue entre la direccion
de la
# velociad
encoder[:,1] = np.abs(delta_pulsos/deltat_encoder)*1E9/8800*0.32*2*
np.pi
t_min= np.min(np.concatenate((velocidad,encoder))[:,0])
velocidad[:,0] -= t_min
encoder[:,0] -= t_min
velocidad[:,0] /= 1E9
encoder[:,0] /= 1E9

#Representamos los datos
```

```
%matplotlib notebook
plt.figure(figsize=(9, 6))

plt.grid( linestyle='--')
plt.plot(velocidad[:,0],velocidad[:,1],encoder[:,0],encoder[:,1])
plt.legend(['Sensor Ultrasonidos','Encoder'])
plt.xlabel("Tiempo(s)", fontsize = 20)
plt.ylabel("Velocidad(m/s)", fontsize = 20)
plt.ylim(0,2)
#plt.savefig(r"C:\Users\Alexis\Desktop\4º GIEIA\TFG\Datos silla\24
junio\entrada.jpg")


#Generamos un array con los datos del encoder, del tamaño
#del array de la velocidad, para poder medir su correlación
#Elegimos el valor del puntos más cercano en el tiempo
#al dato de la velocidad
encoder_reducido = np.array([])
for i in range(len(velocidad)):
    min_arg = np.argmin(np.abs(encoder[:,0]-velocidad[i,0]))
    encoder_reducido = np.append(encoder_reducido, encoder[min_arg]
).reshape(-1,2)
encoder_reducido.shape


%matplotlib inline
plt.figure(figsize=(14, 6))

plt.grid( linestyle='--')
plt.plot(velocidad[:,0],velocidad[:,1],encoder_reducido[:,0],encode
r_reducido[:,1])
plt.legend(['Sensor Ultrasonidos','Encoder'])
plt.xlabel("Tiempo(s)", fontsize = 20)
plt.ylabel("Velocidad(m/s)", fontsize = 20)

print(np.argmin(np.abs(velocidad[:,0]-
100)),np.argmin(np.abs(velocidad[:,0]-170)))
np.corrcoef(velocidad[:,1],encoder_reducido[:,1])
```

14.5 Código WxMaxima

1. Filtro Pasa Banda

```
--> i1:(Vi-Vref)/(R1+1/(s*C1));
    i2:(Vref-Vo)/R2;
    i3:(Vref-Vo)/(1/(s*C2));

--> ec1: i1 = i2 + i3;
    VO:rhs(solve(ec1,Vo)[1]);

--> S1:at(VO,Vref = 0);

--> S2:factor(S1);

--> H:S2/Vi;

--> /*p1 << p2*/
    f1: 1/(2*%pi*C1*R1);
    f2: 1/(2*%pi*C2*R2);
    Amax: R2/R1;
    pars:[R1=1000, R2=10000,C1=2.2E-6,C2=10E-9];
    float(at(f1,pars));
    float(at(f2,pars));
    float(at(Amax,pars));

--> S3:ev(at(H,[C1=2.2E-6,C2=10E-9,R1=1000,R2=10E3]),s=%i*2*%pi*f);

--> log10(x):=float(log(x))/float(log(10));

    plot2d(20*log10(abs(S3)),[f,1,100E3],[logx],[xlabel, "F(Hz)"],[ylabel, "H(dB)"]);

--> S4: 20*log10(abs(S3));
    at(S4,w=1);
```

2. Disparador de Schmitt

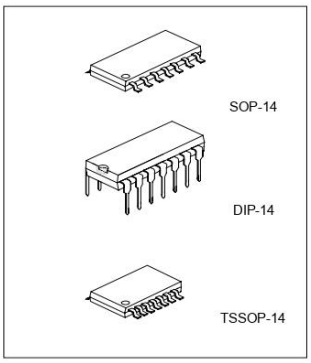
```
--> kill(all);
    ec1:V/R2+(V-Vo)/R3+(V-Vcc)/R1;

--> S:solve(ec1,V);

--> ec1:rhs(at(S,Vo=0)[1]);

--> ec2:rhs(S[1]);

--> S2:solve([VL=ec1,VH=ec2],[R3,R2])[1];
    r3:factor(rhs(S2[1]));
    r2:rhs(S2[2]);

--> pars : [Vo=3.8,VL=2.3,VH=2.7,Vcc=5, R1=1E3];

--> at(r3,pars);
    at(r2,pars);

--> pars2 : [Vo=3.8,R3 = 4270,R2 = 1058,Vcc=5, R1=1E3];
    at(ec1,pars2);
    at(ec2,pars2);
```

1. LM324

LM324

LINEAR INTEGRATED CIRCUIT

QUAD OPERATIONAL AMPLIFIERS



*Pb-free plating product number: LM324L

■ DESCRIPTION

The UTC LM324 consists of four independent, high gain internally frequency compensated operational amplifiers which are designed specifically to operated from a single power supply over a wide voltage range. Operation from split power supplies is also possible. Application areas include transducer amplifier, DC gain blocks and all the conventional OP amp circuits which now can be easily implemented in single power supply system.

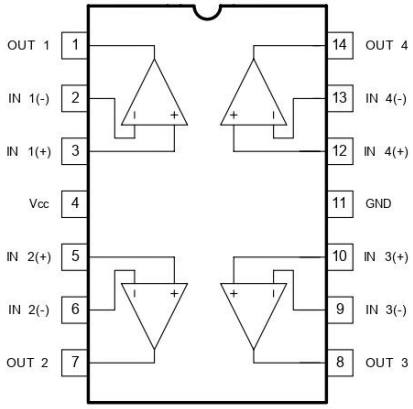
■ FEATURES

*Internally frequency compensated for unity gain.
*Large DC voltage gain :100dB.
*Wide operating supply range (Vcc=3V~32V).
*Input common-mode voltage includes ground.
*Large output voltage swing: From 0V to Vcc-1.5V.
*Power drain suitable for battery operation.

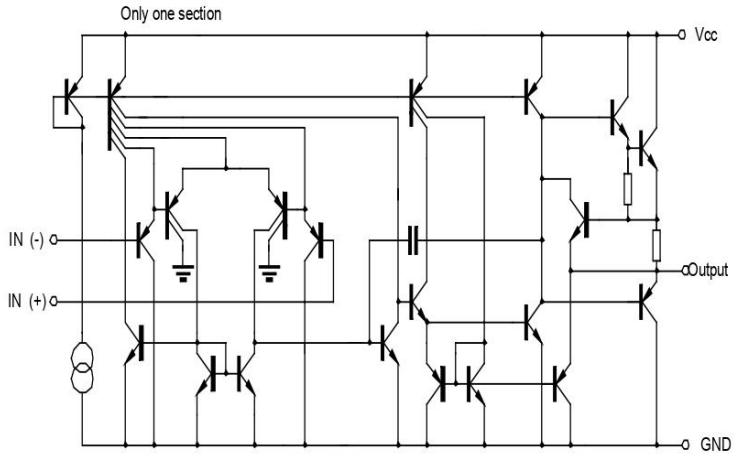
■ ORDERING INFORMATION

Ordering Number		Package	Packing
Normal	Lead Free Plating		
LM324-P14-R	LM324L-P14-R	TSSOP-14	Tape Reel
LM324-P14-T	LM324L-P14-T	TSSOP-14	Tube
LM324-S14-R	LM324L-S14-R	SOP-14	Tape Reel
LM324-S14-T	LM324L-S14-T	SOP-14	Tube
LM324-D14-T	LM324L-D14-T	DIP-14	Tube

■ PIN DESCRIPTION



■ BLOCK DIAGRAM



■ ABSOLUTE MAXIMUM RATINGS

PARAMETER	SYMBOL	RATINGS	UNIT
Supply Voltage	V_{CC}	±18	V
Differential Input Voltage	$V_{I(DIFF)}$	32	V
Input Voltage	V_I	-0.3 ~ +32	V
Power Dissipation	P_D	570	mW
Operating Temperature Range	T_{OPR}	0 ~ +70	°C
Storage Temperature Range	T_{STG}	-40 ~ +150	°C

■ ELECTRICAL CHARACTERISTICS

(V_{CC}=5.0V, All voltage referenced to GND unless otherwise specified.)

PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Input Offset Voltage	V_{IO}	V_{CM}=0V toV_{CC}-1.5V $V_{O(P)}$=1.4V, R_S=0Ω			7.0	mV
Input Offset Current	I_{IO}				50	nA
Input Bias Current	I_{BIAS}				250	nA
Input Common Mode Voltage	$V_{I(R)}$	V_{CC}=30V	0	V_{CC}-1.5		V
Power Supply Current	I_{CC}	R_L=∞, V_{CC}=30V		1.0	3.0	mA
		V_{CC}=5V		0.7	1.2	mA
Large Signal Voltage Gain	G_V	V_{CC}=15V, R_L≧2KΩ $V_{O(P)}$=1V ~ 11V	25	100		V/mV
Output Voltage Swing	$V_{O(H)}$	V_{CC}=30V, R_L=2KΩ	26			V
		V_{CC}=30V, R_L=10KΩ	27	28		V
	$V_{O(L)}$	V_{CC}=5V, R_L>10KΩ		5	20	mV
Common Mode Rejection Ratio	CMRR		65	75		dB
Power Supply Rejection Ratio	PSRR		65	100		dB
Channel Separation	CS	f=1KHZ ~ 20KHZ		120		dB
Short Circuit Current to Ground	I_{SC}			40	60	mA
Output Current	I_{SOURCE}	V_I(+)=1V, V_I(-)=0V V_{CC}=15V, $V_{O(P)}$=2V	20	40		mA
	I_{SINK}	V_I(+)=0V, V_I(-)=1V V_{CC}=15V, V_O(P)=2V	10	13		mA
		V_I(+)=0V, V_I(-)=1V V_{CC}=15V, $V_{O(P)}$=200mV	12	45		mA
Differential Input Voltage	$V_{I(DIFF)}$				V_{CC}	V

■ TYPICAL CHARACTERISTICS

Fig.1 Input Voltage Range

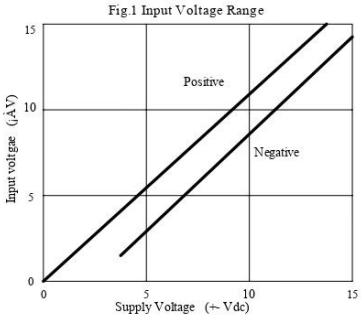


Fig.2 Input Current

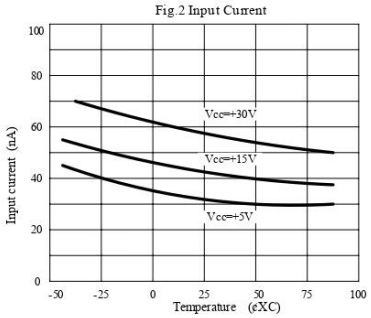


Fig.3 Supply Current

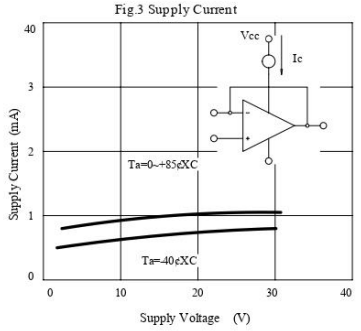


Fig.4 Voltage Gain

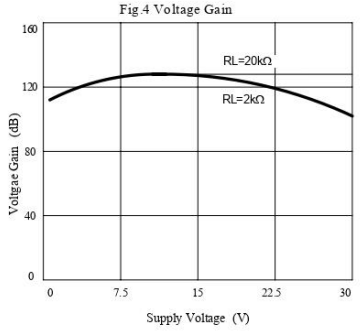


Fig.5 Open Loop Frequency response

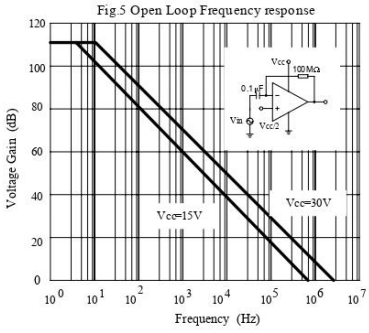
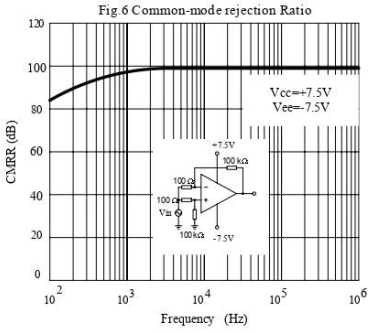


Fig.6 Common-mode rejection Ratio



■ TYPICAL CHARACTERISTICS(cont.)

Fig. 7

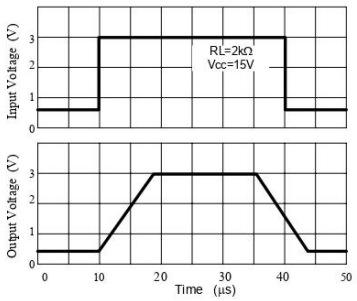


Fig.8 Voltage Follower pulse response (small signal)

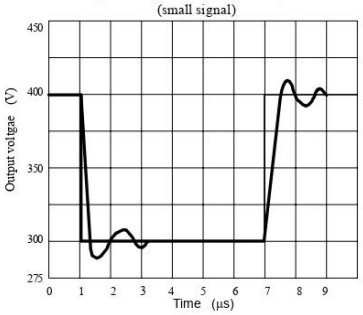


Fig.9 Large signal Frequency Response

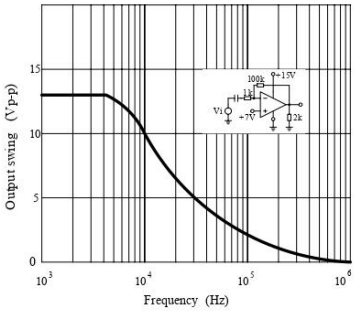


Fig.10 Output Characteristics current sourcing

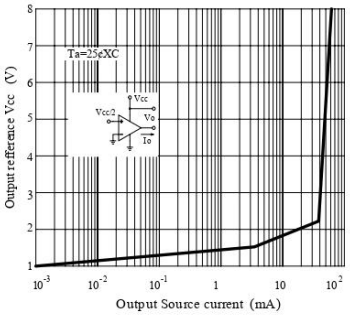


Fig.11 Output Characteristics Current sinking

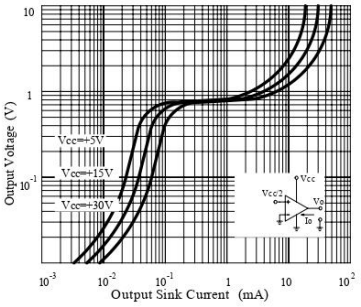
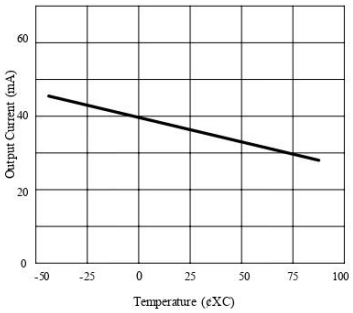


Fig.12 Current Limiting



2. CD4051b

CMOS Analog Multiplexers/Demultiplexers with Logic Level Conversion

The CD4051B, CD4052B, and CD4053B analog multiplexers are digitally-controlled analog switches having low ON impedance and very low OFF leakage current. Control of analog signals up to $20V_{P-P}$ can be achieved by digital signal amplitudes of 4.5V to 20V (if V_{DD}-V_{SS} = 3V, a V_{DD}-V_{EE} of up to 13V can be controlled; for V_{DD}-V_{DD} level differences above 13V, a V_{DD}-V_{DD} of at least 4.5V is required). For example, if V_{DD} = +4.5V, V_{DD} = 0V, and V_{DD} = -13.5V, analog signals from -13.5V to +4.5V can be controlled by digital inputs of 0V to 5V. These multiplexer circuits dissipate extremely low quiescent power over the full V_{DD}-V_{DD} and V_{DD}-V_{DD} supply-voltage ranges, independent of the logic state of the control signals. When a logic "1" is present at the inhibit input terminal, all channels are off.

The CD4051B is a single 8-Channel multiplexer having three binary control inputs, A, B, and C, and an inhibit input. The three binary signals select 1 of 8 channels to be turned on, and connect one of the 8 inputs to the output.

The CD4052B is a differential 4-Channel multiplexer having two binary control inputs, A and B, and an inhibit input. The two binary input signals select 1 of 4 pairs of channels to be turned on and connect the analog inputs to the outputs.

The CD4053B is a triple 2-Channel multiplexer having three separate digital control inputs, A, B, and C, and an inhibit input. Each control input selects one of a pair of channels which are connected in a single-pole, double-throw configuration.

When these devices are used as demultiplexers, the "CHANNEL IN/OUT" terminals are the outputs and the "COMMON OUT/IN" terminals are the inputs.

Ordering Information

PART NUMBER	TEMP. RANGE (^oC)	PACKAGE	PKG. NO.
CD4051BF, CD4052BF, CD4053BF	-55 to 125	16 Ld CERDIP	F16.3
CD4051BE, CD4052BE, CD4053BE	-55 to 125	16 Ld PDIP	E16.3
CD4051BM, CD4052BM, CD4053BM	-55 to 125	16 Ld SOIC	M16.15

Features

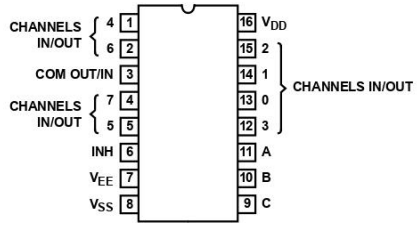
- Wide Range of Digital and Analog Signal Levels
  - Digital . . . . . 3V to 20V
  - Analog. . . . . $\leq 20V_{P-P}$
- Low ON Resistance, 125Ω (Typ) Over $15V_{P-P}$ Signal Input Range for V_{DD}-V_{EE} = 18V
- High OFF Resistance, Channel Leakage of ±100pA (Typ) at V_{DD}-V_{EE} = 18V
- Logic-Level Conversion for Digital Addressing Signals of 3V to 20V (V_{DD}-V_{SS} = 3V to 20V) to Switch Analog Signals to $20V_{P-P}$ (V_{DD}-V_{EE} = 20V)
- Matched Switch Characteristics, r_{ON} = 5Ω (Typ) for V_{DD}-V_{EE} = 15V
- Very Low Quiescent Power Dissipation Under All Digital-Control Input and Supply Conditions, 0.2μW (Typ) at V_{DD}-V_{SS} = V_{DD}-V_{EE} = 10V
- Binary Address Decoding on Chip
- 5V, 10V and 15V Parametric Ratings
- 10% Tested for Quiescent Current at 20V
- Maximum Input Current of 1μA at 18V Over Full Package Temperature Range, 100nA at 18V and 25^oC
- Break-Before-Make Switching Eliminates Channel Overlap

Applications

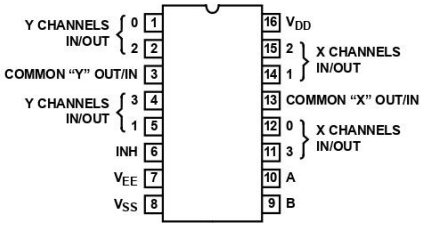
- Analog and Digital Multiplexing and Demultiplexing
- A/D and D/A Conversion
- Signal Gating

Pinouts

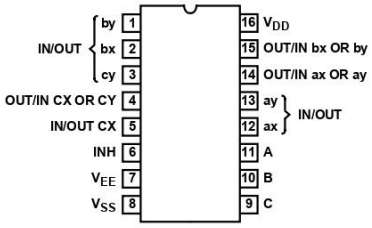
CD4051B (PDIP, CERDIP, SOIC)
TOP VIEW



CD4052B (PDIP, CERDIP)
TOP VIEW

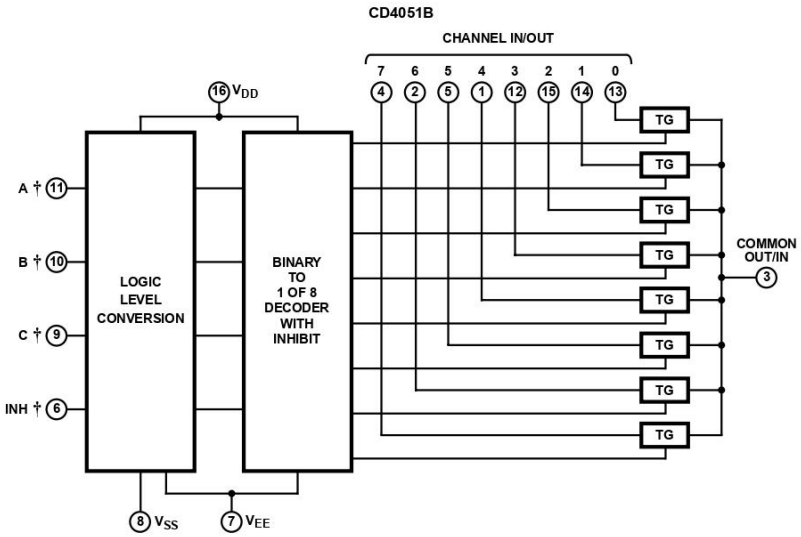


CD4053B (PDIP, CERDIP)
TOP VIEW



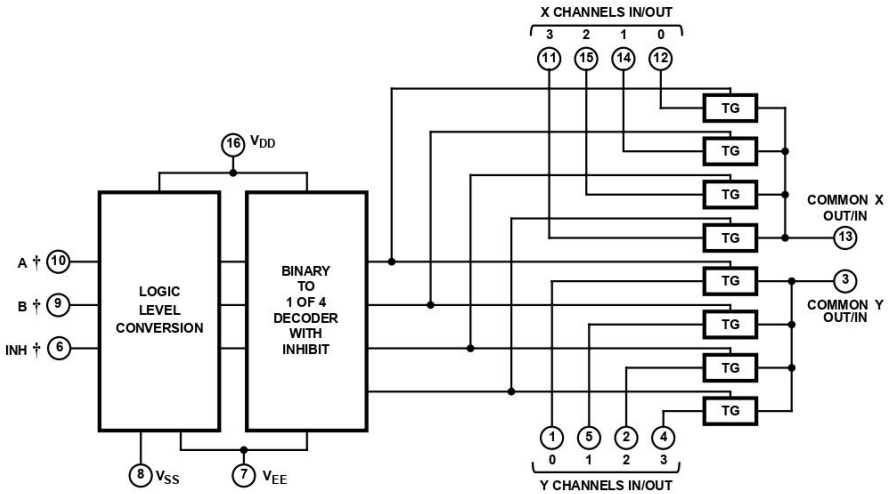
Functional Block Diagrams

CD4051B

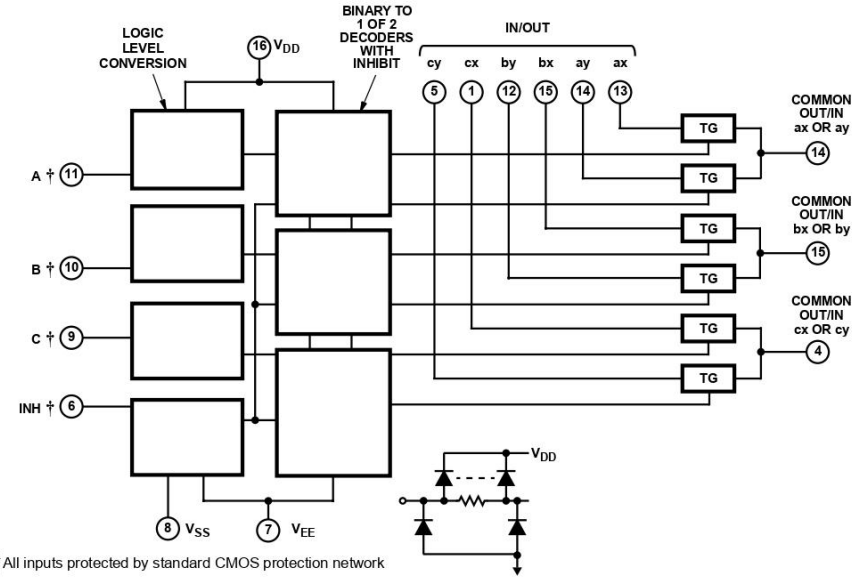


Functional Block Diagrams (Continued)

CD4052B



CD4053B



† All inputs protected by standard CMOS protection network

TRUTH TABLES

INPUT STATES				
INHIBIT	**C**	**B**	**A**	**"ON" CHANNEL(S)**
CD4051B				
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	X	X	X	None
CD4052B				
INHIBIT		**B**	**A**	
0		0	0	0x, 0y
0		0	1	1x, 1y
0		1	0	2x, 2y
0		1	1	3x, 3y
1		X	X	None
CD4053B				
INHIBIT		**A OR B OR C**		
0		0		ax or bx or cx
0		1		ay or by or cy
1		X		None

X = Don't Care

Absolute Maximum Ratings

Supply Voltage (V+ to V-)
Voltages Referenced to V_{SS} Terminal . . . . . -0.5V to 20V
DC Input Voltage Range . . . . . -0.5V to V_{DD} +0.5V
DC Input Current, Any One Input. . . . . ±10mA

Operating Conditions

Temperature Range . . . . . -55^oC to 125^oC

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA} ($^oC/W$)	θ_{JC} ($^oC/W$)
PDIP Package	90	N/A
CERDIP Package	115	45
SOIC Package	115	N/A

Maximum Junction Temperature (Ceramic Package) . . . . . 175^oC
Maximum Junction Temperature (Plastic Package) . . . . . 150^oC
Maximum Storage Temperature Range . . . . . -65^oC to 150^oC
Maximum Lead Temperature (Soldering 10s) . . . . . 265^oC
(SOIC - Lead Tips Only)

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

Common Conditions Here: If Whole Table is For the Full Temp. Range, V_{SUPPLY} = ±5V, A_V = +1, R_L = 100Ω, Unless Otherwise Specified (Note 3)

PARAMETER	CONDITIONS V_{IS} (V)	V_{EE} (V)	V_{SS} (V)	V_{DD} (V)	LIMITS AT INDICATED TEMPERATURES (^oC) -55	-40	85	125	25 MIN	25 TYP	25 MAX	UNITS
SIGNAL INPUTS (V_{IS}) AND OUTPUTS (V_{OS})												
Quiescent Device Current, I_{DD} Max	-	-	-	5	5	5	150	150	-	0.04	5	µA
	-	-	-	10	10	10	300	300	-	0.04	10	µA
	-	-	-	15	20	20	600	600	-	0.04	20	µA
	-	-	-	20	100	100	3000	3000	-	0.08	100	µA
Drain to Source ON Resistance r_{ON} Max $0 \le V_{IS} \le V_{DD}$	-	0	0	5	800	850	1200	1300	-	470	1050	Ω
	-	0	0	10	310	330	520	550	-	180	400	Ω
	-	0	0	15	200	210	300	320	-	125	240	Ω
Change in ON Resistance (Between Any Two Channels), Δr_{ON}	-	0	0	5	-	-	-	-	-	15	-	Ω
	-	0	0	10	-	-	-	-	-	10	-	Ω
	-	0	0	15	-	-	-	-	-	5	-	Ω
OFF Channel Leakage Current: Any Channel OFF (Max) or ALL Channels OFF (Common OUT/IN) (Max)	-	0	0	18	±100 (Note 2)		±1000 (Note 2)		-	±0.01	±100 (Note 2)	µA
Capacitance:	-	-5	5-	5								
Input, C_{IS}					-	-	-	-	-	5	-	pF
Output, C_{OS}												
CD4051					-	-	-	-	-	30	-	pF
CD4052					-	-	-	-	-	18	-	pF
CD4053					-	-	-	-	-	9	-	pF
Feedthrough C_{IOS}					-	-	-	-	-	0.2	-	pF
Propagation Delay Time (Signal Input to Output	V_{DD}	R_L = 200kΩ, C_L = 50pF, t_r, t_f = 20ns		5	-	-	-	-	-	30	60	ns
				10	-	-	-	-	-	15	30	ns
				15	-	-	-	-	-	10	20	ns

Electrical Specifications Common Conditions Here: If Whole Table is For the Full Temp. Range, V_{SUPPLY} = ±5V, A_V = +1, R_L = 100Ω, Unless Otherwise Specified **(Continued)** (Note 3)

PARAMETER	CONDITIONS				LIMITS AT INDICATED TEMPERATURES (°C)							UNITS
									25			
	V_{IS} (V)	V_{EE} (V)	V_{SS} (V)	V_{DD} (V)	-55	-40	85	125	MIN	TYP	MAX	
CONTROL (ADDRESS OR INHIBIT), V_C												
Input Low Voltage, V_{IL}, Max	V_{IL} = V_{DD} through 1kΩ; V_{IH} = V_{DD} through 1kΩ	V_{EE} = V_{SS}, R_L = 1kΩ to V_{SS}, I_{IS} < 2μA on All OFF Channels		5	1.5	1.5	1.5	1.5	-	-	1.5	V
				10	3	3	3	3	-	-	3	V
				15	4	4	4	4	-	-	4	V
Input High Voltage, V_{IH}, Min				5	3.5	3.5	3.5	3.5	3.5	-	-	V
				10	7	7	7	7	7	-	-	V
				15	11	11	11	11	11	-	-	V
Input Current, I_{IN} (Max)	V_{IN} = 0, 18			18	±0.1	±0.1	±1	±1	-	$±10^{-5}$	±0.1	μA
Propagation Delay Time: Address-to-Signal OUT (Channels ON or OFF) See Figures 10, 11, 14	t_r, t_f = 20ns, C_L = 50pF, R_L = 10kΩ	0	0	5	-	-	-	-	-	450	720	ns
		0	0	10	-	-	-	-	-	160	320	ns
		0	0	15	-	-	-	-	-	120	240	ns
		-5	0	5	-	-	-	-	-	225	450	ns
Propagation Delay Time: Inhibit-to-Signal OUT (Channel Turning ON) See Figure 11	t_r, t_f = 20ns, C_L = 50pF, R_L = 1kΩ	0	0	5	-	-	-	-	-	400	720	ns
		0	0	10	-	-	-	-	-	160	320	ns
		0	0	15	-	-	-	-	-	120	240	ns
		-10	0	5	-	-	-	-	-	200	400	ns
Propagation Delay Time: Inhibit-to-Signal OUT (Channel Turning OFF) See Figure 15	t_r, t_f = 20ns, C_L = 50pF, R_L = 10kΩ	0	0	5	-	-	-	-	-	200	450	ns
		0	0	10	-	-	-	-	-	90	210	ns
		0	0	15	-	-	-	-	-	70	160	ns
		-10	0	5	-	-	-	-	-	130	300	ns
Input Capacitance, C_{IN} (Any Address or Inhibit Input)					-	-	-	-	-	5	7.5	pF

NOTE:

2. Determined by minimum feasible leakage measurement for automatic testing.

Electrical Specifications

PARAMETER	TEST CONDITIONS					LIMITS	UNITS
	V_{IS} (V)	V_{DD} (V)	R_L (kΩ)			TYP	
Cutoff (-3dB) Frequency Channel ON (Sine Wave Input)	5 (Note 3)	10	1	V_{OS} at Common OUT/IN	CD4053	30	MHz
	V_{EE} = V_{SS}, $20\text{Log}\frac{V_{OS}}{V_{IS}} = -3\text{dB}$				CD4052	25	MHz
					CD4051	20	MHz
				V_{OS} at Any Channel		60	MHz

Electrical Specifications

PARAMETER	TEST CONDITIONS					LIMITS	UNITS
	V_{IS} (V)	V_{DD} (V)	R_L (kΩ)			TYP	
Total Harmonic Distortion, THD	2 (Note 3)	5	10			0.3	%
	3 (Note 3)	10				0.2	%
	5 (Note 3)	15				0.12	%
	$V_{EE} = V_{SS}$, f_{IS} = 1kHz Sine Wave						%
-40dB Feedthrough Frequency (All Channels OFF)	5 (Note 3)	10	1	V_{OS} at Common OUT/IN	CD4053	8	MHz
	$V_{EE} = V_{SS}$, $20\text{Log}\frac{V_{OS}}{V_{IS}} = -40\text{dB}$				CD4052	10	MHz
					CD4051	12	MHz
				V_{OS} at Any Channel		8	MHz
-40dB Signal Crosstalk Frequency	5 (Note 3)	10	1	Between Any 2 Channels		3	MHz
	$V_{EE} = V_{SS}$, $20\text{Log}\frac{V_{OS}}{V_{IS}} = -40\text{dB}$			Between Sections, CD4052 Only	Measured on Common	6	MHz
					Measured on Any Channel	10	MHz
				Between Any Two Sections, CD4053 Only	In Pin 2, Out Pin 14	2.5	MHz
					In Pin 15, Out Pin 14	6	MHz
Address-or-Inhibit-to-Signal Crosstalk	-	10	10 (Note 4)			65	mV_{PEAK}
	$V_{EE} = 0$, $V_{SS} = 0$, t_r, t_f = 20ns, $V_{CC} = V_{DD} - V_{SS}$ (Square Wave)					65	mV_{PEAK}

NOTES:

3. Peak-to-Peak voltage symmetrical about $\frac{V_{DD} - V_{EE}}{2}$
4. Both ends of channel.

Typical Performance Curves

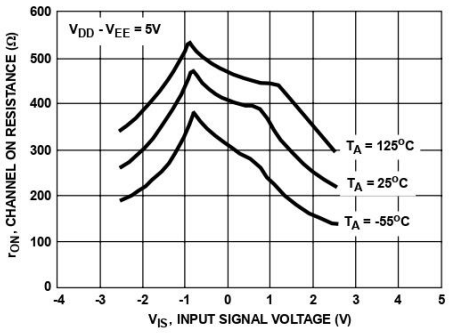


FIGURE 1. CHANNEL ON RESISTANCE vs INPUT SIGNAL VOLTAGE (ALL TYPES)

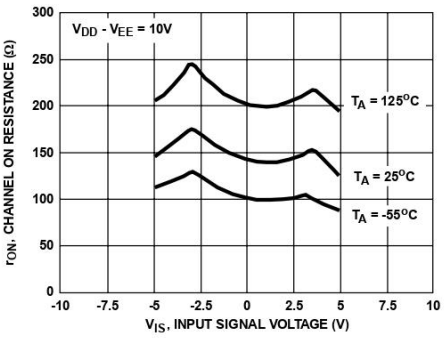


FIGURE 2. CHANNEL ON RESISTANCE vs INPUT SIGNAL VOLTAGE (ALL TYPES)

Typical Performance Curves (Continued)

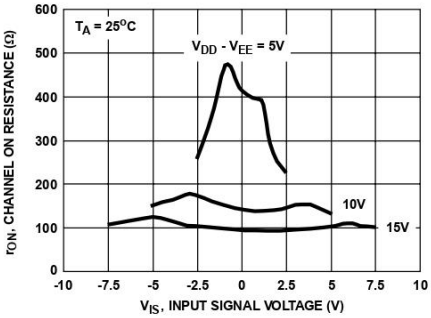


FIGURE 3. CHANNEL ON RESISTANCE vs INPUT SIGNAL VOLTAGE (ALL TYPES)

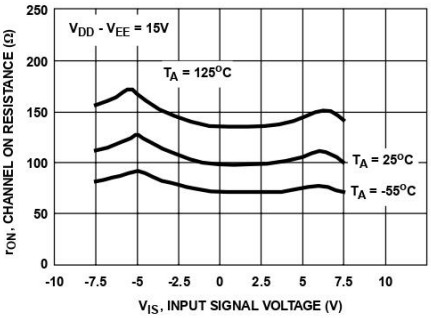


FIGURE 4. CHANNEL ON RESISTANCE vs INPUT SIGNAL VOLTAGE (ALL TYPES)

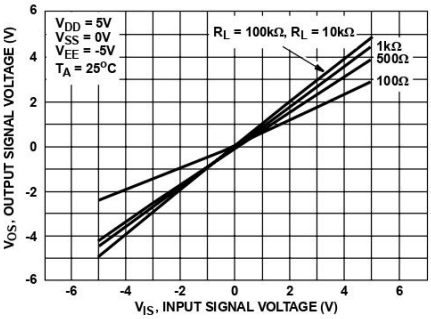


FIGURE 5. ON CHARACTERISTICS FOR 1 OF 8 CHANNELS (CD4051B)

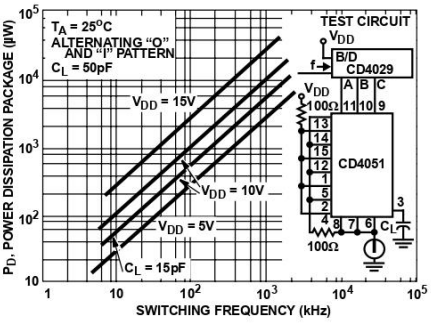


FIGURE 6. DYNAMIC POWER DISSIPATION vs SWITCHING FREQUENCY (CD4051B)

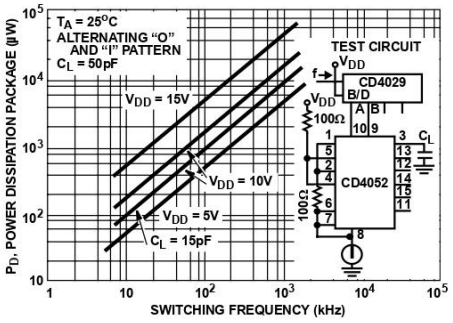


FIGURE 7. DYNAMIC POWER DISSIPATION vs SWITCHING FREQUENCY (CD4052B)

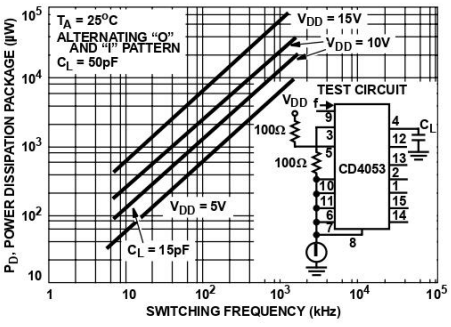
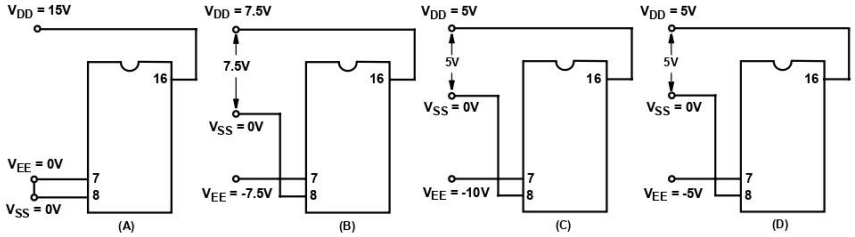


FIGURE 8. DYNAMIC POWER DISSIPATION vs SWITCHING FREQUENCY (CD4053B)

Test Circuits and Waveforms



NOTE: The ADDRESS (digital-control inputs) and INHIBIT logic levels are: "0" = V_{SS} and "1" = V_{DD}. The analog signal (through the TG) may swing from V_{EE} to V_{DD}.

FIGURE 9. TYPICAL BIAS VOLTAGES

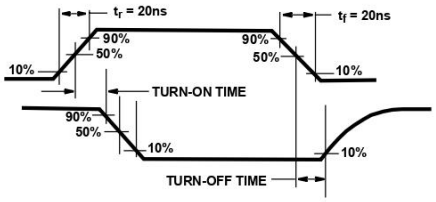


FIGURE 10. WAVEFORMS, CHANCEL BEING TURNED ON (R_L = 1kΩ)

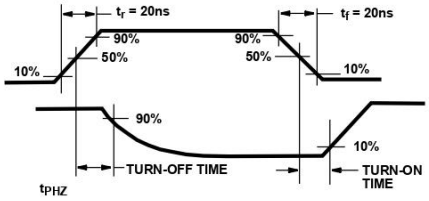


FIGURE 11. WAVEFORMS, CHANNEL BEING TURNED OFF (R_L = 1kΩ)

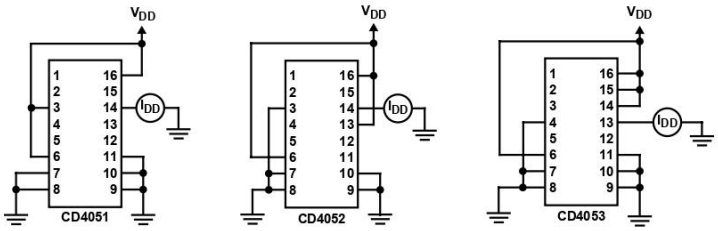


FIGURE 12. OFF CHANNEL LEAKAGE CURRENT - ANY CHANNEL OFF

Test Circuits and Waveforms (Continued)

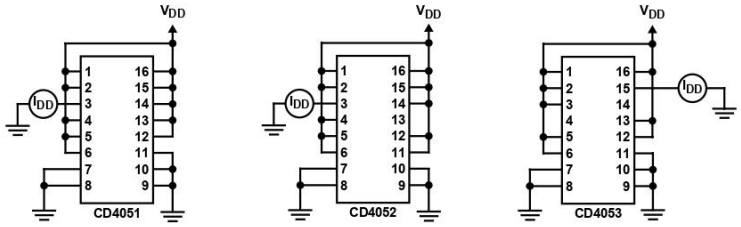


FIGURE 13. OFF CHANNEL LEAKAGE CURRENT - ALL CHANNELS OFF

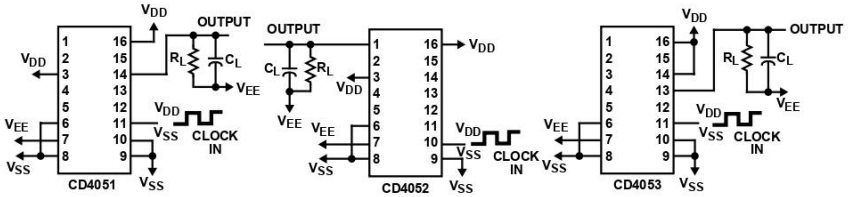


FIGURE 14. PROPAGATION DELAY - ADDRESS INPUT TO SIGNAL OUTPUT

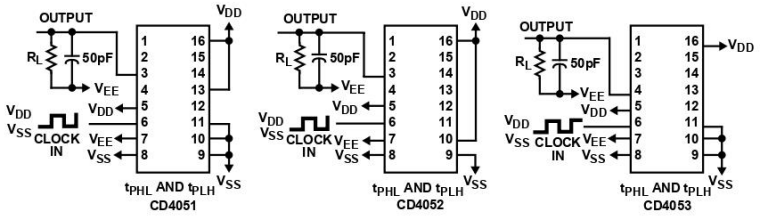


FIGURE 15. PROPAGATION DELAY - INHIBIT INPUT TO SIGNAL OUTPUT

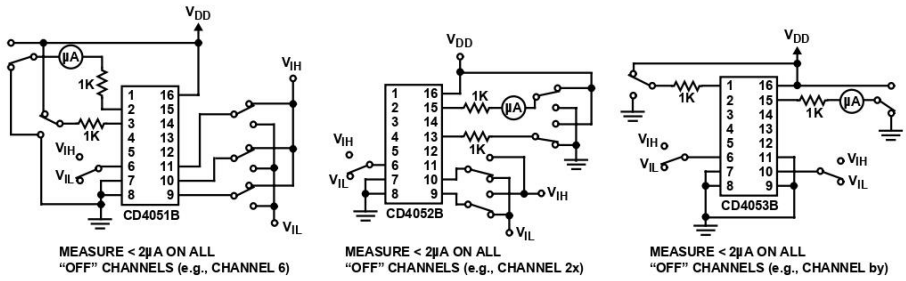


FIGURE 16. INPUT VOLTAGE TEST CIRCUITS (NOISE IMMUNITY)

Test Circuits and Waveforms (Continued)

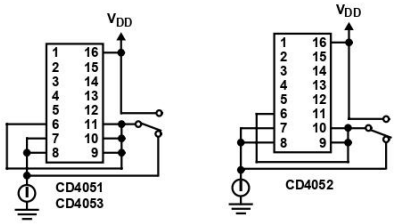


FIGURE 17. QUIESCENT DEVICE CURRENT

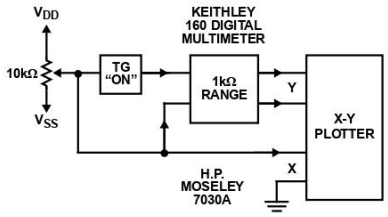


FIGURE 18. CHANNEL ON RESISTANCE MEASUREMENT CIRCUIT

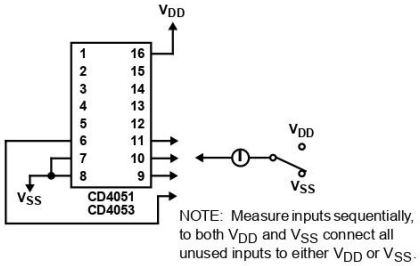


FIGURE 19. INPUT CURRENT

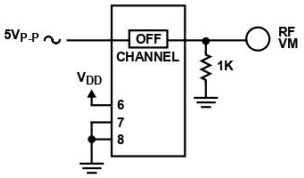


FIGURE 20. FEEDTHROUGH (ALL TYPES)

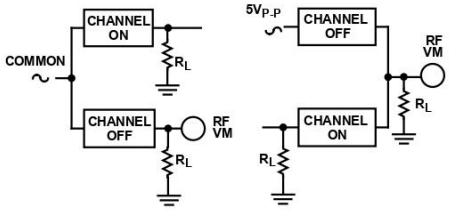


FIGURE 21. CROSSTALK BETWEEN ANY TWO CHANNELS (ALL TYPES)

FIGURE 22. CROSSTALK BETWEEN DUALS OR TRIPLETS (CD4052B, CD4053B)

Test Circuits and Waveforms (Continued)

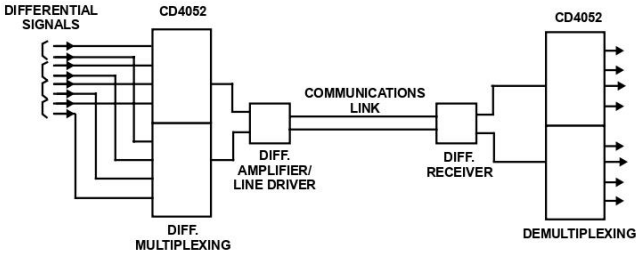


FIGURE 23. TYPICAL TIME-DIVISION APPLICATION OF THE CD4052B

Special Considerations

In applications where separate power sources are used to drive V_{DD} and the signal inputs, the V_{DD} current capability should exceed V_{DD}/R_L (R_L = effective external load). This provision avoids permanent current flow or clamp action on the V_{DD} supply when power is applied or removed from the CD4051B, CD4052B or CD4053B.

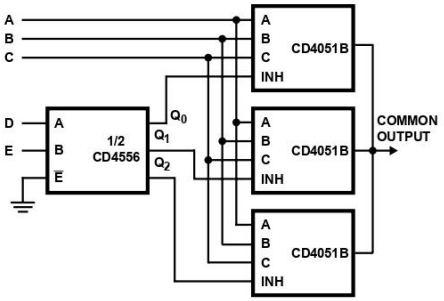
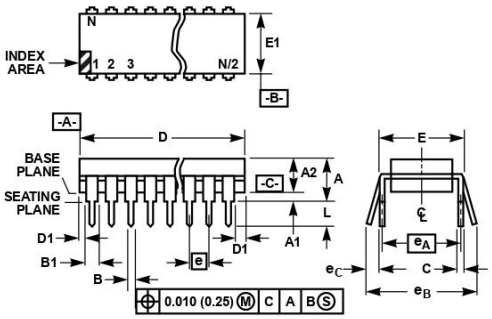


FIGURE 24. 24-TO-1 MUX ADDRESSING

Dual-In-Line Plastic Packages (PDIP)



NOTES:

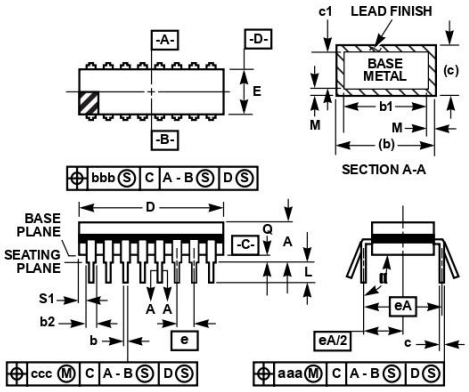
1. Controlling Dimensions: INCH. In case of conflict between English and Metric dimensions, the inch dimensions control.
2. Dimensioning and tolerancing per ANSI Y14.5M-1982.
3. Symbols are defined in the "MO Series Symbol List" in Section 2.2 of Publication No. 95.
4. Dimensions A, A1 and L are measured with the package seated in JEDEC seating plane gauge GS-3.
5. D, D1, and E1 dimensions do not include mold flash or protrusions. Mold flash or protrusions shall not exceed 0.010 inch (0.25mm).
6. E and e_A are measured with the leads constrained to be perpendicular to datum -C-.
7. e_B and e_C are measured at the lead tips with the leads unconstrained. e_C must be zero or greater.
8. B1 maximum dimensions do not include dambar protrusions. Dambar protrusions shall not exceed 0.010 inch (0.25mm).
9. N is the maximum number of terminal positions.
10. Corner leads (1, N, N/2 and N/2 + 1) for E8.3, E16.3, E18.3, E28.3, E42.6 will have a B1 dimension of 0.030 - 0.045 inch (0.76 - 1.14mm).

E16.3 (JEDEC MS-001-BB ISSUE D)
16 LEAD DUAL-IN-LINE PLASTIC PACKAGE

	INCHES		MILLIMETERS		
SYMBOL	MIN	MAX	MIN	MAX	NOTES
A	-	0.210	-	5.33	4
A1	0.015	-	0.39	-	4
A2	0.115	0.195	2.93	4.95	-
B	0.014	0.022	0.356	0.558	-
B1	0.045	0.070	1.15	1.77	8, 10
C	0.008	0.014	0.204	0.355	-
D	0.735	0.775	18.66	19.68	5
D1	0.005	-	0.13	-	5
E	0.300	0.325	7.62	8.25	6
E1	0.240	0.280	6.10	7.11	5
e	0.100 BSC		2.54 BSC		-
e_A	0.300 BSC		7.62 BSC		6
e_B	-	0.430	-	10.92	7
L	0.115	0.150	2.93	3.81	4
N	16		16		9

Rev. 0 12/93

Ceramic Dual-In-Line Frit Seal Packages (CERDIP)



NOTES:

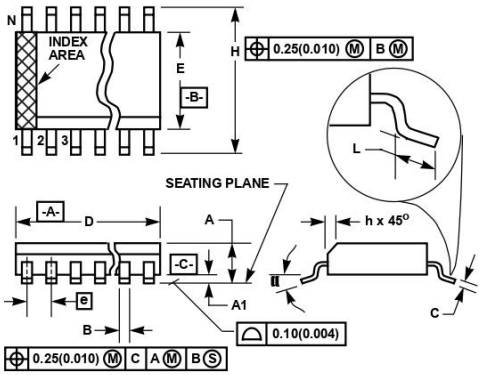
1. Index area: A notch or a pin one identification mark shall be located adjacent to pin one and shall be located within the shaded area shown. The manufacturer's identification shall not be used as a pin one identification mark.
2. The maximum limits of lead dimensions b and c or M shall be measured at the centroid of the finished lead surfaces, when solder dip or tin plate lead finish is applied.
3. Dimensions b1 and c1 apply to lead base metal only. Dimension M applies to lead plating and finish thickness.
4. Corner leads (1, N, N/2, and N/2+1) may be configured with a partial lead paddle. For this configuration dimension b3 replaces dimension b2.
5. This dimension allows for off-center lid, meniscus, and glass overrun.
6. Dimension Q shall be measured from the seating plane to the base plane.
7. Measure dimension S1 at all four corners.
8. N is the maximum number of terminal positions.
9. Dimensioning and tolerancing per ANSI Y14.5M - 1982.
10. Controlling dimension: INCH.

F16.3 MIL-STD-1835 GDIP1-T16 (D-2, CONFIGURATION A)
16 LEAD CERAMIC DUAL-IN-LINE FRIT SEAL PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	-	0.200	-	5.08	-
b	0.014	0.026	0.36	0.66	2
b1	0.014	0.023	0.36	0.58	3
b2	0.045	0.065	1.14	1.65	-
b3	0.023	0.045	0.58	1.14	4
c	0.008	0.018	0.20	0.46	2
c1	0.008	0.015	0.20	0.38	3
D	-	0.840	-	21.34	5
E	0.220	0.310	5.59	7.87	5
e	0.100 BSC		2.54 BSC		-
eA	0.300 BSC		7.62 BSC		-
eA/2	0.150 BSC		3.81 BSC		-
L	0.125	0.200	3.18	5.08	-
Q	0.015	0.060	0.38	1.52	6
S1	0.005	-	0.13	-	7
α	90^o	105^o	90^o	105^o	-
aaa	-	0.015	-	0.38	-
bbb	-	0.030	-	0.76	-
ccc	-	0.010	-	0.25	-
M	-	0.0015	-	0.038	2, 3
N	16		16		8

Rev. 0 4/94

Small Outline Plastic Packages (SOIC)



NOTES:

1. Symbols are defined in the "MO Series Symbol List" in Section 2.2 of Publication Number 95.
2. Dimensioning and tolerancing per ANSI Y14.5M-1982.
3. Dimension "D" does not include mold flash, protrusions or gate burrs. Mold flash, protrusion and gate burrs shall not exceed 0.15mm (0.006 inch) per side.
4. Dimension "E" does not include interlead flash or protrusions. Interlead flash and protrusions shall not exceed 0.25mm (0.010 inch) per side.
5. The chamfer on the body is optional. If it is not present, a visual index feature must be located within the crosshatched area.
6. "L" is the length of terminal for soldering to a substrate.
7. "N" is the number of terminal positions.
8. Terminal numbers are shown for reference only.
9. The lead width "B", as measured 0.36mm (0.014 inch) or greater above the seating plane, shall not exceed a maximum value of 0.61mm (0.024 inch).
10. Controlling dimension: MILLIMETER. Converted inch dimensions are not necessarily exact.

M16.15 (JEDEC MS-012-AC ISSUE C)
16 LEAD NARROW BODY SMALL OUTLINE PLASTIC PACKAGE

SYMBOL	INCHES		MILLIMETERS		NOTES
	MIN	MAX	MIN	MAX	
A	0.0532	0.0688	1.35	1.75	-
A1	0.0040	0.0098	0.10	0.25	-
B	0.013	0.020	0.33	0.51	9
C	0.0075	0.0098	0.19	0.25	-
D	0.3859	0.3937	9.80	10.00	3
E	0.1497	0.1574	3.80	4.00	4
e	0.050 BSC		1.27 BSC		-
H	0.2284	0.2440	5.80	6.20	-
h	0.0099	0.0196	0.25	0.50	5
L	0.016	0.050	0.40	1.27	6
N	16		16		7
α	0°	8°	0°	8°	-

Rev. 0 12/93

3. Arduino UNO

8-bit AVR Microcontrollers

ATmega328/P

DATASHEET SUMMARY

Introduction

The Atmel® picoPower® ATmega328/P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328/P achieves throughputs close to 1MIPS per MHz. This empowers system designer to optimize the device for power consumption versus processing speed.

Feature

High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family

- Advanced RISC Architecture
  - 131 Powerful Instructions
  - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 32KBytes of In-System Self-Programmable Flash program Memory
  - 1KBytes EEPROM
  - 2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data Retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® Library Support
  - Capacitive Touch Buttons, Sliders and Wheels
  - QTouch and QMatrix® Acquisition
  - Up to 64 sense channels

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Two Master/Slave SPI Serial Interface
  - One Programmable Serial USART
  - One Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - One On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 105°C
- Speed Grade:
  - 0 - 4MHz @ 1.8 - 5.5V
  - 0 - 10MHz @ 2.7 - 5.5V
  - 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

Table of Contents

1. Description

The Atmel AVR® core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega328/P provides the following features: 32Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 1Kbytes EEPROM, 2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 1 serial programmable USARTs , 1 byte-oriented 2-wire Serial Interface (I2C), a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages) , a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main oscillator and the asynchronous timer continue to run.

Atmel offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS™) technology for unambiguous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328/P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega328/P is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2. Configuration Summary

Features	ATmega328/P
Pin Count	28/32
Flash (Bytes)	32K
SRAM (Bytes)	2K
EEPROM (Bytes)	1K
Interrupt Vector Size (instruction word/vector)	1/1/2
General Purpose I/O Lines	23
SPI	2
TWI (I^2C)	1
USART	1
ADC	10-bit 15kSPS
ADC Channels	8
8-bit Timer/Counters	2
16-bit Timer/Counters	1

and support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In , there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

3. Ordering Information

3.1. ATmega328

Speed [MHz][3]	Power Supply [V]	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328-AU ATmega328-AUR[5] ATmega328-MMH[4] ATmega328-MMHR[4][5] ATmega328-MU ATmega328-MUR[5] ATmega328-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. V_{CC}
4. Tape & Reel.
5. NiPdAu Lead Finish.

Package Type	
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/ MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/ MLF)
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)

3.2. ATmega328P

Speed [MHz][3]	Power Supply [V]	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328P-AU ATmega328P-AUR[5] ATmega328P-MMH[4] ATmega328P-MMHR[4][5] ATmega328P-MU ATmega328P-MUR[5] ATmega328P-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega328P-AN ATmega328P-ANR[5] ATmega328P-MN ATmega328P-MNR[5] ATmega328P-PN	32A 32A 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

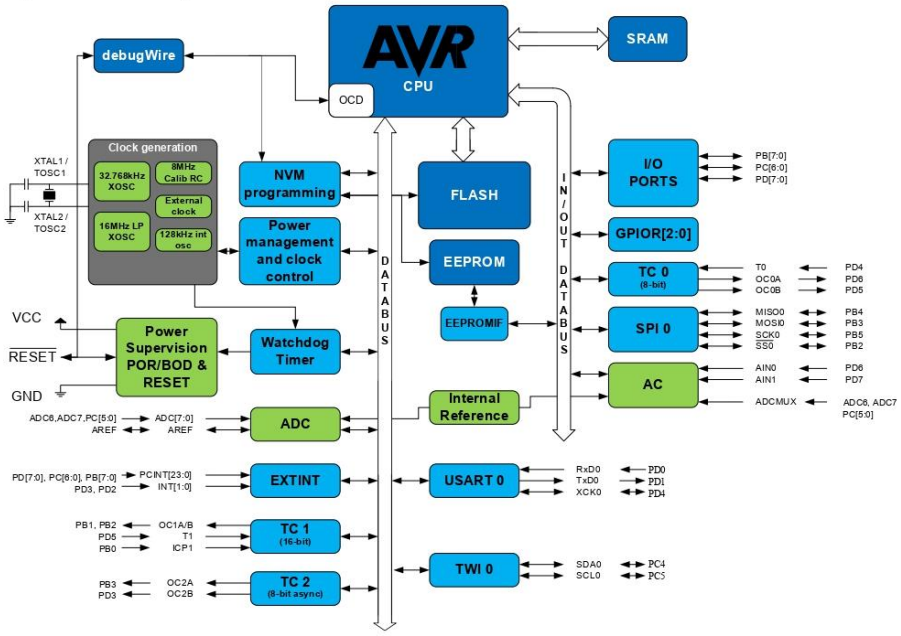
Note:

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. V_{CC}
4. Tape & Reel.
5. NiPdAu Lead Finish.

Package Type	
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)

4. Block Diagram

Figure 4-1. Block Diagram



5. Pin Configurations

5.1. Pin-out

Figure 5-1. 28-pin PDIP

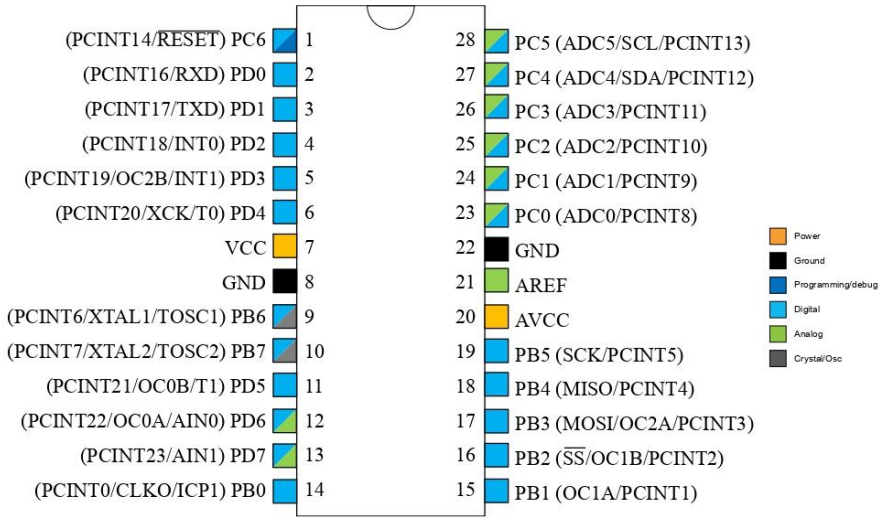


Figure 5-2. 28-pin MLF Top View

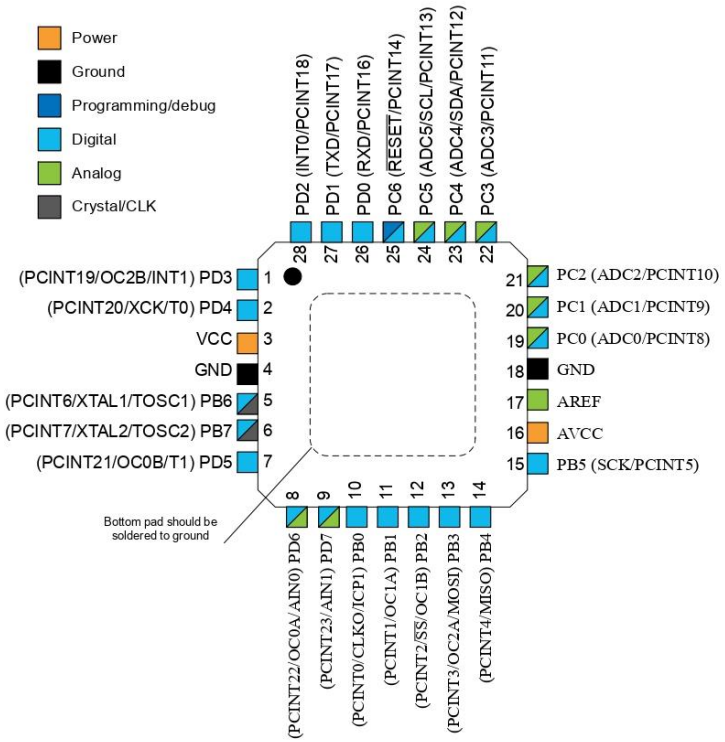


Figure 5-3. 32-pin TQFP Top View

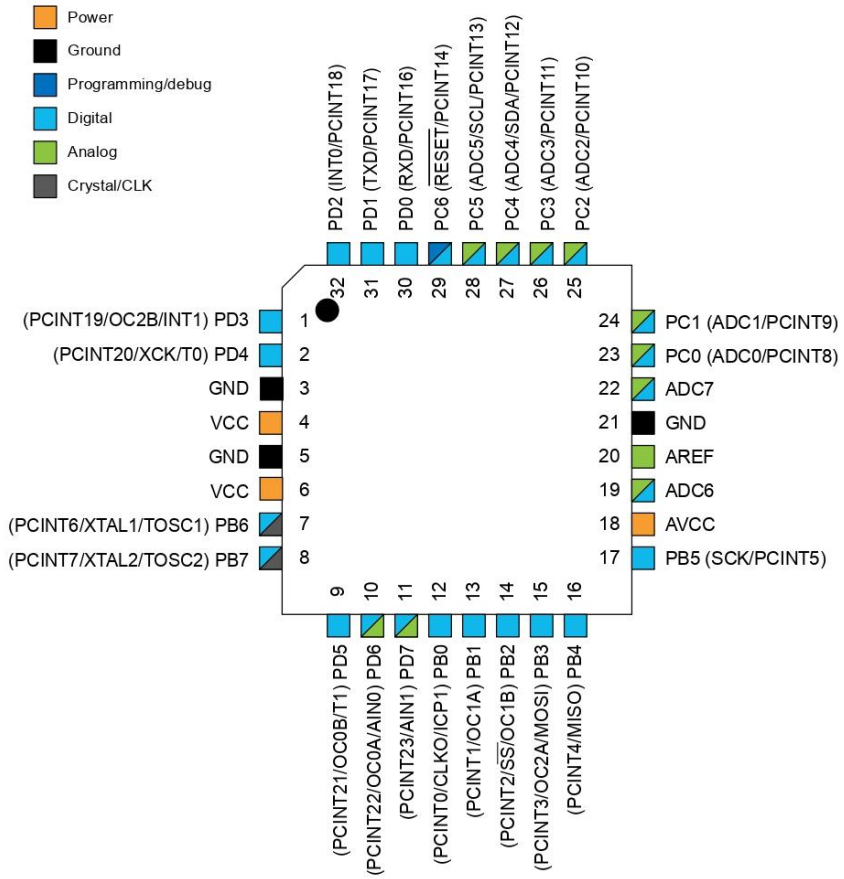
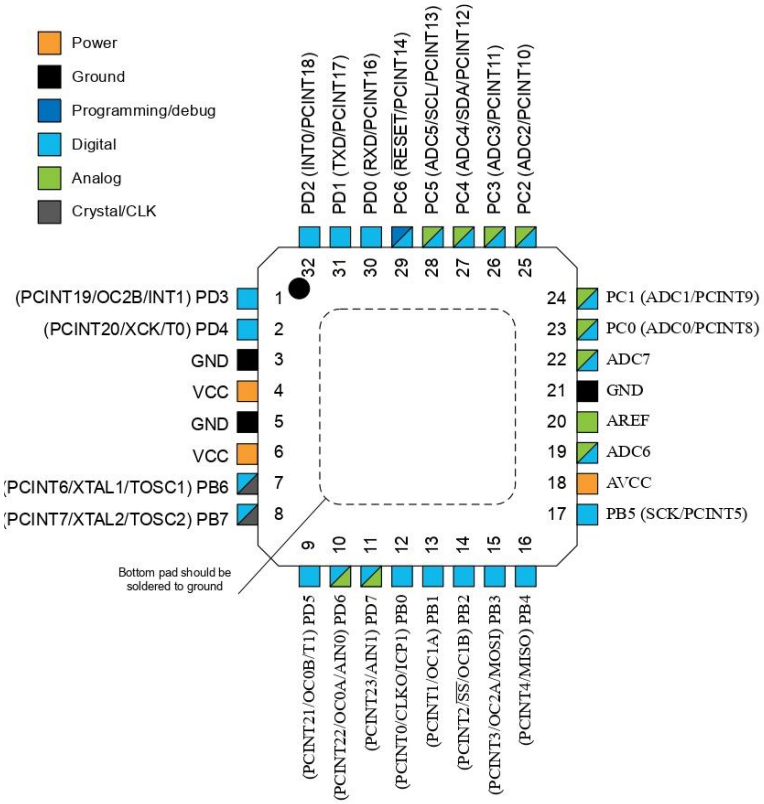


Figure 5-4. 32-pin MLF Top View



5.2. Pin Descriptions

5.2.1. VCC

Digital supply voltage.

5.2.2. GND

Ground.

5.2.3. Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

5.2.4. Port C (PC[5:0])

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

5.2.5. PC6/$\overline{\text{RESET}}$

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

The various special features of Port C are elaborated in the *Alternate Functions of Port C* section.

5.2.6. Port D (PD[7:0])

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

5.2.7. AV_{CC}

AV_{CC} is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter. Note that PC[6:4] use digital supply voltage, V_{CC}.

5.2.8. AREF

AREF is the analog reference pin for the A/D Converter.

5.2.9. ADC[7:6] (TQFP and VFQFN Package Only)

In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

6. I/O Multiplexing

Each pin is by default controlled by the PORT as a general purpose I/O and alternatively it can be assigned to one of the peripheral functions.

The following table describes the peripheral signals multiplexed to the PORT I/O pins.

Table 6-1. PORT Function Multiplexing

(32-pin MLF/TQFP) Pin#	(28-pin MLF) Pin#	(28-pin PIPD) Pin#	PAD	EXTINT	PCINT	ADC/AC	OSC	T/C #0	T/C #1	USART 0	I2C 0	SPI 0
1	1	5	PD[3]	INT1	PCINT19			OC2B				
2	2	6	PD[4]		PCINT20			T0		XCK0		
3	3	7	VCC									
4	4	8	GND									
5	-	-	VCC									
6	-	-	GND									
7	5	9	PB[6]		PCINT6		XTAL1/ TOSC1					
8	6	10	PB[7]		PCINT7		XTAL2/ TOSC2					
9	7	11	PD[5]		PCINT21			OC0B	T1			
10	8	12	PD[6]		PCINT22	AIN0		OC0A				
11	9	13	PD[7]		PCINT23	AIN1						
12	10	14	PB[0]		PCINT0		CLKO	ICP1				
13	11	15	PB[1]		PCINT1			OC1A				
14	12	16	PB[2]		PCINT2			OC1B				$\overline{SS}$0
15	13	17	PB[3]		PCINT3			OC2A				MOSI0
16	14	18	PB[4]		PCINT4							MISO0
17	15	19	PB[5]		PCINT5							SCK0
18	16	20	AVCC									
19	-	-	ADC6			ADC6						
20	17	21	AREF									
21	18	22	GND									
22	-	-	ADC7			ADC7						
23	19	13	PC[0]		PCINT8	ADC0						
24	20	24	PC[1]		PCINT9	ADC1						
25	21	25	PC[2]		PCINT10	ADC2						
26	22	26	PC[3]		PCINT11	ADC3						
27	23	27	PC[4]		PCINT12	ADC4					SDA0	
28	24	28	PC[5]		PCINT13	ADC5					SCL0	
29	25	1	PC[6]/ $\overline{RESET}$		PCINT14							

(32-pin MLF/TQFP) Pin#	(28-pin MLF) Pin#	(28-pin PIPD) Pin#	PAD	EXTINT	PCINT	ADC/AC	OSC	T/C #0	T/C #1	USART 0	I2C 0	SPI 0
30	26	2	PD[0]		PCINT16					RXD0		
31	27	3	PD[1]		PCINT17					TXD0		
32	28	4	PD[2]	INT0	PCINT18							

7. Resources

A comprehensive set of development tools, application notes, and datasheets are available for download on http://www.atmel.com/avr.

8. Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

9. About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Confirm with the C compiler documentation for more details.

For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

10. Capacitive Touch Sensing

10.1. QTouch Library

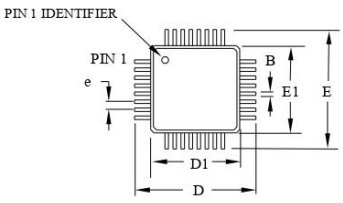
The Atmel® QTouch® Library provides a simple to use solution to realize touch sensitive interfaces on most Atmel AVR® microcontrollers. The QTouch Library includes support for the Atmel QTouch and Atmel QMatrix® acquisition methods.

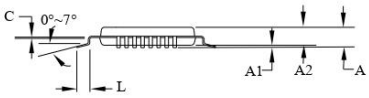
Touch sensing can be added to any application by linking the appropriate Atmel QTouch Library for the AVR Microcontroller. This is done by using a simple set of APIs to define the touch channels and sensors, and then calling the touch sensing API's to retrieve the channel information and determine the touch sensor states.

The QTouch Library is FREE and downloadable from the Atmel website at the following location: http://www.atmel.com/technologies/touch/. For implementation details and other information, refer to the Atmel QTouch Library User Guide - also available for download from the Atmel website.

11. Packaging Information

11.1. 32-pin 32A





COMMON DIMENSIONS
(Unit of measure = mm)

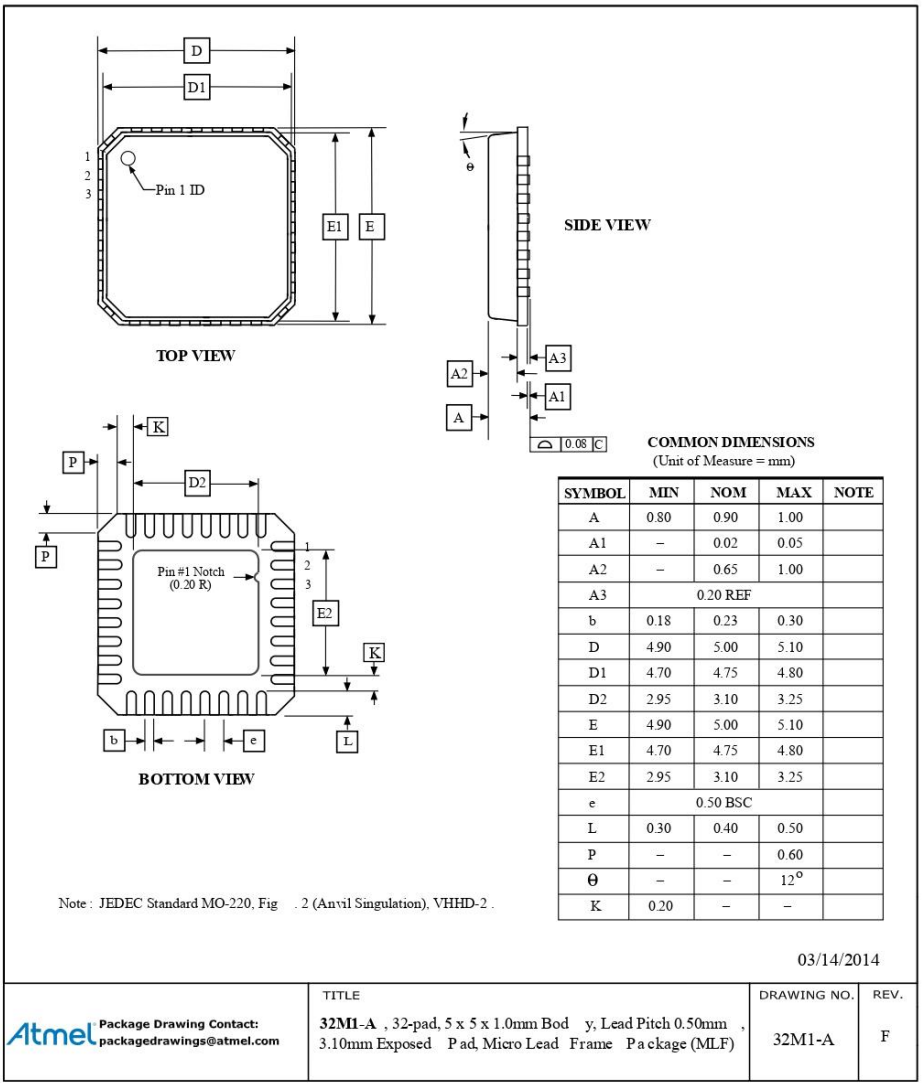
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	1.20	
A1	0.05	–	0.15	
A2	0.95	1.00	1.05	
D	8.75	9.00	9.25	
D1	6.90	7.00	7.10	Note 2
E	8.75	9.00	9.25	
E1	6.90	7.00	7.10	Note 2
B	0.30	–	0.45	
C	0.09	–	0.20	
L	0.45	–	0.75	
e	0.80 TYP			

Notes:
1. This package conforms to JEDEC reference MS-026, Variation ABA.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is 0.25mm per side. Dimensions D1 and E1 are maximum plastic body size dimensions including mold mismatch.
3. Lead coplanarity is 0.10mm maximum.

2010-10-20

	TITLE	DRAWING NO.	REV.
Atmel	**32A,** 32-lead, 7 x 7mm body size, 1.0mm body thickness, 0.8mm lead pitch, thin profile plastic quad flat package (TQFP)	32A	C

11.2. 32-pin 32M1-A



COMMON DIMENSIONS
(Unit of Measure = mm)

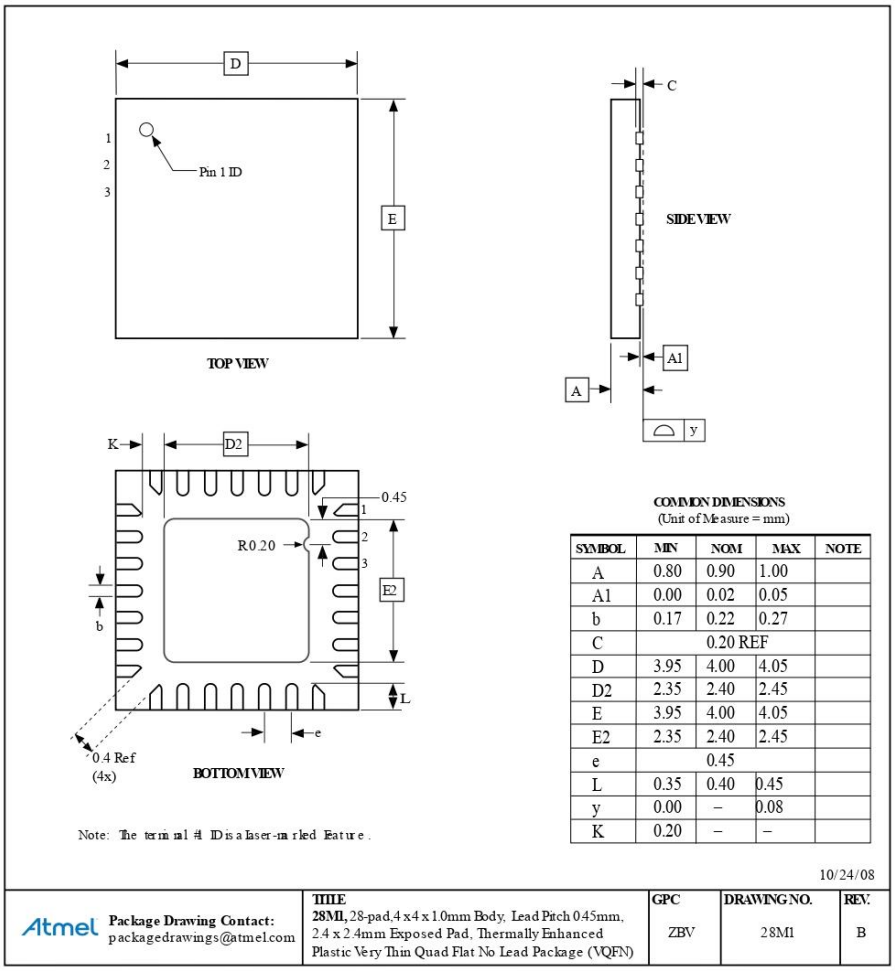
SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	–	0.02	0.05	
A2	–	0.65	1.00	
A3		0.20 REF		
b	0.18	0.23	0.30	
D	4.90	5.00	5.10	
D1	4.70	4.75	4.80	
D2	2.95	3.10	3.25	
E	4.90	5.00	5.10	
E1	4.70	4.75	4.80	
E2	2.95	3.10	3.25	
e		0.50 BSC		
L	0.30	0.40	0.50	
P	–	–	0.60	
θ	–	–	12°	
K	0.20	–	–	

Note : JEDEC Standard MO-220, Fig . 2 (Anvil Singulation), VHHD-2 .

03/14/2014

	TITLE	DRAWING NO.	REV.
Atmel Package Drawing Contact: packagedrawings@atmel.com	**32M1-A** , 32-pad, 5 x 5 x 1.0mm Bod y, Lead Pitch 0.50mm , 3.10mm Exposed Pad, Micro Lead Frame Package (MLF)	32M1-A	F

11.3. 28-pin 28M1



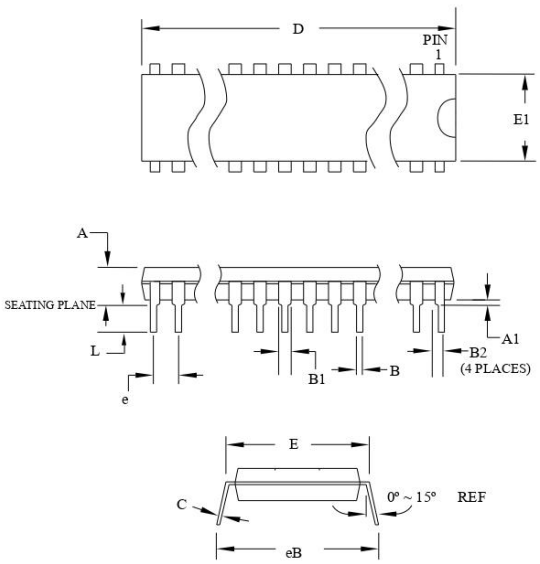
COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	0.80	0.90	1.00	
A1	0.00	0.02	0.05	
b	0.17	0.22	0.27	
C		0.20 REF		
D	3.95	4.00	4.05	
D2	2.35	2.40	2.45	
E	3.95	4.00	4.05	
E2	2.35	2.40	2.45	
e		0.45		
L	0.35	0.40	0.45	
y	0.00	–	0.08	
K	0.20	–	–	

10/24/08

Atmel Package Drawing Contact: packagedrawings@atmel.com	TITLE **28M1**, 28-pad, 4 x 4 x 1.0mm Body, Lead Pitch 0.45mm, 2.4 x 2.4mm Exposed Pad, Thermally Enhanced Plastic Very Thin Quad Flat No Lead Package (VQFN)	GPC ZBV	DRAWING NO. 28M1	REV. B

11.4. 28-pin 28P3



COMMON DIMENSIONS
(Unit of Measure = mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.5724	
A1	0.508	–	–	
D	34.544	–	34.798	Note 1
E	7.620	–	8.255	
E1	7.112	–	7.493	Note 1
B	0.381	–	0.533	
B1	1.143	–	1.397	
B2	0.762	–	1.143	
L	3.175	–	3.429	
C	0.203	–	0.356	
eB	–	–	10.160	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25mm (0.010").

09/28/01

	TITLE	DRAWING NO.	REV.
Atmel 2325 Orchard Parkway San Jose, CA 95131	**28P3**, 28-lead (0.300"/7.62mm Wide) Plastic Dual Inline Package (PDIP)	28P3	B

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

4. STM32F103C8T6

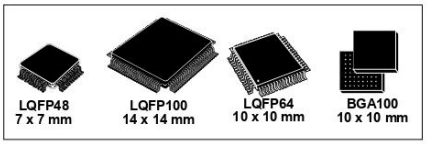
STM32F103x6
STM32F103x8 STM32F103xB

Performance line, ARM-based 32-bit MCU with Flash, USB, CAN, seven 16-bit timers, two ADCs and nine communication interfaces

Preliminary Data

Features

- Core: ARM 32-bit Cortex™-M3 CPU
  - 72 MHz, 90 DMIPS with 1.25 DMIPS/MHz
  - Single-cycle multiplication and hardware division
  - Nested interrupt controller with 43 maskable interrupt channels
  - Interrupt processing (down to 6 CPU cycles) with tail chaining
- Memories
  - 32-to-128 Kbytes of Flash memory
  - 6-to-20 Kbytes of SRAM
- Clock, reset and supply management
  - 2.0 to 3.6 V application supply and I/Os
  - POR, PDR, and programmable voltage detector (PVD)
  - 4-to-16 MHz quartz oscillator
  - Internal 8 MHz factory-trimmed RC
  - Internal 32 kHz RC
  - PLL for CPU clock
  - Dedicated 32 kHz oscillator for RTC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - V_{BAT} supply for RTC and backup registers
- 2 x 12-bit, 1 µs A/D converters (16-channel)
  - Conversion range: 0 to 3.6 V
  - Dual-sample and hold capability
  - Synchronizable with advanced control timer
  - Temperature sensor
- DMA
  - 7-channel DMA controller
  - Peripherals supported: timers, ADC, SPIs, I^2Cs and USARTs



- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
- Up to 80 fast I/O ports
  - 32/49/80 5 V-tolerant I/Os
  - All mappable on 16 external interrupt vectors
  - Atomic read/modify/write operations
- Up to 7 timers
  - Up to three 16-bit timers, each with up to 4 IC/OC/PWM or pulse counter
  - 16-bit, 6-channel advanced control timer:
    - up to 6 channels for PWM output
    - Dead time generation and emergency stop
  - 2 x 16-bit watchdog timers (Independent and Window)
  - SysTick timer: a 24-bit downcounter
- Up to 9 communication interfaces
  - Up to 2 x I^2C interfaces (SMBus/PMBus)
  - Up to 3 USARTs (ISO 7816 interface, LIN, IrDA capability, modem control)
  - Up to 2 SPIs (18 Mbit/s)
  - CAN interface (2.0B Active)
  - USB 2.0 full speed interface

Table 1. Device summary

Reference	Root part number
STM32F103x6	STM32F103C6, STM32F103R6
STM32F103x8	STM32F103C8, STM32F103R8 STM32F103V8
STM32F103xB	STM32F103RB STM32F103VB

Contents

List of tables

List of figures

1 Introduction

This datasheet provides the STM32F103xx performance line ordering information and mechanical device characteristics.

For information on programming, erasing and protection of the internal Flash memory please refer to the *STM32F10xxx Flash programming reference manual,* pm0042, available from *www.st.com*.

For information on the Cortex-M3 core please refer to the Cortex-M3 Technical Reference Manual.

2 Description

The STM32F103xx performance line family incorporates the high-performance ARM Cortex-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I^2Cs and SPIs, three USARTs, an USB and a CAN.

The STM32F103xx performance line family operates in the -40 to +105 °C temperature range, from a 2.0 to 3.6 V power supply. A comprehensive set of power-saving mode allows to design low-power applications.

The complete STM32F103xx performance line family includes devices in 4 different package types: from 48 pins to 100 pins. Depending on the device chosen, different sets of peripherals are included, the description below gives an overview of the complete range of peripherals proposed in this family.

These features make the STM32F103xx performance line microcontroller family suitable for a wide range of applications:

- Motor drive and application control
- Medical and handheld equipment
- PC peripherals gaming and GPS platforms
- Industrial applications: PLC, inverters, printers, and scanners
- Alarm systems, Video intercom, and HVAC

Figure 1 shows the general block diagram of the device family.

2.1 Device overview

Table 2. Device features and peripheral counts (STM32F103xx performance line)

Peripheral		STM32F103Cx		STM32F103Rx			STM32F103Vx	
Flash - Kbytes		32	64	32	64	128	64	128
SRAM - Kbytes		10	20	10	20		20	
Timers	General purpose	2	3	2	3		3	
	Advanced Control	1		1			1	
Communication	SPI	1	2	1	2		2	
	I^2C	1	2	1	2		2	
	USART	2	3	2	3		3	
	USB	1	1	1	1		1	
	CAN	1	1	1	1		1	
GPIOs		32		49			80	
12-bit synchronized ADC Number of channels		2 10 channels		2 16 channels				
CPU frequency		72 MHz						
Operating voltage		2.0 to 3.6 V						
Operating temperature		-40 to +85 °C / -40 to +105 °C						
Packages		LQFP48		LQFP64			LQFP100, BGA100	

2.2 Overview

ARM® Cortex™-M3 core with embedded Flash and SRAM

The ARM Cortex-M3 processor is the latest generation of ARM processors for embedded systems. It has been developed to provide a low-cost platform that meets the needs of MCU implementation, with a reduced pin count and low-power consumption, while delivering outstanding computational performance and an advanced system response to interrupts.

The ARM Cortex-M3 32-bit RISC processor features exceptional code-efficiency, delivering the high-performance expected from an ARM core in the memory size usually associated with 8- and 16-bit devices.

The STM32F103xx performance line family having an embedded ARM core, is therefore compatible with all ARM tools and software.

Figure 1 shows the general block diagram of the device family.

Embedded Flash memory

- Up to 128 Kbytes of embedded Flash is available for storing programs and data.

Embedded SRAM

Up to 20 Kbytes of embedded SRAM accessed (read/write) at CPU clock speed with 0 wait states.

Nested vectored interrupt controller (NVIC)

The STM32F103xx performance line embeds a Nested Vectored Interrupt Controller able to handle up to 43 maskable interrupt channels (not including the 16 interrupt lines of Cortex-M3) and 16 priority levels.

- Closely coupled NVIC gives low latency interrupt processing
- Interrupt entry vector table address passed directly to the core
- Closely coupled NVIC core interface
- Allows early processing of interrupts
- Processing of *late arriving* higher priority interrupts
- Support for tail-chaining
- Processor state automatically saved
- Interrupt entry restored on interrupt exit with no instruction overhead

This hardware block provides flexible interrupt management features with minimal interrupt latency.

External interrupt/event controller (EXTI)

The external interrupt/event controller consists of 19 edge detectors lines used to generate interrupt/event requests. Each line can be independently configured to select the trigger event (rising edge, falling edge, both) and can be masked independently. A pending register maintains the status of the interrupt requests. The EXTI can detect external line with pulse width lower than the Internal APB2 clock period. Up to 80 GPIOs are connected to the 16 external interrupt lines.

Clocks and startup

System clock selection is performed on startup, however the internal RC 8 MHz oscillator is selected as default CPU clock on reset. An external 4-16 MHz clock can be selected and is monitored for failure. During such a scenario, it is disabled and software interrupt management follows. Similarly, full interrupt management of the PLL clock entry is available when necessary (for example with failure of an indirectly used external oscillator).

Several prescalers allow the configuration of the AHB frequency, the High Speed APB (APB2) and the low Speed APB (APB1) domains. The maximum frequency of the AHB and the High Speed APB domains is 72 MHz. The maximum allowed frequency of the Low Speed APB domain is 36 MHz.

Boot modes

At startup, boot pins are used to select one of three boot options:

- Boot from User Flash
- Boot from System Memory
- Boot from SRAM

The boot loader is located in System Memory. It is used to reprogram the Flash memory by using the USART.

Power supply schemes

- V_{DD} = 2.0 to 3.6 V: external power supply for I/Os and the internal regulator. Provided externally through V_{DD} pins.
- V_{SSA}, V_{DDA} = 2.0 to 3.6 V: external analog power supplies for ADC, Reset blocks, RCs and PLL. In V_{DD} range (ADC is limited at 2.4 V).
- V_{BAT} = 1.8 to 3.6 V: power supply for RTC, external clock 32 kHz oscillator and backup registers (through power switch) when V_{DD} is not present.

Power supply supervisor

The device has an integrated Power On Reset (POR)/Power Down Reset (PDR) circuitry. It is always active, and ensures proper operation starting from/down to 2 V. The device remains in reset mode when V_{DD} is below a specified threshold, $V_{POR/PDR}$, without the need for an external reset circuit.

The device features an embedded programmable voltage detector (PVD) that monitors the V_{DD} power supply and compares it to the V_{PVD} threshold. An interrupt can be generated when V_{DD} drops below the V_{PVD} and/or when V_{DD} is higher than the V_{PVD} threshold. The interrupt service routine can then generate a warning message and/or put the MCU into a safe state. The PVD is enabled by software.

Refer to *Table 9: Embedded reset and power control block characteristics* for the values of $V_{POR/PDR}$ and V_{PVD}.

Voltage regulator

The regulator has three operation modes: main (MR), low power (LPR) and power down.

- MR is used in the nominal regulation mode (Run)
- LPR is used in the Stop modes.
- Power down is used in Standby Mode: the regulator output is in high impedance: the kernel circuitry is powered-down, inducing zero consumption (but the contents of the registers and SRAM are lost)

This regulator is always enabled after reset. It is disabled in Standby Mode, providing high impedance output.

Low-power modes

The STM32F103xx performance line supports three low-power modes to achieve the best compromise between low power consumption, short startup time and available wakeup sources:

- **Sleep** mode

  In Sleep mode, only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.

- **Stop** mode

  Stop mode allows to achieve the lowest power consumption while retaining the content of SRAM and registers. All clocks in the 1.8 V domain are stopped, the PLL, the HSI and the HSE RC oscillators are disabled. The voltage regulator can also be put either in normal or in low power mode.
  The device can be woken up from Stop mode by any of the EXTI line. The EXTI line source can be one of the 16 external lines, the PVD output, the RTC alarm or the USB wakeup.

- **Standby** mode

  The Standby mode allows to achieve the lowest power consumption. The internal voltage regulator is switched off so that the entire 1.8 V domain is powered off. The PLL, the HSI and the HSE RC oscillators are also switched off. After entering Standby mode, SRAM and registers content are lost except for registers in the Backup domain and Standby circuitry.

  The device exits Standby mode when an external reset (NRST pin), a IWDG reset, a rising edge on the WKUP pin, or an RTC alarm occurs.

Note: The RTC, the IWDG, and the corresponding clock sources are not stopped by entering Stop or Standby mode.

DMA

The flexible 7-channel general-purpose DMA is able to manage memory-to-memory, peripheral-to-memory and memory-to-peripheral transfers. The DMA controller supports circular buffer management avoiding the generation of interrupts when the controller reaches the end of the buffer.

Each channel is connected to dedicated hardware DMA requests, with support for software trigger on each channel. Configuration is made by software and transfer sizes between source and destination are independent.

The DMA can be used with the main peripherals: SPI, I^2C, USART, general purpose and advanced control timers TIMx and ADC.

RTC (real-time clock) and backup registers

The RTC and the backup registers are supplied through a switch that takes power either on V_{DD} supply when present or through the V_{BAT} pin. The backup registers (ten 16-bit registers) can be used to store data when V_{DD} power is not present.

The real-time clock provides a set of continuously running counters which can be used with suitable software to provide a clock calendar function, and provides an alarm interrupt and a periodic interrupt. It is clocked by an external 32.768 kHz oscillator, the internal low power RC oscillator or the High Speed External clock divided by 128. The internal low power RC has a typical frequency of 32 kHz. The RTC can be calibrated using an external 512 Hz output to compensate for any natural quartz deviation. The RTC features a 32-bit programmable counter for long term measurement using the Compare register to generate an alarm. A 20-bit prescaler is used for the time base clock and is by default configured to generate a time base of 1 second from a clock at 32.768 kHz.

Independent watchdog

The independent watchdog is based on a 12-bit downcounter and 8-bit prescaler. It is clocked from an independent 32 kHz internal RC and as it operates independently from the main clock, it can operate in Stop and Standby modes. It can be used either as a watchdog to reset the device when a problem occurs, or as a free running timer for application time out management. It is hardware or software configurable through the option bytes. The counter can be frozen in debug mode.

Window watchdog

The window watchdog is based on a 7-bit downcounter that can be set as free running. It can be used as a watchdog to reset the device when a problem occurs. It is clocked from the main clock. It has an early warning interrupt capability and the counter can be frozen in debug mode.

SysTick timer

This timer is dedicated for OS, but could also be used as a standard down counter. It features:

- A 24-bit down counter
- Autoreload capability
- Maskable system interrupt generation when the counter reaches 0.
- Programmable clock source

General purpose timers (TIMx)

There are up to 3 synchronizable standard timers embedded in the STM32F103xx performance line devices. These timers are based on a 16-bit auto-reload up/down counter, a 16-bit prescaler and feature 4 independent channels each for input capture/output compare, PWM or one pulse mode output. This gives up to 12 input captures / output compares / PWMs on the largest packages. They can work together with the Advanced Control Timer via the Timer Link feature for synchronization or event chaining.

The counter can be frozen in debug mode.

Any of the standard timers can be used to generate PWM outputs. Each of the timers has independent DMA request generations.

Advanced control timer (TIM1)

The advanced control timer (TIM1) can be seen as a three-phase PWM multiplexed on 6 channels. It can also be seen as a complete general-purpose timer. The 4 independent channels can be used for

- Input Capture
- Output Compare
- PWM generation (edge or center-aligned modes)
- One Pulse Mode output
- Complementary PWM outputs with programmable inserted dead-times.

If configured as a standard 16-bit timer, it has the same features as the TIMx timer. If configured as the 16-bit PWM generator, it has full modulation capability (0-100%).

The counter can be frozen in debug mode.

Many features are shared with those of the standard TIM timers which have the same architecture. The advanced control timer can therefore work together with the TIM timers via the Timer Link feature for synchronization or event chaining.

I²C bus

Up to two I²C bus interfaces can operate in multi-master and slave modes. They can support standard and fast modes.

They support dual slave addressing (7-bit only) and both 7/10-bit addressing in master mode. A hardware CRC generation/verification is embedded.

They can be served by DMA and they support SM Bus 2.0/PM Bus.

Universal synchronous/asynchronous receiver transmitter (USART)

One of the USART interfaces is able to communicate at speeds of up to 4.5 Mbit/s. The other available interfaces communicate at up to 2.25 Mbit/s. They provide hardware management of the CTS and RTS signals, IrDA SIR ENDEC support, are ISO 7816 compliant and have LIN Master/Slave capability.

All USART interfaces can be served by the DMA controller.

Serial peripheral interface (SPI)

Up to two SPIs are able to communicate up to 18 Mbits/s in slave and master modes in full-duplex and simplex communication modes. The 3-bit prescaler gives 8 master mode frequencies and the frame is configurable from 8-bit to 16-bit. The hardware CRC generation/verification supports basic SD Card/MMC modes.

Both SPIs can be served by the DMA controller.

Controller area network (CAN)

The CAN is compliant with specifications 2.0A and B (active) with a bit rate up to 1 Mbit/s. It can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers. It has three transmit mailboxes, two receive FIFOs with 3 stages and 14 scalable filter banks.

Universal serial bus (USB)

The STM32F103xx performance line embeds a USB device peripheral compatible with the USB Full-speed 12 Mbs. The USB interface implements a full speed (12 Mbit/s) function interface. It has software configurable endpoint setting and suspend/resume support. The dedicated 48 MHz clock source is generated from the internal main PLL.

GPIOs (general-purpose inputs/outputs)

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions. All GPIOs are high current-capable.

The I/Os alternate function configuration can be locked if needed following a specific sequence in order to avoid spurious writing to the I/Os registers.

I/Os on APB2 with up to 18 MHz toggling speed

ADC (analog to digital converter)

Two 12-bit Analog to Digital Converters are embedded into STM32F103xx performance line devices and each ADC shares up to 16 external channels, performing conversions in single-shot or scan modes. In scan mode, automatic conversion is performed on a selected group of analog inputs.

Additional logic functions embedded in the ADC interface allow:

- Simultaneous sample and hold
- Interleaved sample and hold
- Single shunt

The ADC can be served by the DMA controller.

An analog watchdog feature allows very precise monitoring of the converted voltage of one, some or all selected channels. An interrupt is generated when the converted voltage is outside the programmed thresholds.

The events generated by the standard timers (TIMx) and the Advanced Control timer (TIM1) can be internally connected to the ADC start trigger, injection trigger, and DMA trigger respectively, to allow the application to synchronize A/D conversion and timers.

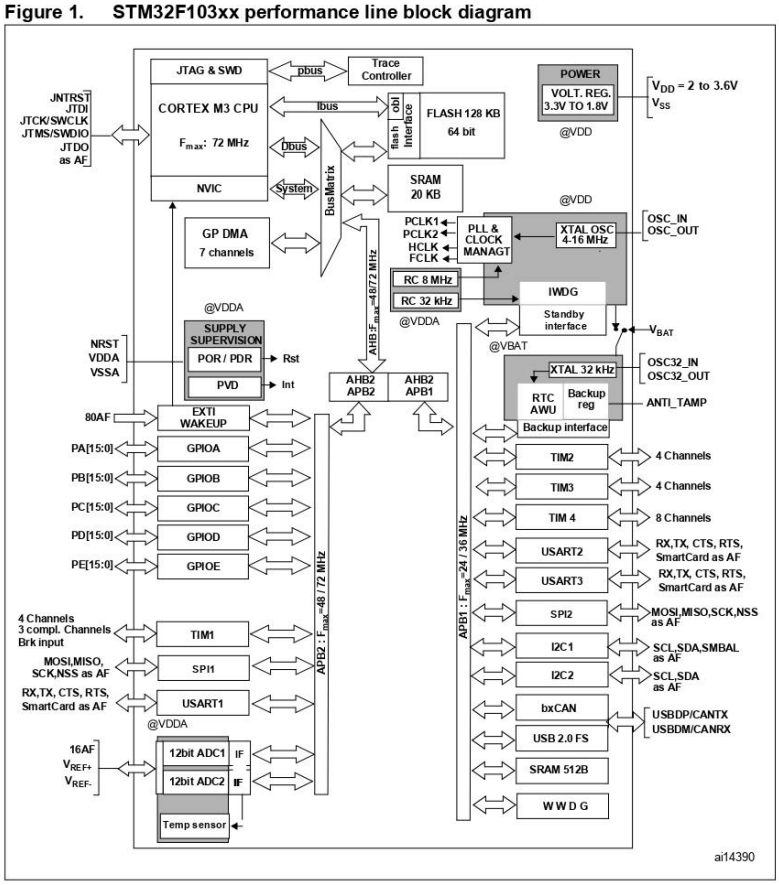
Temperature sensor

The temperature sensor has to generate a linear voltage with any variation in temperature. The conversion range is between 2 V < V_{DDA} < 3.6 V. The temperature sensor is internally connected to the ADC_IN16 input channel which is used to convert the sensor output voltage into a digital value.

Serial wire JTAG debug port (SWJ-DP)

The ARM SWJ-DP Interface is embedded. and is a combined JTAG and serial wire debug port that enables either a serial wire debug or a JTAG probe to be connected to the target. The JTAG TMS and TCK pins are shared respectively with SWDIO and SWCLK and a specific sequence on the TMS pin is used to switch between JTAG-DP and SW-DP.

Figure 1. STM32F103xx performance line block diagram



1. T_A = –40 °C to +105 °C (junction temperature up to 125 °C).
2. AF = alternate function on I/O port pin.

3 Pin descriptions

Figure 2. STM32F103xx performance line LQFP100 pinout

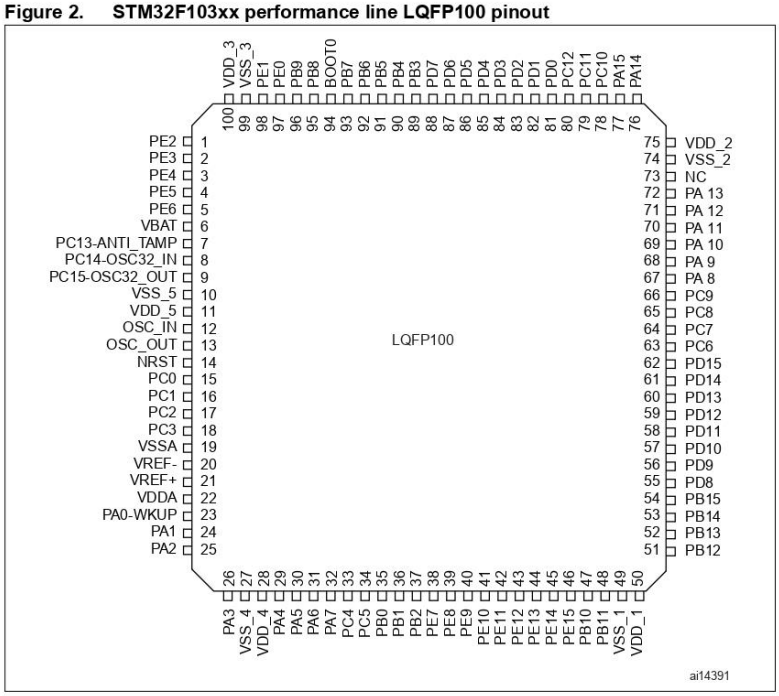


Figure 3. STM32F103xx performance line LQFP64 pinout

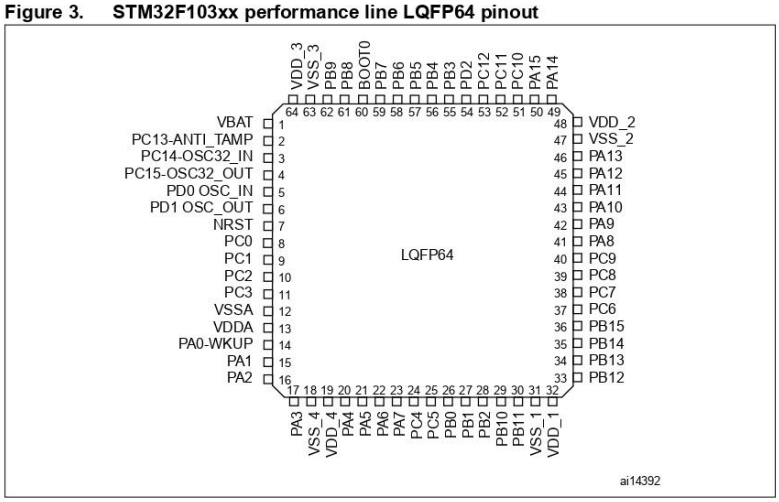


Figure 4. STM32F103xx performance line LQFP48 pinout

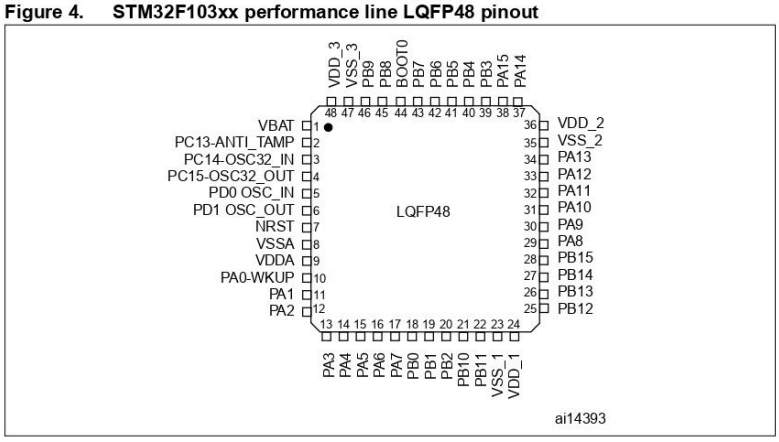


Figure 5. STM32F103xx performance line BGA100 ballout

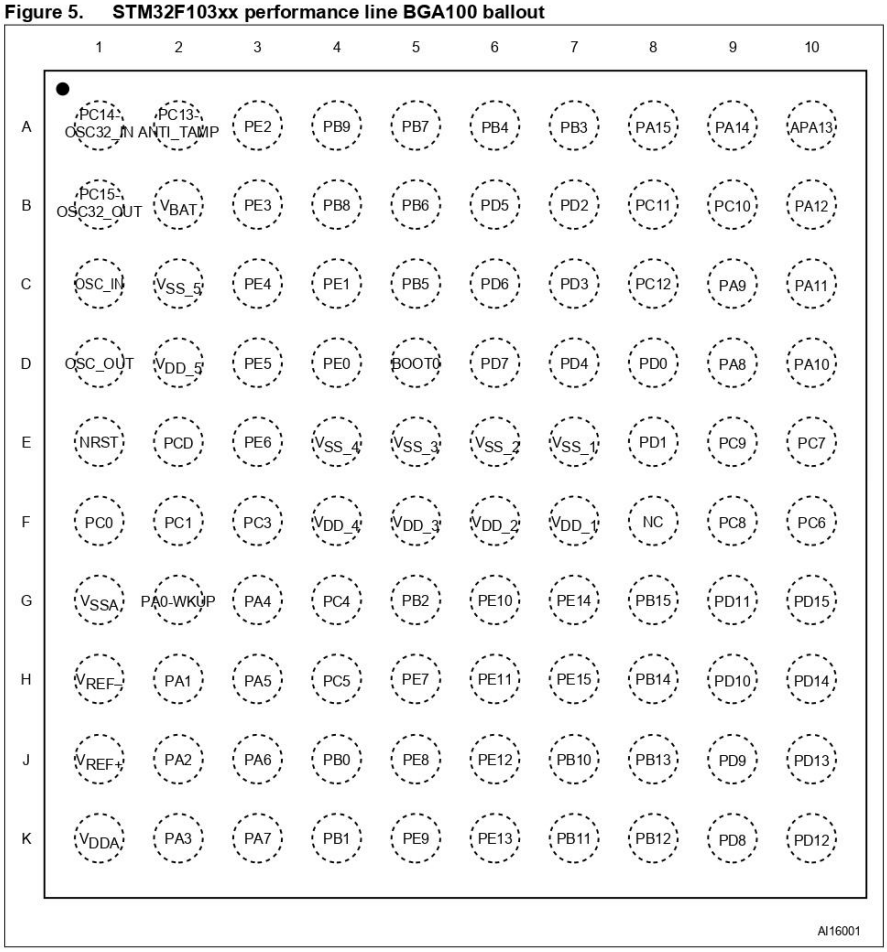


Table 3. Pin definitions

Pins				Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Default alternate functions
BGA100	LQFP48	LQFP64	LQFP100					
A3	-	-	1	PE2/TRACECK	I/O	FT	PE2	TRACECK
B3	-	-	2	PE3/TRACED0	I/O	FT	PE3	TRACED0
C3	-	-	3	PE4/TRACED1	I/O	FT	PE4	TRACED1
D3	-	-	4	PE5/TRACED2	I/O	FT	PE5	TRACED2
E3	-	-	5	PE6/TRACED3	I/O	FT	PE6	TRACED3
B2	1	1	6	V_{BAT}	S		V_{BAT}	
A2	2	2	7	PC13-ANTI_TAMP(4)	I/O		PC13	ANTI_TAMP
A1	3	3	8	PC14-OSC32_IN(4)	I/O		PC14-OSC32_IN	
B1	4	4	9	PC15-OSC32_OUT(4)	I/O		PC15-OSC32_OUT	
C2	-	-	10	V_{SS_5}	S		V_{SS_5}	
D2	-	-	11	V_{DD_5}	S		V_{DD_5}	
C1	5	5	12	OSC_IN	I		OSC_IN	
D1	6	6	13	OSC_OUT	O		OSC_OUT	
E1	7	7	14	NRST	I/O		NRST	
F1	-	8	15	PC0/ADC_IN10	I/O		PC0	ADC_IN10
F2	-	9	16	PC1/ADC_IN11	I/O		PC1	ADC_IN11
E2	-	10	17	PC2/ADC_IN12	I/O		PC2	ADC_IN12
F3	-	11	18	PC3/ADC_IN13	I/O		PC3	ADC_IN13
G1	8	12	19	V_{SSA}	S		V_{SSA}	
H1	-	-	20	V_{REF-}	S		V_{REF-}	
J1	-	-	21	V_{REF+}	S		V_{REF+}	
K1	9	13	22	V_{DDA}	S		V_{DDA}	
G2	10	14	23	PA0-WKUP/ USART2_CTS/ ADC_IN0/TIM2_CH1_ETR	I/O		PA0	WKUP/USART2_CTS(6)/AD C_IN0/ TIM2_CH1_ETR(6)
H2	11	15	24	PA1/USART2_RTS/ ADC_IN1/TIM2_CH2	I/O		PA1	USART2_RTS(6)/ ADC_IN1/ TIM2_CH2(6)
J2	12	16	25	PA2/USART2_TX/ ADC_IN2/ TIM2_CH3	I/O		PA2	USART2_TX(6)/ ADC_IN2/ TIM2_CH3(6)
K2	13	17	26	PA3/USART2_RX/ ADC_IN3/TIM2_CH4	I/O		PA3	USART2_RX(6)/ ADC_IN3/TIM2_CH4(6)
E4	-	18	27	V_{SS_4}	S		V_{SS_4}	
F4	-	19	28	V_{DD_4}	S		V_{DD_4}	

Table 3. Pin definitions (continued)

Pins				Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Default alternate functions
BGA100	LQFP48	LQFP64	LQFP100					
G3	14	20	29	PA4/SPI1_NSS/ USART2_CK/ADC_IN4	I/O		PA4	SPI1_NSS(6)/ USART2_CK(6)/ ADC_IN4
H3	15	21	30	PA5/SPI1_SCK/ ADC_IN5	I/O		PA5	SPI1_SCK(6)/ ADC_IN5
J3	16	22	31	PA6/SPI1_MISO/ ADC_IN6/TIM3_CH1	I/O		PA6	SPI1_MISO(6)/ ADC_IN6/TIM3_CH1(6)
K3	17	23	32	PA7/SPI1_MOSI/ ADC_IN7/TIM3_CH2	I/O		PA7	SPI1_MOSI(6)/ ADC_IN7/TIM3_CH2(6)
G4	-	24	33	PC4/ADC_IN14	I/O		PC4	ADC_IN14
H4	-	25	34	PC5/ADC_IN15	I/O		PC5	ADC_IN15
J4	18	26	35	PB0/ADC_IN8/ TIM3_CH3	I/O		PB0	ADC_IN8/TIM3_CH3(6)
K4	19	27	36	PB1/ADC_IN9/ TIM3_CH4	I/O		PB1	ADC_IN9/TIM3_CH4(6)
G5	20	28	37	PB2 / BOOT1	I/O	FT	PB2/BOOT1	
H5	-	-	38	PE7	I/O	FT	PE7	
J5	-	-	39	PE8	I/O	FT	PE8	
K5	-	-	40	PE9	I/O	FT	PE9	
G6	-	-	41	PE10	I/O	FT	PE10	
H6	-	-	42	PE11	I/O	FT	PE11	
J6	-	-	43	PE12	I/O	FT	PE12	
K6	-	-	44	PE13	I/O	FT	PE13	
G7	-	-	45	PE14	I/O	FT	PE14	
H7	-	-	46	PE15	I/O	FT	PE15	
J7	21	29	47	PB10/I2C2_SCL/ USART3_TX	I/O	FT	PB10	I2C2_SCL/USART3_TX(5)(6)
K7	22	30	48	PB11/I2C2_SDA / USART3_RX	I/O	FT	PB11	I2C2_SDA/ USART3_RX(5)(6)
E7	23	31	49	V_{SS_1}	S		V_{SS_1}	
F7	24	32	50	V_{DD_1}	S		V_{DD_1}	
K8	25	33	51	PB12/SPI2_NSS / I2C2_SMBAI/ USART3_CK / TIM1_BKIN	I/O	FT	PB12	SPI2_NSS(5) /I2C2_SMBAI(5)/ USART3_CK(5)(6)/ TIM1_BKIN(6)
J8	26	34	52	PB13/SPI2_SCK / USART3_CTS / TIM1_CH1N	I/O	FT	PB13	SPI2_SCK(5)/ USART3_CTS(5)(6)/ TIM1_CH1N (6)
H8	27	35	53	PB14/SPI2_MISO / USART3_RTS / TIM1_CH2N	I/O	FT	PB14	SPI2_MISO(5) /USART3_RTS(5)(6) TIM1_CH2N (6)

Table 3. Pin definitions (continued)

Pins				Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Default alternate functions
BGA100	LQFP48	LQFP64	LQFP100					
G8	28	36	54	PB15/SPI2_MOSI TIM1_CH3N	I/O	FT	PB15	SPI2_MOSI(5)/ TIM1_CH3N(6)
K9	-	-	55	PD8	I/O	FT	PD8	
J9	-	-	56	PD9	I/O	FT	PD9	
H9	-	-	57	PD10	I/O	FT	PD10	
G9	-	-	58	PD11	I/O	FT	PD11	
K10	-	-	59	PD12	I/O	FT	PD12	
J10	-	-	60	PD13	I/O	FT	PD13	
H10	-	-	61	PD14	I/O	FT	PD14	
G10	-	-	62	PD15	I/O	FT	PD15	
F10	-	37	63	PC6	I/O	FT	PC6	
E10		38	64	PC7	I/O	FT	PC7	
F9		39	65	PC8	I/O	FT	PC8	
E9	-	40	66	PC9	I/O	FT	PC9	
D9	29	41	67	PA8/USART1_CK/ TIM1_CH1/MCO	I/O	FT	PA8	USART1_CK/ TIM1_CH1(6)/MCO
C9	30	42	68	PA9/USART1_TX/ TIM1_CH2	I/O	FT	PA9	USART1_TX(6)/ TIM1_CH2(6)
D10	31	43	69	PA10/USART1_RX/ TIM1_CH3	I/O	FT	PA10	USART1_RX(6)/ TIM1_CH3(6)
C10	32	44	70	PA11 / USART1_CTS/ CANRX / USBDM/ TIM1_CH4	I/O	FT	PA11	USART1_CTS/ CANRX(6)/ TIM1_CH4(6) / USBDM
B10	33	45	71	PA12 / USART1_RTS/ CANTX / USBDP/ TIM1_ETR	I/O	FT	PA12	USART1_RTS/ CANTX(6) / TIM1_ETR(6) / USBDP
A10	34	46	72	PA13/JTMS/SWDIO	I/O	FT	JTMS/SWDIO	PA13
F8	-	-	73	Not connected				
E6	35	47	74	V_{SS_2}	S		V_{SS_2}	
F6	36	48	75	V_{DD_2}	S		V_{DD_2}	
A9	37	49	76	PA14/JTCK/SWCLK	I/O	FT	JTCK/SWCLK	PA14
A8	38	50	77	PA15/JTDI	I/O	FT	JTDI	PA15
B9	-	51	78	PC10	I/O	FT	PC10	
B8	-	52	79	PC11	I/O	FT	PC11	
C8	-	53	80	PC12	I/O	FT	PC12	

Table 3. Pin definitions (continued)

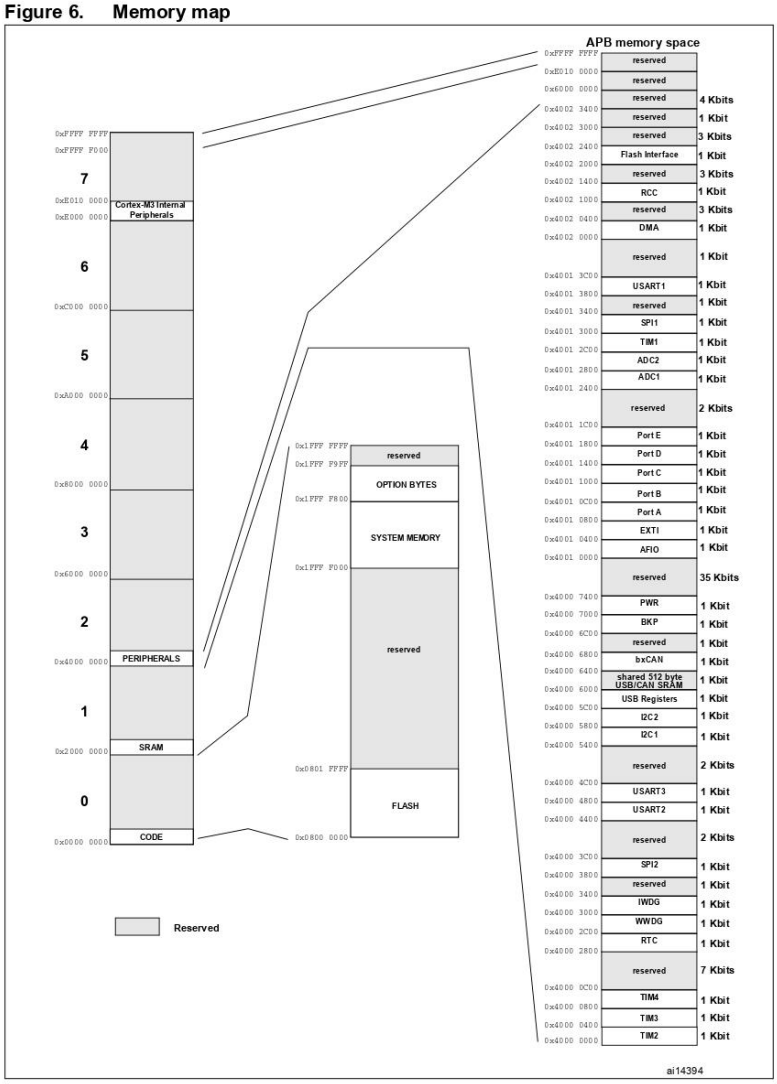
Pins				Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Default alternate functions
BGA100	LQFP48	LQFP64	LQFP100					
D8	5	5	81	PD0	I/O	FT	OSC_IN(7)	
E8	6	6	82	PD1	I/O	FT	OSC_OUT(7)	
B7		54	83	PD2/TIM3_ETR	I/O	FT	PD2	TIM3_ETR
C7	-	-	84	PD3	I/O	FT	PD3	
D7	-	-	85	PD4	I/O	FT	PD4	
B6	-	-	86	PD5	I/O	FT	PD5	
C6	-	-	87	PD6	I/O	FT	PD6	
D6	-	-	88	PD7	I/O	FT	PD7	
A7	39	55	89	PB3/JTDO/TRACESWO	I/O	FT	JTDO	PB3/TRACESWO
A6	40	56	90	PB4/JNTRST	I/O	FT	JNTRST	PB4
C5	41	57	91	PB5/I2C1_SMBAI	I/O		PB5	I2C1_SMBAI
B5	42	58	92	PB6/I2C1_SCL/ TIM4_CH1	I/O	FT	PB6	I2C1_SCL(6)/ TIM4_CH1(5)(6)
A5	43	59	93	PB7/I2C1_SDA/ TIM4_CH2	I/O	FT	PB7	I2C1_SDA(6)/ TIM4_CH2(5) (6)
D5	44	60	94	BOOT0	I		BOOT0	
B4	45	61	95	PB8/TIM4_CH3	I/O	FT	PB8	TIM4_CH3(5) (6)
A4	46	62	96	PB9/TIM4_CH4	I/O	FT	PB9	TIM4_CH4(5) (6)
D4	-	-	97	PE0/TIM4_ETR	I/O	FT	PE0	TIM4_ETR(5)
C4	-	-	98	PE1	I/O	FT	PE1	
E5	47	63	99	V_{SS_3}	S		V_{SS_3}	
F5	48	64	100	V_{DD_3}	S		V_{DD_3}	

1. I = input, O = output, S = supply, HiZ = high impedance.
2. FT= 5 V tolerant.
3. Function availability depends on the chosen device. Refer to *Table 2 on page 7*.
4. PC13, PC14 and PC15 are supplied through the power switch, and so their use in ouptut mode is limited: they can be used only in output 2 MHz mode with a maximum load of 30 pF and only one pin can be put in output mode at a time.
5. Available only on devices with a Flash memory density equal or higher than 64 Kbytes.
6. This alternate function can be remapped by software to some other port pins (if available on the used package). For more details, refer to the Alternate function I/O and debug configuration section in the STM32F10xxx reference manual, UM0306, available from the STMicroelectronics website: www.st.com.
7. For the LQFP48 and LQFP64 packages, the pins number 5 and 6 are configured as OSC_IN/OSC_OUT after reset, however the functionality of PD0 and PD1 can be remapped by software on these pins.

4 Memory mapping

The memory map is shown in *Figure 6*.

Figure 6. Memory map



5 Electrical characteristics

5.1 Test conditions

Unless otherwise specified, all voltages are referred to V_{SS}.

5.1.1 Minimum and maximum values

Unless otherwise specified the minimum and maximum values are guaranteed in the worst conditions of ambient temperature, supply voltage and frequencies by tests in production on 100% of the devices with an ambient temperature at T_A=25°C and T_A=$T_A$max (given by the selected temperature range).

Data based on characterization results, design simulation and/or technology characteristics are indicated in the table footnotes and are not tested in production. Based on characterization, the minimum and maximum values refer to sample tests and represent the mean value plus or minus three times the standard deviation (mean±3Σ).

5.1.2 Typical values

Unless otherwise specified, typical data are based on T_A = 25 °C, V_{DD} = 3.3 V (for the 2 V ≤V_{DD} ≤3.6 V voltage range). They are given only as design guidelines and are not tested.

Typical ADC accuracy values are determined by characterization of a batch of samples from a standard diffusion lot over the full temperature range, where 95% of the devices have an error less than or equal to the value indicated (mean±2Σ).

5.1.3 Typical curves

Unless otherwise specified, all typical curves are given only as design guidelines and are not tested.

5.1.4 Loading capacitor

The loading conditions used for pin parameter measurement are shown in *Figure 7*.

5.1.5 Pin input voltage

The input voltage measurement on a pin of the device is described in *Figure 8*.

Figure 7. Pin loading conditions

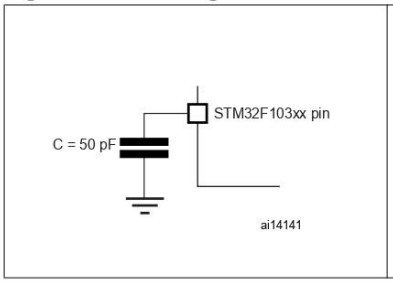
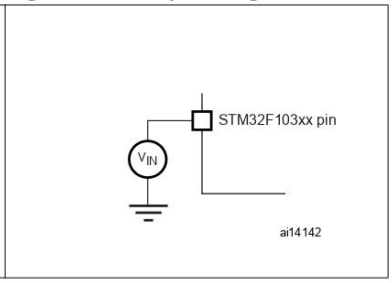
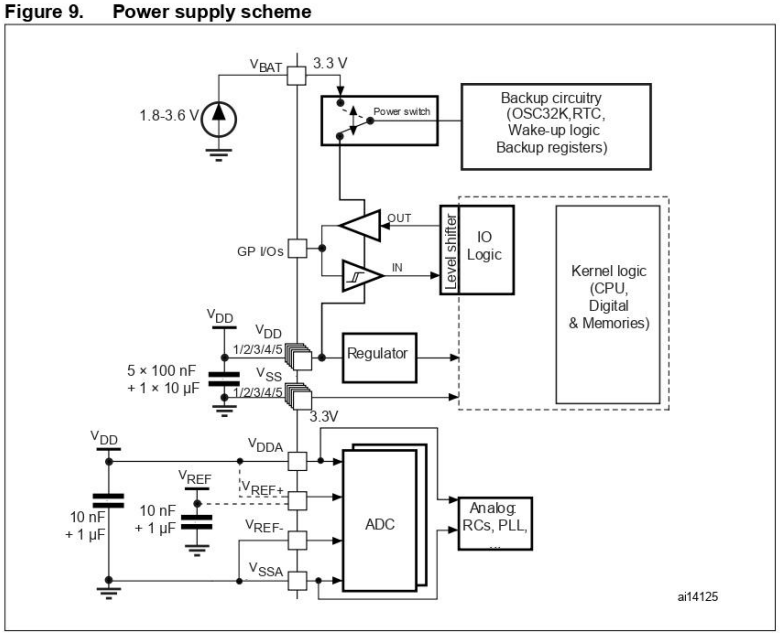


Figure 8. Pin input voltage



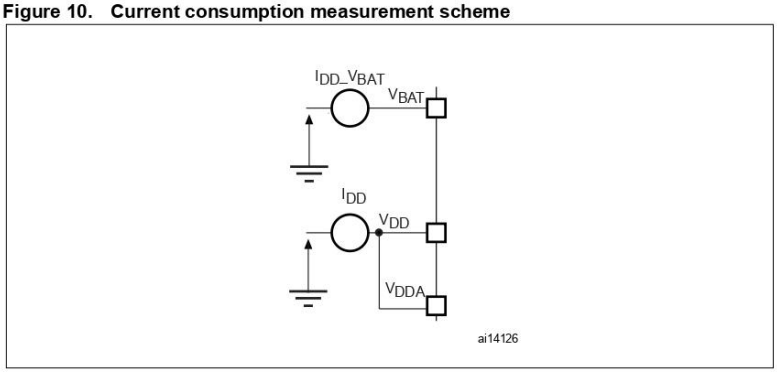
5.1.6 Power supply scheme

Figure 9. Power supply scheme



5.1.7 Current consumption measurement

Figure 10. Current consumption measurement scheme



5.2 Absolute maximum ratings

Stresses above the absolute maximum ratings listed in *Table 4: Voltage characteristics*, *Table 5: Current characteristics*, and *Table 6: Thermal characteristics* may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 4. Voltage characteristics

Symbol	Ratings	Min	Max	Unit
V_{DD}–V_{SS}	External 3.3 V supply voltage (including V_{DDA} and V_{DD})(1)	–0.3	4.0	V
V_{IN}	Input voltage on five volt tolerant pin(2)	V_{SS} –0.3	+5.5	
	Input voltage on any other pin(2)	V_{SS} –0.3	V_{DD}+0.3	
$\lvert\Delta V_{DDx}\rvert$	Variations between different power pins	50	50	mV
$\lvert V_{SSX} - V_{SS}\rvert$	Variations between all the different ground pins	50	50	
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	see *Section 5.3.11: Absolute maximum ratings (electrical sensitivity)*		

1. All 3.3 V power (V_{DD}, V_{DDA}) and ground (V_{SS}, V_{SSA}) pins must always be connected to the external 3.3 V supply.
2. $I_{INJ(PIN)}$ must never be exceeded (see *Table 5: Current characteristics*). This is implicitly insured if V_{IN} maximum is respected. If V_{IN} maximum cannot be respected, the injection current must be limited externally to the $I_{INJ(PIN)}$ value. A positive injection is induced by V_{IN}>V_{DD} while a negative injection is induced by V_{IN} < V_{SS}.

Table 5. Current characteristics

Symbol	Ratings	Max.	Unit
I_{VDD}	Total current into V_{DD} power lines (source)(1)	150	mA
I_{VSS}	Total current out of V_{SS} ground lines (sink)(1)	150	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current source by any I/Os and control pin	–25	
$I_{INJ(PIN)}$ (2)(3)	Injected current on NRST pin	± 5	
	Injected current on HSE OSC_IN and LSE OSC_IN pins	± 5	
	Injected current on any other pin(4)	± 5	
$\Sigma I_{INJ(PIN)}$(2)	Total injected current (sum of all I/O and control pins)(4)	± 25	

1. All 3.3 V power (V_{DD}, V_{DDA}) and ground (V_{SS}, V_{SSA}) pins must always be connected to the external 3.3 V supply.
2. $I_{INJ(PIN)}$ must never be exceeded. This is implicitly insured if V_{IN} maximum is respected. If V_{IN} maximum cannot be respected, the injection current must be limited externally to the $I_{INJ(PIN)}$ value. A positive injection is induced by V_{IN} > V_{DD} while a negative injection is induced by V_{IN} < V_{SS}.
3. Negative injection disturbs the analog performance of the device. See note in *Section 5.3.17: 12-bit ADC characteristics*.
4. When several inputs are submitted to a current injection, the maximum $\Sigma I_{INJ(PIN)}$ is the absolute sum of the positive and negative injected currents (instantaneous values). These results are based on characterization with $\Sigma I_{INJ(PIN)}$ maximum current injection on four I/O port pins of the device.

Table 6. Thermal characteristics

Symbol	Ratings	Value	Unit
T_{STG}	Storage temperature range	–65 to +150	°C
T_J	Maximum junction temperature (see Thermal characteristics)		

5.3 Operating conditions

5.3.1 General operating conditions

Table 7. General operating conditions

Symbol	Parameter	Conditions	Min	Max	Unit
f_{HCLK}	Internal AHB clock frequency		0	72	MHz
f_{PCLK1}	Internal APB1 clock frequency		0	36	
f_{PCLK2}	Internal APB2 clock frequency		0	72	
V_{DD}	Standard operating voltage		2	3.6	V
V_{BAT}	Backup operating voltage		1.8	3.6	V
T_A	Ambient temperature range		-40	105	°C

5.3.2 Operating conditions at power-up / power-down

The parameters given in *Table 8* are derived from tests performed under the ambient temperature condition summarized in *Table 7*.

Table 8. Operating conditions at power-up / power-down

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_{VDD}	V_{DD} rise/fall time rate		20			µs/V
					20	ms/V

5.3.3 Embedded reset and power control block characteristics

The parameters given in *Table 9* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 9. Embedded reset and power control block characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{PVD}	Programmable voltage detector level selection	PLS[2:0]=000 (rising edge)	2.1	2.18	2.26	V
		PLS[2:0]=000 (falling edge)	2	2.08	2.16	V
		PLS[2:0]=001 (rising edge)	2.19	2.28	2.37	V
		PLS[2:0]=001 (falling edge)	2.09	2.18	2.27	V
		PLS[2:0]=010 (rising edge)	2.28	2.38	2.48	V
		PLS[2:0]=010 (falling edge)	2.18	2.28	2.38	V
		PLS[2:0]=011 (rising edge)	2.38	2.48	2.58	V
		PLS[2:0]=011 (falling edge)	2.28	2.38	2.48	V
		PLS[2:0]=100 (rising edge)	2.47	2.58	2.69	V
		PLS[2:0]=100 (falling edge)	2.37	2.48	2.59	V
		PLS[2:0]=101 (rising edge)	2.57	2.68	2.79	V
		PLS[2:0]=101 (falling edge)	2.47	2.58	2.69	V
		PLS[2:0]=110 (rising edge)	2.66	2.78	2.9	V
		PLS[2:0]=110 (falling edge)	2.56	2.68	2.8	V
		PLS[2:0]=111 (rising edge)	2.76	2.88	3	V
		PLS[2:0]=111 (falling edge)	2.66	2.78	2.9	V
$V_{PVDhyst}$	PVD hysteresis			100		mV
$V_{POR/PDR}$	Power on/power down reset threshold	Falling edge	1.8	1.88	1.96	V
		Rising edge	1.84	1.92	2.0	V
$V_{PDRhyst}$	PDR hysteresis			40		mV
$T_{RSTTEMPO}$	Reset temporization		1	2.5	4.5	mS

5.3.4 Embedded reference voltage

The parameters given in *Table 10* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 10. Embedded internal reference voltage

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{REFINT}	Internal reference voltage	-45°C < T_A < +105°C	1.16	1.20	1.26	V
		-45°C < T_A < +85°C	1.16	1.20	1.24	V

5.3.5 Supply current characteristics

The current consumption is measured as described in *Figure 10: Current consumption measurement scheme*.

Maximum current consumption

The MCU is placed under the following conditions:

- All I/O pins are in input mode with a static value at V_{DD} or V_{SS} (no load)
- All peripherals are disabled except if it is explicitly mentioned
- The Flash access time is adjusted to f_{HCLK} frequency (0 wait state from 0 to 24 MHz, 1 wait state from 24 to 48 MHz and 2 wait states above)

The parameters given in *Table 11* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 11. Maximum current consumption in Run and Sleep modes(1)

Symbol	Parameter	Conditions	F_{HCLK}	Typ(2)	Max(3)		Unit
					T_A = 85 °C	T_A= 105 °C	
I_{DD}	Supply current in Run mode	External clock with PLL, code running from Flash, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	72 MHz	36	TBD	TBD	mA
			48 MHz	30	TBD	TBD	
			36 MHz	22	TBD	TBD	
			24 MHz	21	TBD	TBD	
		External clock, PLL stopped, code running from Flash, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	8 MHz	10	TBD	TBD	
		External clock with PLL, code running from RAM, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	72 MHz	32	45	47	
			48 MHz	22	31	33	
			36 MHz	13	18	20	
			24 MHz	11	15	17	
		External clock, PLL stopped, code running from RAM, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	8 MHz	4.5	TBD	TBD	
	Supply current in Sleep mode	External clock with PLL, code running from RAM or Flash, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	72 MHz	22	35	37	mA
			48 MHz	14	23	25	
			36 MHz	13	22	24	
			24 MHz	10	17	19	
		External clock, PLL stopped, code running from RAM or Flash, all peripherals enabled (see RCC register description): $f_{PCLK1}= f_{HCLK}/2$, $f_{PCLK2} = f_{HCLK}$	8 MHz	3.5	TBD	TBD	

1. TBD stands for to be determined.
2. Typical values are measured at T_A = 25 °C, and V_{DD} = 3.3 V
3. Data based on characterization results, tested in production at V_{Dmax}, f_{HCLK} max. T_{Amax}, and code executed from RAM.

Table 12. Maximum current consumption in Stop and Standby modes(1)

Symbol	Parameter	Conditions	Typ(2)		Max(3)		Unit
			V_{DD}/V_{BAT} = 2.4 V	V_{DD}/V_{BAT} = 3.3 V	T_A = 85 °C	T_A = 105 °C	
I_{DD}	Supply current in Stop mode	Regulator in Run mode, Low-speed and high-speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	TBD	24	TBD	TBD	µA
		Regulator in Low Power mode, Low-speed and high-speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	TBD(4)	14(4)	TBD(4)	TBD(4)	
	Supply current in Standby mode(5)	Low-speed internal RC oscillator and independent watchdog OFF, low-speed oscillator and RTC OFF	TBD(4)	2(4)	TBD(4)	TBD(4)	
I_{DD_VBAT}	Backup domain supply current	Low-speed oscillator and RTC ON	1(4)	1.4(4)	TBD(4)	TBD(4)	

1. TBD stands for to be determined.
2. Typical values are measured at T_A = 25 °C, V_{DD} = 3.3 V, unless otherwise specified.
3. Data based on characterization results, tested in production at $V_{DD\ max}$, f_{HCLK} max. and T_A max (for other temperature.
4. Values expected for next silicon revision.
5. To have the Standby consumption with RTC ON, add I_{DD_VBAT} (Low-speed oscillator and RTC ON) to I_{DD} Standby (when V_{DD} is present the Backup Domain is powered by V_{DD} supply).

Typical current consumption

The MCU is placed under the following conditions:

- All I/O pins are in input mode with a static value at V_{DD} or V_{SS} (no load).
- All peripherals are disabled except if it is explicitly mentioned.
- The Flash access time is adjusted to f_{HCLK} frequency (0 wait state from 0 to 24 MHz, 1 wait state from 24 to 48 MHZ and 2 wait states above).
- Ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 13. Typical current consumption in Run and Sleep modes(1)

Symbol	Parameter	Conditions	f_{HCLK}	Typ(2)	Unit
I_{DD}	Supply current in Run mode	Oscillator running at 8 MHz with PLL, code running from Flash, all peripheral disabled (see RCC register description): f_{PCLK1}= f_{HCLK}/2, f_{PCLK2}=f_{HCLK}	72 MHz	21	mA
			48 MHz	18	
			36 MHz	TBD	
			24 MHz	13	
			16 MHz	TBD	
		Running on HSI clock, code running from Flash, all peripheral disabled (see RCC register description): f_{PCLK1}= f_{HCLK}/2, f_{PCLK2}=f_{HCLK}. AHB pre-scaler used to reduce the frequency	8 MHz	7.8	mA
			4 MHz	7	
			2 MHz	6.3	
			1 MHz	6.2	
			500 kHz	6.1	
			125 kHz	5.95	
		Running on HSI clock, code running from RAM, all peripheral disabled (see RCC register description): f_{PCLK1}= f_{HCLK}/2, f_{PCLK2}=f_{HCLK}. AHB pre-scaler used to reduce the frequency	8 MHz	2.3	mA
			4 MHz	1.6	
			2 MHz	1.2	
			1 MHz	1	
			500 kHz	0.88	
			125 kHz	0.82	
	Supply current in Sleep mode	Oscillator running at 8MHz with PLL, code running from Flash, all peripheral disabled (see RCC register description): f_{PCLK1}= f_{HCLK}/2, f_{PCLK2}=f_{HCLK}	72 MHz	6	mA
			48 MHz	TBD	
			36 MHz	TBD	
			24 MHz	TBD	
			16 MHz	1	
		Running on HSI clock, code running from Flash, all peripheral disabled (see RCC register description): f_{PCLK1}= f_{HCLK}/2, f_{PCLK2}=f_{HCLK}. AHB pre-scaler used to reduce the frequency	8 MHz	TBD	mA
			4 MHz	TBD	
			2 MHz	TBD	
			1 MHz	TBD	
			500 kHz	TBD	

1. TBD stands for to be determined.
2. Typical values are measures at T_A = 25 °C, V_{DD} = 3.3 V.

Table 14. Typical current consumption in Stop and Standby modes(1)

Symbol	Parameter	Conditions	V_{DD}	Typ(2)	Unit
I_{DD}	Supply current in Stop mode	Regulator in Run mode, Low-speed and high-speed internal RC oscillators OFF High-speed oscillator OFF (no independent watchdog)	3.3 V	24	µA
			2.4 V	TBD	
		Regulator in Low Power mode, Low-speed and high-speed internal RC oscillators OFF, High-speed oscillator OFF (no independent watchdog)	3.3 V	14(3)	
			2.4 V	TBD(3)	
	Supply current in Standby mode(4)	Low-speed internal RC oscillator and independent watchdog OFF	3.3 V	2(3)	µA
			2.4 V	TBD(3)	
		Low-speed internal RC oscillator and independent watchdog ON	3.3 V	3.1(3)	
			2.4 V	TBD(3)	
		Low-speed internal RC oscillator ON, independent watchdog OFF	3.3 V	2.9(3)	
			2.4 V	TBD(3)	
I_{DD_VBAT}	Backup domain supply current	Low-speed oscillator and RTC ON	3.3 V	1.4(3)	µA
			2.4 V	1(3)	
		Low-speed oscillator OFF, RTC ON	3.3 V	0.5(3)	
			2.4 V	TBD(3)	

1. TBD stands for to be determined.
2. Typical values are measures at T_A = 25 °C, V_{DD} = 3.3 V.
3. Values expected for next silicon revision.
4. To obtain Standby consumption with RTC ON, add I_{DD_VBAT} (Low-speed oscillator and RTC ON) to I_{DD} Standby.

5.3.6 External clock source characteristics

High-speed external user clock

The characteristics given in *Table 15* result from tests performed using an high-speed external clock source, and under ambient temperature and supply voltage conditions summarized in *Table 7*.

Table 15. High-speed external (HSE) user clock characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{HSE_ext}	User external clock source frequency(1)			8	25	MHz
V_{HSEH}	OSC_IN input pin high level voltage		$0.7V_{DD}$		V_{DD}	V
V_{HSEL}	OSC_IN input pin low level voltage		V_{SS}		$0.3V_{DD}$	
$t_{w(HSE)}$ $t_{w(HSE)}$	OSC_IN high or low time(1)		16			ns
$t_{r(HSE)}$ $t_{f(HSE)}$	OSC_IN rise or fall time(1)				5	
I_L	OSC_IN Input leakage current	$V_{SS} \le V_{IN} \le V_{DD}$			±1	µA

1. Value based on design simulation and/or technology characteristics. It is not tested in production.

Low-speed external user clock

The characteristics given in *Table 16* result from tests performed using an low-speed external clock source, and under ambient temperature and supply voltage conditions summarized in *Table 7*.

Table 16. Low-speed external user clock characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{LSE_ext}	User External clock source frequency(1)			32.768	1000	kHz
V_{LSEH}	OSC32_IN input pin high level voltage		$0.7V_{DD}$		V_{DD}	V
V_{LSEL}	OSC32_IN input pin low level voltage		V_{SS}		$0.3V_{DD}$	
$t_{w(LSE)}$ $t_{w(LSE)}$	OSC32_IN high or low time(1)		450			ns
$t_{r(LSE)}$ $t_{f(LSE)}$	OSC32_IN rise or fall time(1)				5	
I_L	OSC32_IN Input leakage current	$V_{SS} \le V_{IN} \le V_{DD}$			±1	µA

1. Value based on design simulation and/or technology characteristics. It is not tested in production.

Figure 11. High-speed external clock source AC timing diagram

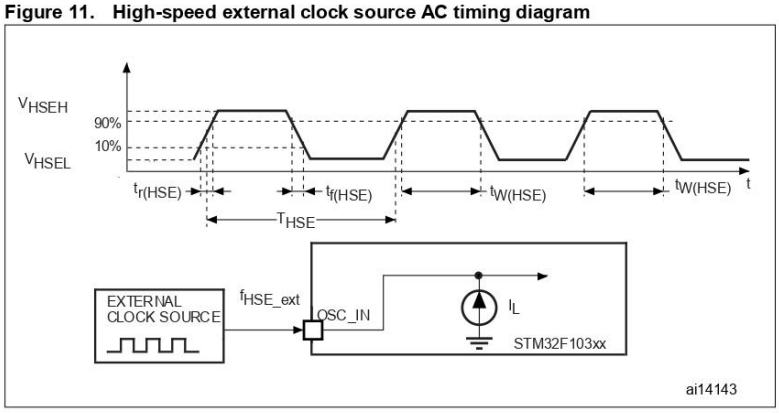
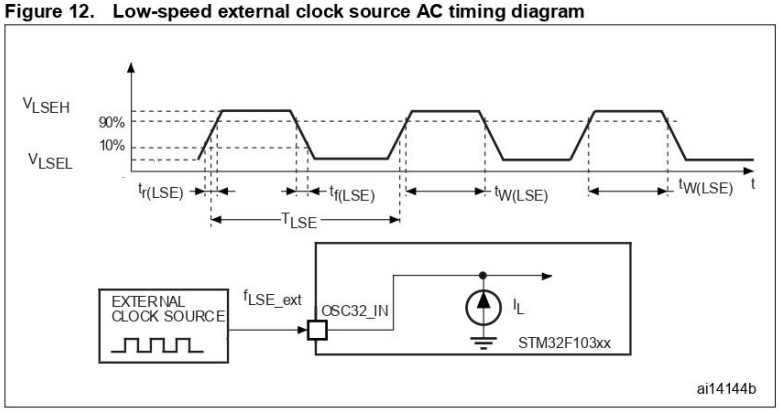


Figure 12. Low-speed external clock source AC timing diagram



High-speed external clock

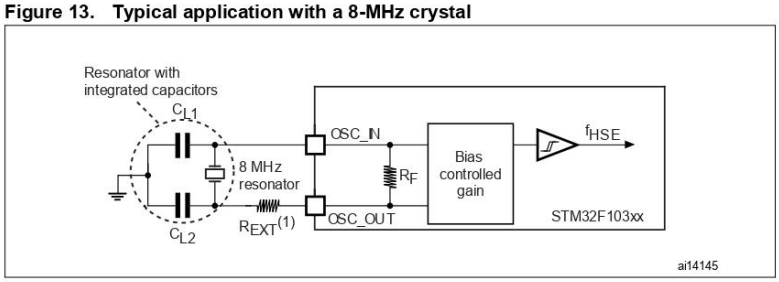
The high-speed external (HSE) clock can be supplied with a 4 to 16 MHz crystal/ceramic resonator oscillator. All the information given in this paragraph are based on characterization results obtained with typical external components specified in *Table 17*. In the application, the resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. Refer to the crystal resonator manufacturer for more details on the resonator characteristics (frequency, package, accuracy).

Table 17. HSE 4-16 MHz oscillator characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
f_{OSC_IN}	Oscillator frequency		4	8	16	MHz
R_F	Feedback resistor			200		kΩ
C_{L1} C_{L2}(2)	Recommended load capacitance versus equivalent serial resistance of the crystal (R_S)(3)	R_S = 30 Ω		30		pF
i_2	HSE driving current	V_{DD}= 3.3 V V_{IN}=V_{SS} with 30 pF load			1	mA
g_m	Oscillator Transconductance	Startup	25			mA/V
$t_{SU(HSE)}$(4)	startup time	V_{SS} is stabilized		2		ms

1. Resonator characteristics given by the crystal/ceramic resonator manufacturer.
2. For C_{L1} and C_{L2} it is recommended to use high-quality ceramic capacitors in the 5 pF to 25pF range (typ.), designed for high-frequency applications, and selected to match the requirements of the crystal or resonator. C_{L1} and C_{L2} are usually the same size. The crystal manufacturer typically specifies a load capacitance which is the series combination of C_{L1} and C_{L2}. PCB and MCU pin capacitance must be included when sizing C_{L1} and C_{L2} (10 pF can be used as a rough estimate of the combined pin and board capacitance).
3. The relatively low value of the RF resistor offers a good protection against issues resulting from use in a humid environment, due to the induced leakage and the bias condition change. However, it is recommended to take this point into account if the MCU is used in tough humidity conditions.
4. $t_{SU(HSE)}$ is the startup time measured from the moment it is enabled (by software) to a stabilized 8 MHz oscillation is reached. This value is measured for a standard crystal resonator and it can vary significantly with the crystal manufacturer

Figure 13. Typical application with a 8-MHz crystal



1. R_{EXT} value depends on the crystal characteristics. Typical value is in the range of 5 to 6R_S.

Low-speed external clock

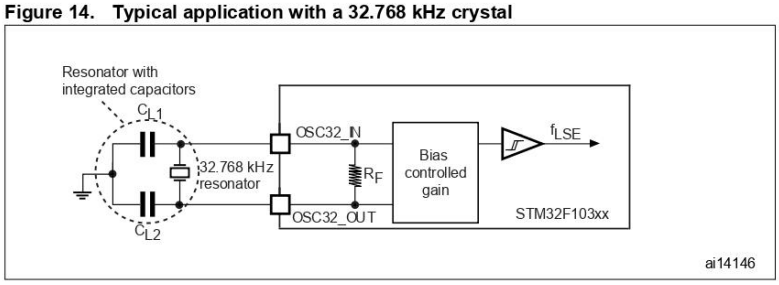
The low-speed external (LSE) clock can be supplied with a 32.768 kHz crystal/ceramic resonator oscillator. All the information given in this paragraph are based on characterization results obtained with typical external components specified in *Table 18*. In the application, the resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. Refer to the crystal resonator manufacturer for more details on the resonator characteristics (frequency, package, accuracy).

Table 18. LSE oscillator characteristics (f_{LSE} = 32.768 kHz)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_F	Feedback resistor			5		MΩ
C_{L1} C_{L2}	Recommended load capacitance versus equivalent serial resistance of the crystal (R_S)(1)	R_S = 30 kΩ			15	pF
I_2	LSE driving current	V_{DD} = 3.3 V V_{IN} = V_{SS}			1.4	µA
g_m	Oscillator Transconductance		5			µA/V
$t_{SU(LSE)}$(2)	startup time	V_{SS} is stabilized		3		s

1. The oscillator selection can be optimized in terms of supply current using an high quality resonator with small R_S value for example MSIV-TIN32.768kHz. Refer to crystal manufacturer for more details
2. $t_{SU(LSE)}$ is the startup time measured from the moment it is enabled (by software) to a stabilized 32.768 kHz oscillation is reached. This value is measured for a standard crystal resonator and it can vary significantly with the crystal manufacturer

Figure 14. Typical application with a 32.768 kHz crystal



5.3.7 Internal clock source characteristics

The parameters given in *Table 19* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

High-speed internal (HSI) RC oscillator

Table 19. HSI oscillator characteristics(1)(2)

Symbol	Parameter	Conditions	Min	Typ	Max(3)	Unit
f_{HSI}	Frequency			8		MHz
ACC_{HSI}	Accuracy of HSI oscillator	T_A = –40 to 105 °C	TBD	±3	TBD	%
		at T_A = 25°C	TBD	±1	TBD	%
$t_{su(HSI)}$	HSI oscillator start up time		1		2	µs
$I_{DD(HSI)}$	HSI oscillator power consumption			80	100	µA

1. V_{DD} = 3.3 V, T_A = -40 to 105 °C unless otherwise specified.
2. TBD stands for to be determined.
3. Values based on device characterization, not tested in production.

LSI Low Speed Internal RC Oscillator

Table 20. LSI oscillator characteristics (1)

Symbol	Parameter	Conditions	Min	Typ	Max(2)	Unit
f_{LSI}	Frequency		30		60	kHz
$t_{su(LSI)}$	LSI oscillator start up time				85	µs
$I_{DD(LSI)}$	LSI oscillator power consumption			0.65	1.2	µA

1. V_{DD} = 3 V, T_A = -40 to 105 °C unless otherwise specified.
2. Value based on device characterization, not tested in production.

Wakeup time from low power mode

The wakeup times given in *Table 21* is measured on a wakeup phase with a 8-MHz HSI RC oscillator. The clock source used to wake up the device depends from the current operating mode:

- Stop or Standby mode: the clock source is the RC oscillator
- Sleep mode: the clock source is the clock that was set before entering Sleep mode.

All timings are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 21. Low-power mode wakeup timings(1)

Symbol	Parameter	Conditions	Typ	Max	Unit
$t_{WUSLEEP}$(2)	Wakeup from Sleep mode	Wakeup on HSI RC clock	0.75	TBD	µs
t_{WUSTOP}(2)	Wakeup from Stop mode (regulator in run mode)	HSI RC wakeup time = 2 µs	4	TBD	µs
	Wakeup from Stop mode (regulator in low power mode)	HSI RC wakeup time = 2 µs, Regulator wakeup from LP mode time = 5 µs	7	TBD	
$t_{WUSTDBY}$(3)	Wakeup from Standby mode	HSI RC wakeup time = 2 µs, Regulator wakeup from power down time = 38 µs	40	TBD	µs

1. TBD stands for to be determined.
2. The wakeup time from Sleep and Stop mode are measured from the wakeup event to the point in which the user application code reads the first instruction.
3. The wakeup time from Standby mode is measured from the wakeup event to the point in which the device exits from reset.

5.3.8 PLL characteristics

The parameters given in *Table 22* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 22. PLL characteristics(1)

Symbol	Parameter	Test Conditions	Value			Unit
			Min	Typ	Max(2)	
f_{PLL_IN}	PLL input clock			8.0		MHz
	PLL input clock duty cycle		40		60	%
f_{PLL_OUT}	PLL multiplier output clock		16		72	MHz
f_{VCO}	VCO frequency range	When PLL operates (locked)	32		144	MHz
t_{LOCK}	PLL lock time				200	µs
t_{JITTER}	Cycle to cycle jitter (+/-3Σ peak to peak)	V_{DD} is stable	TBD		TBD	%

1. TBD stands for to be determined.
2. Data based on device characterization, not tested in production.

5.3.9 Memory characteristics

Flash memory

The characteristics are given at T_A = -40 to 105 °C unless otherwise specified.

Table 23. Flash memory characteristics

Symbol	Parameter	Conditions	Min	Typ	Max(1)	Unit
t_{prog}	Word programming time	T_A = -40 to +105 °C	20		40	µs
t_{ERASE}	Page (1kB) erase time	T_A = -40 to +105 °C	20		40	ms
t_{ME}	Mass erase time	T_A = -40 to +105 °C	20		40	ms
I_{DD}	Supply current	Read mode f_{HCLK} = 72 MHz with 2 wait states, V_{DD} = 3.3 V			20	mA
		Write / Erase modes f_{HCLK} = 72 MHz, V_{DD} = 3.3 V			5	mA
		Power-down mode / HALT, V_{DD} = 3.0 to 3.6 V			50	µA

1. Values based on characterization and not tested in production.

Table 24. Flash memory endurance and data retention

Symbol	Parameter	Conditions	Value			Unit
			Min(1)	Typ	Max	
N_{END}	Endurance		1	10		kcycles
t_{RET}	Data retention	T_A = 85 °C	30			Years

1. Values based on characterization not tested in production.

5.3.10 EMC characteristics

Susceptibility tests are performed on a sample basis during device characterization.

Functional EMS (electromagnetic susceptibility)

While a simple application is executed on the device (toggling 2 LEDs through I/O ports). the device is stressed by two electromagnetic events until a failure occurs. The failure is indicated by the LEDs:

- **Electrostatic discharge (ESD)** (positive and negative) is applied to all device pins until a functional disturbance occurs. This test is compliant with the IEC 1000-4-2 standard.
- **FTB**: A Burst of Fast Transient voltage (positive and negative) is applied to V_{DD} and V_{SS} through a 100 pF capacitor, until a functional disturbance occurs. This test is compliant with the IEC 1000-4-4 standard.

A device reset allows normal operations to be resumed.

The test results are given in *Table 25*. They are based on the EMS levels and classes defined in application note AN1709.

Table 25. EMS characteristics(1)

Symbol	Parameter	Conditions	Level/ Class
V_{FESD}	Voltage limits to be applied on any I/O pin to induce a functional disturbance	V_{DD} = 3.3 V, T_A = +25 °C, f_{HCLK}=48 MHz conforms to IEC 1000-4-2	TBD
V_{EFTB}	Fast transient voltage burst limits to be applied through 100pF on V_{DD} and V_{SS} pins to induce a functional disturbance	V_{DD} = 3.3 V, T_A = +25 °C, f_{HCLK} = 48 MHz conforms to IEC 1000-4-4	4A

1. TBD stands for to be determined.

Designing hardened software to avoid noise problems

EMC characterization and optimization are performed at component level with a typical application environment and simplified MCU software. It should be noted that good EMC performance is highly dependent on the user application and the software in particular.

Therefore it is recommended that the user applies EMC software optimization and prequalification tests in relation with the EMC level requested for his application.

Software recommendations

The software flowchart must include the management of runaway conditions such as:

- Corrupted program counter
- Unexpected reset
- Critical Data corruption (control registers...)

Prequalification trials

Most of the common failures (unexpected reset and program counter corruption) can be reproduced by manually forcing a low state on the NRST pin or the Oscillator pins for 1 second.

To complete these trials, ESD stress can be applied directly on the device, over the range of specification values. When unexpected behavior is detected, the software can be hardened to prevent unrecoverable errors occurring (see application note AN1015).

Electromagnetic Interference (EMI)

The electromagnetic field emitted by the device are monitored while a simple application is executed (toggling 2 LEDs through the I/O ports). This emission test is compliant with SAE J 1752/3 standard which specifies the test board and the pin loading.

Table 26. EMI characteristics

Symbol	Parameter	Conditions	Monitored Frequency Band	Max vs. [f_{HSE}/f_{HCLK}]		Unit
				8/48 MHz	8/72 MHz	
S_{EMI}	Peak level	V_{DD} = 3.3 V, T_A = 2 5 °C, LQFP100 package compliant with SAE J 1752/3	0.1 to 30 MHz	12	12	dBµV
			30 to 130 MHz	22	19	
			130 MHz to 1GHz	23	29	
			SAE EMI Level	4	4	-

5.3.11 Absolute maximum ratings (electrical sensitivity)

Based on three different tests (ESD, LU) using specific measurement methods, the device is stressed in order to determine its performance in terms of electrical sensitivity.

Electrostatic discharge (ESD)

Electrostatic discharges (a positive then a negative pulse separated by 1 second) are applied to the pins of each sample according to each pin combination. The sample size is either 3 parts (cumulative mode) or 3 parts × (n + 1) supply pins (non-cumulative mode). The human body model (HBM) can be simulated. The tests are compliant with JESD22-A114A standard.
For more details, refer to the application note AN1181.

Table 27. ESD absolute maximum ratings(1)

Symbol	Ratings	Conditions	Maximum value(2)	Unit
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	T_A = +25 °C	2000	V
$V_{ESD(CDM)}$	Electrostatic discharge voltage (charge device model)		TBD	

1. TBD stands for to be determined.
2. Values based on characterization results, not tested in production.

Static latch-up

Two complementary static tests are required on six parts to assess the latch-up performance:

- A supply overvoltage is applied to each power supply pin
- A current injection is applied to each input, output and configurable I/O pin

These tests are compliant with EIA/JESD 78A IC latch-up standard.

Table 28. Electrical sensitivities

Symbol	Parameter	Conditions	Class
LU	Static latch-up class	T_A = +105 °C	II level A

5.3.12 I/O port pin characteristics

General input/output characteristics

Unless otherwise specified, the parameters given in *Table 29* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

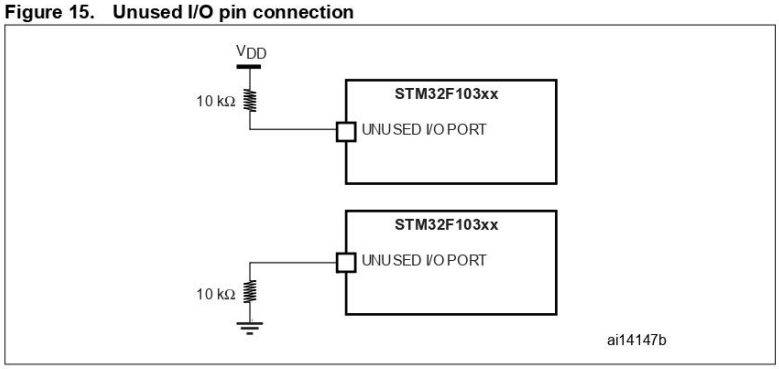
All unused pins must be held at a fixed voltage, by using the I/O output mode, an external pull-up or pull-down resistor (see *Figure 15*).

Table 29. I/O static characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IL}	Input low level voltage(2)	TTL ports	–0.5		0.8	V
V_{IH}	IO TC input high level voltage(2)		2		V_{DD}+0.5	
	IO FT high level voltage(2)		2		5.5V	
V_{IL}	Input low level voltage(2)	CMOS ports	–0.5		0.35 V_{DD}	V
V_{IH}	Input high level voltage(2)		0.65 V_{DD}		V_{DD}+0.5	
V_{hys}	IO TC Schmitt trigger voltage hysteresis(3)			200		mV
	IO TC Schmitt trigger voltage hysteresis(3)			5% V_{DD}(4)		mV
I_{lkg}	Input leakage current (5)	$V_{SS} \le V_{IN} \le V_{DD}$ Standard I/Os			±1	µA
		V_{IN}= 5 V 5 V tolerant I/Os			3	
R_{PU}	Weak pull-up equivalent resistor(6)	$V_{IN} = V_{SS}$	30	40	50	kΩ
R_{PD}	Weak pull-down equivalent resistor(6)	$V_{IN} = V_{DD}$	30	40	50	kΩ
C_{IO}	I/O pin capacitance			5		pF

1. V_{DD} = 3.3 V, T_A = -40 to 105 °C unless otherwise specified.
2. Values based on characterization results, and not tested in production.
3. Hysteresis voltage between Schmitt trigger switching levels. Based on characterization results, not tested.
4. With a minimum of 100 mV.
5. Leakage could be higher than max. if negative current is injected on adjacent pins.
6. Pull-up and pull-down resistors are designed with a true resistance in series with a switchable PMOS/NMOS. This MOS/NMOS contribution to the series resistance is minimum (~10% order).

Figure 15. Unused I/O pin connection



Output driving current

The GPIOs (general purpose input/outputs) can sink or source up to +/-8 mA, and sink +20 mA (with a relaxed V_{OL}).

In the user application, the number of I/O pins which can drive current must be limited to respect the absolute maximum rating specified in *Section 5.2*:

- The sum of the currents sourced by all the I/Os on $V_{DD,}$ plus the maximum Run consumption of the MCU sourced on $V_{DD,}$ cannot exceed the absolute maximum rating I_{VDD} (see *Table 5*).
- The sum of the currents sunk by all the I/Os on V_{SS} plus the maximum Run consumption of the MCU sunk on V_{SS} cannot exceed the absolute maximum rating I_{VSS} (see *Table 5*).

Output voltage levels

Unless otherwise specified, the parameters given in *Table 30* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 30. Output voltage characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	TTL port I_{IO} = +8 mA 2.7 V < V_{DD} < 3.6 V		0.4	V
V_{OH} (2)	Output high level voltage for an I/O pin when 4 pins are sourced at same time		V_{DD}–0.4		
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	CMOS port I_{IO} =+ 8mA 2.7 V < V_{DD} < 3.6 V		0.4	V
V_{OH} (2)	Output high level voltage for an I/O pin when 4 pins are sourced at same time		2.4		
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	I_{IO} = +20 mA 2.7 V < V_{DD} < 3.6 V		1.3	V
V_{OH} (2)	Output high level voltage for an I/O pin when 4 pins are sourced at same time		V_{DD}–1.3		
V_{OL} (1)	Output low level voltage for an I/O pin when 8 pins are sunk at same time	I_{IO} = +6 mA 2 V < V_{DD} < 2.7 V		0.4	V
V_{OH} (2)	Output high level voltage for an I/O pin when 4 pins are sourced at same time		V_{DD}–0.4		

1. The I_{IO} current sunk by the device must always respect the absolute maximum rating specified in *Table 5* and the sum of I_{IO} (I/O ports and control pins) must not exceed I_{VSS}.
2. The I_{IO} current sourced by the device must always respect the absolute maximum rating specified in *Table 5* and the sum of I_{IO} (I/O ports and control pins) must not exceed I_{VDD}.

Input/output AC characteristics

The definition and values of input/output AC characteristics are given in *Figure 16* and *Table 31*, respectively.

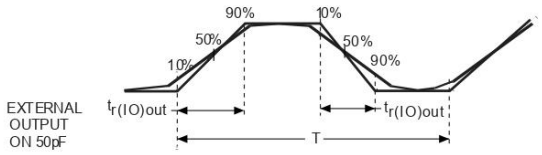
Unless otherwise specified, the parameters given in *Table 31* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 31. I/O AC characteristics(1)

I/O mode(1)	Symbol	Parameter	Conditions	Min	Max	Unit
10	$f_{max(IO)out}$	Maximum frequency(2)	C_L = 50 pF, V_{DD} = 2 V to 3.6 V		2	MHz
	$t_{f(IO)out}$	Output high to low level fall time(3)	C_L = 50 pF, V_{DD} = 2 V to 3.6 V		125	ns
	$t_{r(IO)out}$	Output low to high level rise time(3)			125	
01	$f_{max(IO)out}$	Maximum frequency(2)	C_L = 50 pF, V_{DD} = 2 V to 3.6 V		10	MHz
	$t_{f(IO)out}$	Output high to low level fall time(3)	C_L = 50 pF, V_{DD} = 2 V to 3.6 V		25	ns
	$t_{r(IO)out}$	Output low to high level rise time(3)			25	
11	$F_{max(IO)out}$	Maximum frequency(2)	C_L = 30 pF, V_{DD} = 2.7 V to 3.6 V		50	MHz
			C_L = 50 pF, V_{DD} = 2.7 V to 3.6 V		30	MHz
			C_L = 50 pF, V_{DD} = 2 V to 2.7 V		20	MHz
	$t_{f(IO)out}$	Output high to low level fall time(3)	C_L = 30 pF, V_{DD} = 2.7 V to 3.6 V		5	ns
			C_L = 50 pF, V_{DD} = 2.7 V to 3.6 V		8	
			C_L = 50 pF, V_{DD} = 2 V to 2.7 V		12	
	$t_{r(IO)out}$	Output low to high level rise time(3)	C_L = 30 pF, V_{DD} = 2.7 V to 3.6 V		5	
			C_L = 50 pF, V_{DD} = 2.7 V to 3.6 V		8	
			C_L = 50 pF, V_{DD} = 2 V to 2.7 V		12	
-	t_{EXTIpw}	Pulse width of external signals detected by the EXTI controller		10		ns

1. Refer to the Reference user manual UM0306 for a description of GPIO Port configuration register.
2. The maximum frequency is defined in *Figure 16*.
3. Values based on design simulation and validated on silicon, not tested in production.

Figure 16. I/O AC characteristics definition



5.3.13 NRST pin characteristics

The NRST pin input driver uses CMOS technology. It is connected to a permanent pull-up resistor, R_{PU} (see *Table 29*).

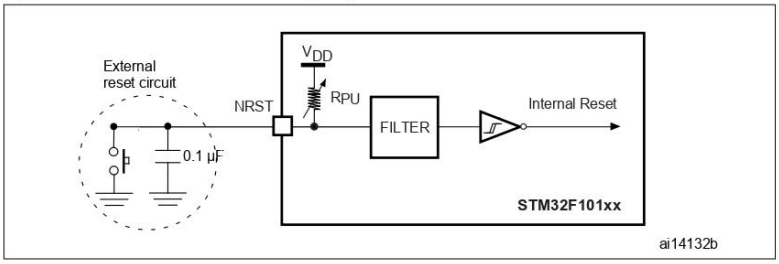
Unless otherwise specified, the parameters given in *Table 32* are derived from tests performed under ambient temperature and V_{DD} supply voltage conditions summarized in *Table 7*.

Table 32. NRST pin characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{IL(NRST)}$	NRST Input low level voltage		–0.5		0.8	V
$V_{IH(NRST)}$	NRST Input high level voltage		2		V_{DD}+0.5	
$V_{hys(NRST)}$	NRST Schmitt trigger voltage hysteresis			200		
R_{PU}	Weak pull-up equivalent resistor(2)	$V_{IN} = V_{SS}$	30	40	50	kΩ
$V_{F(NRST)}$	NRST Input filtered pulse(3)				100	ns
$V_{NF(NRST)}$	NRST Input not filtered pulse(3)		300			µs

1. TBD stands for to be determined.
2. The pull-up is designed with a true resistance in series with a switchable PMOS. This PMOS contribution to the series resistance must be minimum (~10% order).
3. Values guaranteed by design, not tested in production.

Figure 17. Recommended NRST pin protection



2. The reset network protects the device against parasitic resets.
3. The user must ensure that the level on the NRST pin can go below the $V_{IL(NRST)}$ max level specified in *Table 32*. Otherwise the reset will not be taken into account by the device.

5.3.14 TIM timer characteristics

Unless otherwise specified, the parameters given in *Table 33* are derived from tests performed under ambient temperature, f_{PCLKx} frequency and V_{DD} supply voltage conditions summarized in *Table 7*.

Refer to *Section 5.3.12: I/O port pin characteristics* for details on the input/output alternate function characteristics (output compare, input capture, external clock, PWM output).

Table 33. TIMx(1) characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
$t_{res(TIM)}$	Timer resolution time		1		$t_{TIMxCLK}$
		$f_{TIMxCLK}$ = 72 MHz	13.9		ns
f_{EXT}	Timer external clock frequency on CH1 to CH4		0	$f_{TIMxCLK}/2$	MHz
		$f_{TIMxCLK}$ = 72 MHz	0	36	MHz
Res_{TIM}	Timer resolution			16	bit
$t_{COUNTER}$	16-bit counter clock period when internal clock is selected		1	65536	$t_{TIMxCLK}$
		$f_{TIMxCLK}$ = 72 MHz	0.0139	910	µs
t_{MAX_COUNT}	Maximum possible count			65536 × 65536	$t_{TIMxCLK}$
		$f_{TIMxCLK}$ = 72 MHz		59.6	s

1. TIMx is used as a general term to refer to the TIM1, TIM2, TIM3 and TIM4 timers.

5.3.15 Communications interfaces

I^2C interface characteristics

Unless otherwise specified, the parameters given in *Table 34* are derived from tests performed under ambient temperature, f_{PCLK1} frequency and V_{DD} supply voltage conditions summarized in *Table 7*.

The STM32F103xx performance line I^2C interface meets the requirements of the standard I^2C communication protocol with the following restrictions: the I/O pins SDA and SCL are mapped to are not “true” open-drain. When configured as open-drain, the PMOS connected between the I/O pin and V_{DD} is disabled, but is still present. In addition, there is a protection diode between the I/O pin and V_{DD}. As a consequence, when multiple master devices are connected to the I^2C bus, it is not possible to power off the STM32F103xx while another I^2C master node remains powered on. Otherwise, the STM32F103xx would be powered by the protection diode.

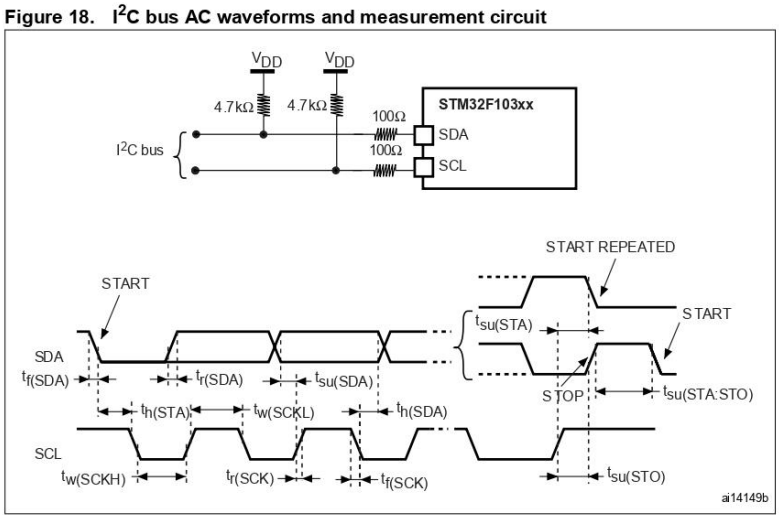
The I^2C characteristics are described in *Table 34*. Refer also to *Section 5.3.12: I/O port pin characteristics* for more details on the input/output alternate function characteristics (SDA and SCL).

Table 34. I^2C characteristics

Symbol	Parameter	Standard mode I^2C(1)		Fast mode I^2C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{w(SCLL)}$	SCL clock low time	4.7		1.3		µs
$t_{w(SCLH)}$	SCL clock high time	4.0		0.6		
$t_{su(SDA)}$	SDA setup time	250		100		ns
$t_{h(SDA)}$	SDA data hold time	0(3)		0(4)	900(3)	
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time		1000	20 + 0.1C_b	300	
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time		300	20 + 0.1C_b	300	
$t_{h(STA)}$	Start condition hold time	4.0		0.6		µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7		0.6		
$t_{su(STO)}$	Stop condition setup time	4.0		0.6		µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7		1.3		µs
C_b	Capacitive load for each bus line		400		400	pF

1. Values based on standard I^2C protocol requirement, not tested in production.
2. f_{PCLK1} must be higher than 2 MHz to achieve the maximum standard mode I^2C frequency. It must be higher than 4 MHz to achieve the maximum fast mode I^2C frequency.
3. The maximum hold time of the Start condition has only to be met if the interface does not stretch the low period of SCL signal.
4. The device must internally provide a hold time of at least 300ns for the SDA signal in order to bridge the undefined region of the falling edge of SCL.

Figure 18. I²C bus AC waveforms and measurement circuit



1. Measurement points are done at CMOS levels: $0.3V_{DD}$ and $0.7V_{DD}$.

Table 35. SCL frequency (f_{PCLK1}= 36 MHz.,V_{DD} = 3.3 V)(1)(2)(3)

f_{SCL} (kHz)	I2C_CCR value
	R_P = 4.7 kΩ
400	TBD
300	TBD
200	TBD
100	TBD
50	TBD
20	TBD

1. TBD = to be determined.
2. R_P = External pull-up resistance, f_{SCL} = I²C speed,
3. For speeds around 200 kHz, the tolerance on the achieved speed is of ±5%. For other speed ranges, the tolerance on the achieved speed ±2%. These variations depend on the accuracy of the external components used to design the application.

SPI interface characteristics

Unless otherwise specified, the parameters given in *Table 36* are derived from tests performed under ambient temperature, f_{PCLKx} frequency and V_{DD} supply voltage conditions summarized in *Table 7*.

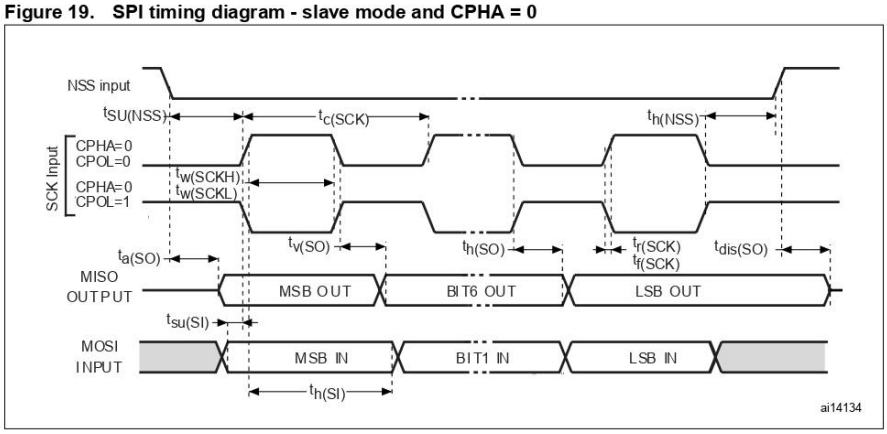
Refer to *Section 5.3.12: I/O port pin characteristics* for more details on the input/output alternate function characteristics (NSS, SCK, MOSI, MISO).

Table 36. SPI characteristics(1)

Symbol	Parameter	Conditions	Min	Max	Unit
f_{SCK} $1/t_{c(SCK)}$	SPI clock frequency	Master mode	TBD	TBD	MHz
		Slave mode	0	TBD	
$t_{r(SCK)}$ $t_{f(SCK)}$	SPI clock rise and fall time	Capacitive load: C=50 pF		TBD	ns
$t_{su(NSS)}$(2)	NSS setup time	Slave mode	0		
$t_{h(NSS)}$(2)	NSS hold time	Slave mode	0		
$t_{w(SCKH)}$(2) $t_{w(SCKL)}$(2)	SCK high and low time	Master mode, f_{PCLK}= TBD, presc = TBD	TBD		
$t_{su(MI)}$(2) $t_{su(SI)}$(2)	Data input setup time	Master mode	TBD		
		Slave mode	TBD		
$t_{h(MI)}$(2) $t_{h(SI)}$(2)	Data input hold time	Master mode	TBD		
		Slave mode	TBD		
		Master mode, f_{PCLK}= TBD	TBD(3)		
		Slave mode, f_{PCLK}= TBD	TBD(3)		
$t_{a(SO)}$(2)(4)	Data output access time	Slave mode	TBD	TBD	
		Slave mode, f_{PCLK}= TBD	TBD	TBD	
$t_{dis(SO)}$(2)(5)	Data output disable time	Slave mode	TBD	TBD	
$t_{v(SO)}$(2)(1)	Data output valid time	Slave mode (after enable edge)		TBD	
		f_{PCLK}= TBD		TBD	
$t_{v(MO)}$(2)(1)	Data output valid time	Master mode (after enable edge)		TBD	
		f_{PCLK}= TBD	TBD	TBD	
$t_{h(SO)}$(2)	Data output hold time	Slave mode (after enable edge)	TBD		
$t_{h(MO)}$(2)		Master mode (after enable edge)	TBD		

1. TBD = to be determined.
2. Values based on design simulation and/or characterization results, and not tested in production.
3. Depends on f_{PCLK}. For example, if f_{PCLK}= 8MHz, then t_{PCLK} = $1/f_{PLCLK}$ =125 ns and $t_{v(MO)}$ = 255 ns.
4. Min time is for the minimum time to drive the output and the max time is for the maximum time to validate the data.
5. Min time is for the minimum time to invalidate the output and the max time is for the maximum time to put the data in Hi-Z

Figure 19. SPI timing diagram - slave mode and CPHA = 0



1. Measurement points are done at CMOS levels: $0.3V_{DD}$ and $0.7V_{DD}$.

Figure 20. SPI timing diagram - slave mode and CPHA = 1[1)]

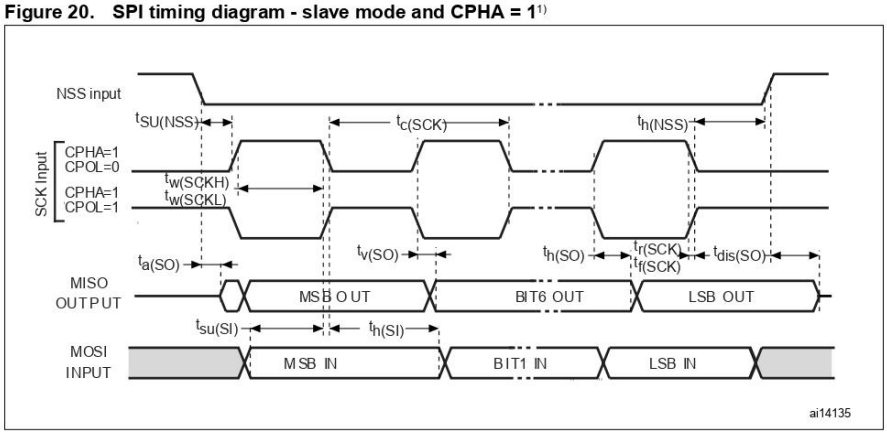
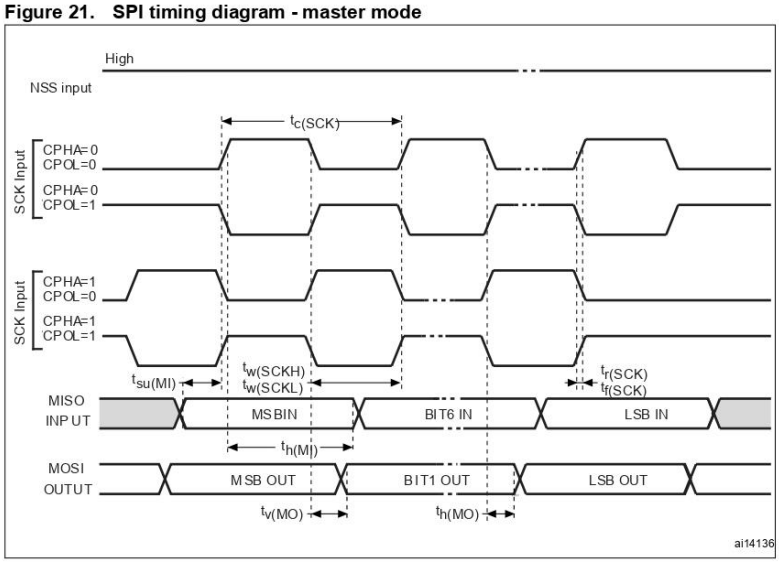


Figure 21. SPI timing diagram - master mode



1. Measurement points are done at CMOS levels: $0.3V_{DD}$ and $0.7V_{DD}$.

USB characteristics

The USB interface is USB-IF certified (Full Speed).

Table 37. USB DC electrical characteristics

Symbol	Parameter	Conditions	Min.(1)	Max.(1)	Unit
Input levels					
V_{DI}	Differential input sensitivity	I(USBDP, USBDM)	0.2		V
V_{CM}	Differential common mode range	Includes V_{DI} range	0.8	2.5	
V_{SE}	Single ended receiver threshold		1.3	2.0	
Output levels					
V_{OL}	Static output level low	R_L of 1.5 kΩ to 3.6 V(2)		0.3	V
V_{OH}	Static output level high	R_L of 15 kΩ to V_{SS}(2)	2.8	3.6	

1. All the voltages are measured from the local ground potential.
2. R_L is the load connected on the USB drivers

Figure 22. USB timings: definition of data signal rise and fall time

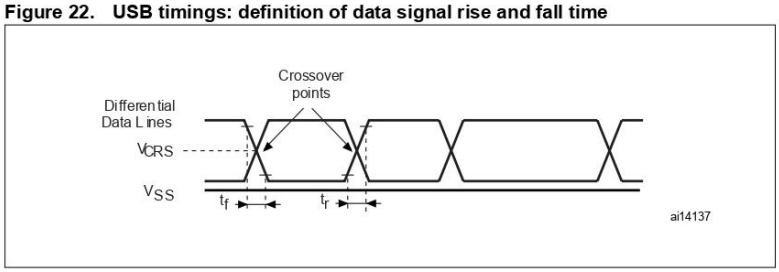


Table 38. USB: Full speed electrical characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
Driver characteristics					
t_r	Rise time(1)	C_L = 50 pF	4	20	ns
t_f	Fall Time(1)	C_L = 50 pF	4	20	ns
t_{rfm}	Rise/ fall time matching	t_r/t_f	90	110	%
V_{CRS}	Output signal crossover voltage		1.3	2.0	V

1. Measured from 10% to 90% of the data signal. For more detailed informations, please refer to USB Specification - Chapter 7 (version 2.0).

5.3.16 CAN (controller area network) interface

Refer to I/O port characteristics for more details on the input/output alternate function characteristics (CANTX and CANRX).

5.3.17 12-bit ADC characteristics

Unless otherwise specified, the parameters given in *Table 39* are derived from tests performed under ambient temperature, f_{PCLK2} frequency and V_{DDA} supply voltage conditions summarized in *Table 7*.

Note: It is recommended to perform a calibration after each power-up.

Table 39. ADC characteristics(1)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{DDA}	ADC power supply		2.4V		3.6V	V
V_{REF+}	Positive reference voltage		2.0		V_{DDA}	V
f_{ADC}	ADC clock frequency		0.6		14	MHz
f_S	Sampling rate	TBD	0.05		1	MHz
f_{TRIG}	External trigger frequency	f_{ADC} = 14 MHz			823	kHz
					17	$1/f_{ADC}$
V_{AIN}	Conversion voltage range(2)		V_{SSA}		V_{DDA}	V

Table 39. ADC characteristics(1) (continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
R_{AIN}	External input impedance		TBD(2)(3)			kΩ
C_{AIN}	External capacitor on analog input					pF
I_{lkg}	Negative input leakage current on analog pins	$V_{IN} < V_{SS}$, $\lvert I_{IN} \rvert$ < 400 µA on adjacent analog pin		5	6	µA
R_{ADC}	Sampling switch resistance				1	kΩ
C_{ADC}	Internal sample and hold capacitor				5	pF
t_{CAL}	Calibration time	f_{ADC} = 14MHz	5.9			µs
			83			$1/f_{ADC}$
t_{lat}	Injection conversion latency	f_{ADC} = 14 MHz			0.214	µs
					3	$1/f_{ADC}$
t_S	Sampling time	f_{ADC} = 14 MHz	0.107		17.1	µs
t_{STAB}	Power-up time		0	0	1	µs
t_{CONV}	Total conversion time (including sampling time)	f_{ADC} = 14 MHz	1		18	µs
			14 (1.5 for sampling +12.5 for successive approximation)			$1/f_{ADC}$

1. TBD = to be determined.
2. Depending on the input signal variation (f_{AIN}), C_{AIN} can be increased for stabilization time and reduced to allow the use of a larger serial resistor (R_{AIN}). It is valid for all f_{ADC} frequencies ≤14 MHz.
3. During the sample time the input capacitance C_{AIN} (5 max) can be charged/discharged by the external source. The internal resistance of the analog source must allow the capacitance to reach its final voltage level within t_S. After the end of the sample time t_S, changes of the analog input voltage have no effect on the conversion result. Values for the sample clock t_S depend on programming.

Table 40. ADC accuracy (f_{PCLK2} = 14 MHz, f_{ADC} = 14 MHz, R_{AIN} <10 kΩ, V_{DDA} = 3.3 V)(1)

Symbol	Parameter	Conditions	Typ	Max	Unit
$\lvert E_T \rvert$	Total unadjusted error(2)		3	TBD	LSB
$\lvert E_O \rvert$	Offset error(2)		1	TBD	
$\lvert E_G \rvert$	Gain Error(2)		2	TBD	
$\lvert E_D \rvert$	Differential linearity error(2)		3	TBD	
$\lvert E_L \rvert$	Integral linearity error(2)		2	TBD	

1. TBD = to be determined.
2. ADC Accuracy vs. Negative Injection Current: Injecting negative current on any of the standard (non-robust) analog input pins should be avoided as this significantly reduces the accuracy of the conversion being performed on another analog input. It is recommended to add a Schottky diode (pin to ground) to standard analog pins which may potentially inject negative current.
Any positive injection current within the limits specified for $I_{INJ(PIN)}$ and $\Sigma I_{INJ(PIN)}$ in *Section 5.3.12* does not affect the ADC accuracy.

Figure 23. ADC accuracy characteristics

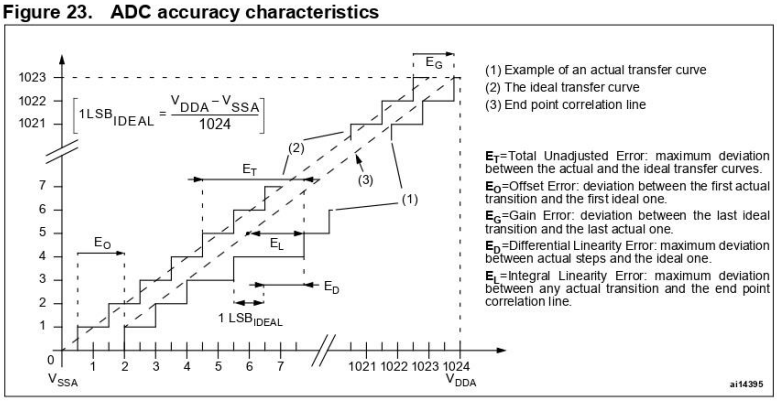
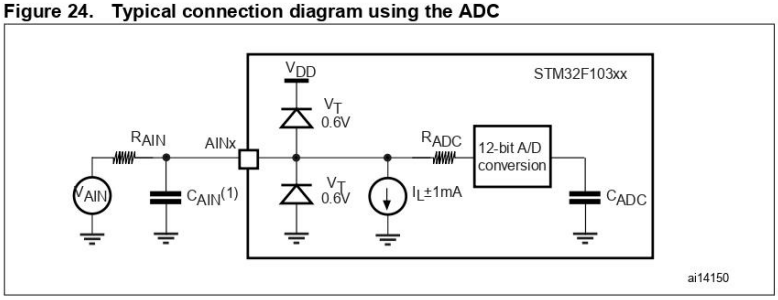


Figure 24. Typical connection diagram using the ADC

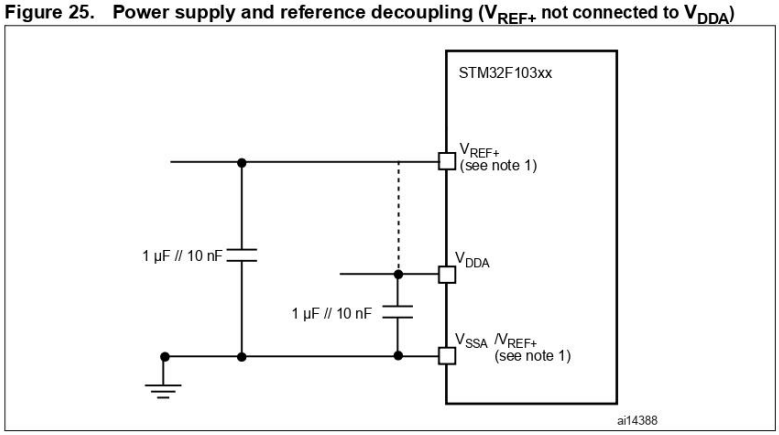


1. Refer to *Table 39* for the values of R_{ADC} and C_{ADC}.
2. $C_{PARASITIC}$ must be added to C_{AIN}. It represents the capacitance of the PCB (dependent on soldering and PCB layout quality) plus the pad capacitance (3 pF). A high $C_{PARASITIC}$ value will downgrade conversion accuracy. To remedy this, f_{ADC} should be reduced.

General PCB design guidelines

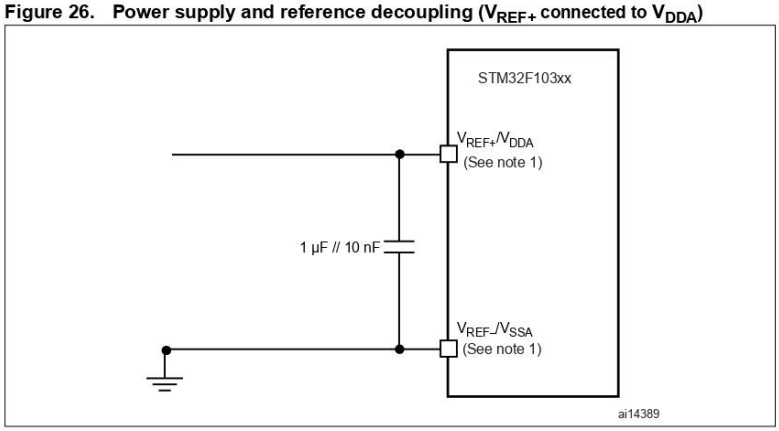
Power supply decoupling should be performed as shown in *Figure 25* or *Figure 26*, depending on whether V_{REF+} is connected to V_{DDA} or not. The 10 nF capacitors should be ceramic (good quality). They should be placed them as close as possible to the chip.

Figure 25. Power supply and reference decoupling (V_{REF+} not connected to V_{DDA})



1. V_{REF+} and V_{REF-} inputs are available only on 100-pin packages.

Figure 26. Power supply and reference decoupling (V_{REF+} connected to V_{DDA})



1. V_{REF+} and V_{REF-} inputs are available only on 100-pin packages.

5.3.18 Temperature sensor characteristics

Table 41. TS characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
T_L	V_{SENSE} linearity with temperature			±1.5		°C
Avg_Slope	Average slope			4.478		mV/°C
V_{25}	Voltage at 25 °C			1.4		V
t_{START}	Startup time		4		10	µs

6 Package characteristics

Figure 27. LFBGA100 - low profile fine pitch ball grid array package outline

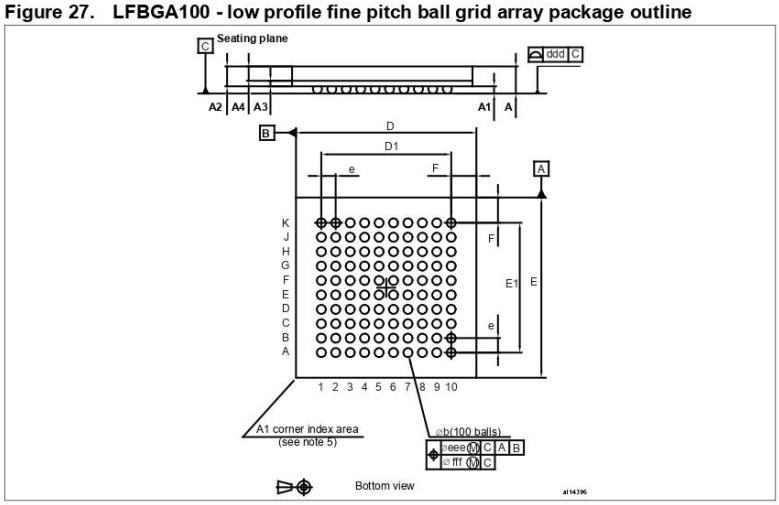
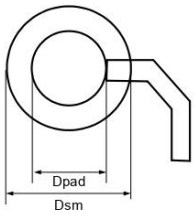


Table 42. LFBGA100 - low profile fine pitch ball grid array package mechanical data

Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A			1.700			0.067
A1	0.270			0.011		
A2		1.085			0.043	
A3		0.30			0.012	
A4			0.80			0.031
b	0.45	0.50	0.55	0.018	0.020	0.022
D	9.85	10.00	10.15	0.388	0.394	0.40
D1		7.20			0.283	
E	9.85	10.00	10.15	0.388	0.394	0.40
E1		7.20			0.283	
e		0.80			0.031	
F		1.40			0.055	
ddd			0.12			0.005
eee			0.15			0.006
fff			0.08			0.003
N (number of balls)	100					

Figure 28. Recommended PCB design rules (0.80/0.75 mm pitch BGA)



Dpad	0.37 mm
Dsm	0.52 mm typ. (depends on solder mask registration tolerance
Solder paste	0.37 mm aperture diameter
– Non solder mask defined pads are recommended – 4 to 6 mils screen print	

Figure 29. LQFP100 – 100-pin low-profile quad flat package outline

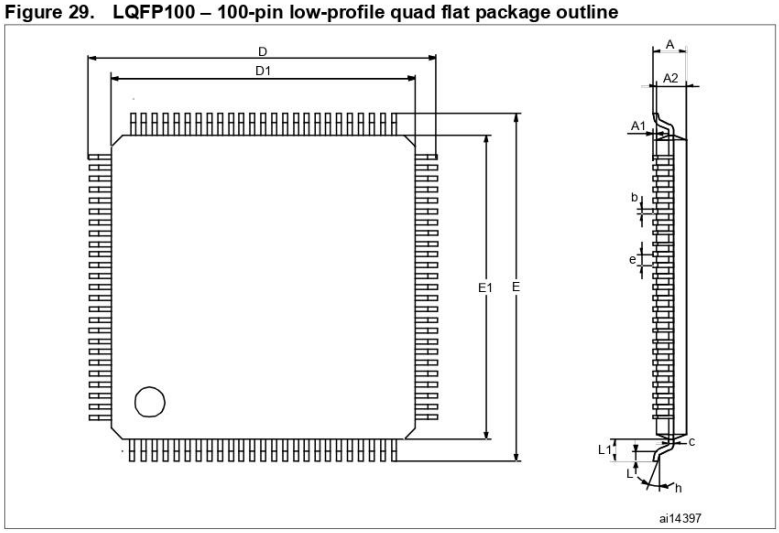


Table 43. LQFP100 – 100-pin low-profile quad flat package mechanical data

Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A			1.60			0.063
A1	0.05		0.15	0.002		0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
b	0.17	0.22	0.27	0.007	0.009	0.011
C	0.09		0.20	0.004		0.008
D		16.00			0.630	
D1		14.00			0.551	
E		16.00			0.630	
E1		14.00			0.551	
e		0.50			0.020	
θ	0°	3.5°	7°	0°	3.5°	7°
L	0.45	0.60	0.75	0.018	0.024	0.030
L1		1.00			0.039	
	Number of pins					
N	100					

Figure 30. LQFP64 – 64 pin low-profile quad flat package outline

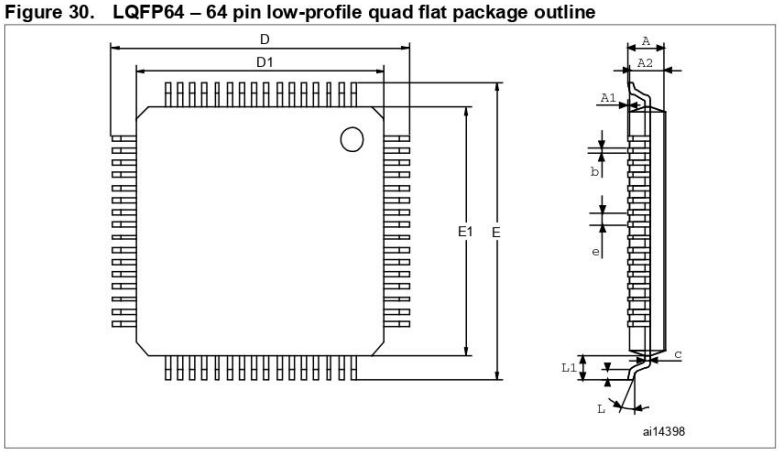


Table 44. LQFP64 – 64 pin low-profile quad flat package mechanical data

Dim.	mm			inches		
	Min	Typ	Max	Min	Typ	Max
A			1.60			0.063
A1	0.05		0.15	0.002		0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
b	0.17	0.22	0.27	0.007	0.009	0.011
c	0.09		0.20	0.004		0.008
D		12.00			0.472	
D1		10.00			0.394	
E		12.00			0.472	
E1		10.00			0.394	
e		0.50			0.020	
θ	0°	3.5°	7°	0°	3.5°	7°
L	0.45	0.60	0.75	0.018	0.024	0.030
L1		1.00			0.039	
	Number of pins					
N	64					

Figure 31. LQFP48 – 48 pin low-profile quad flat package outline

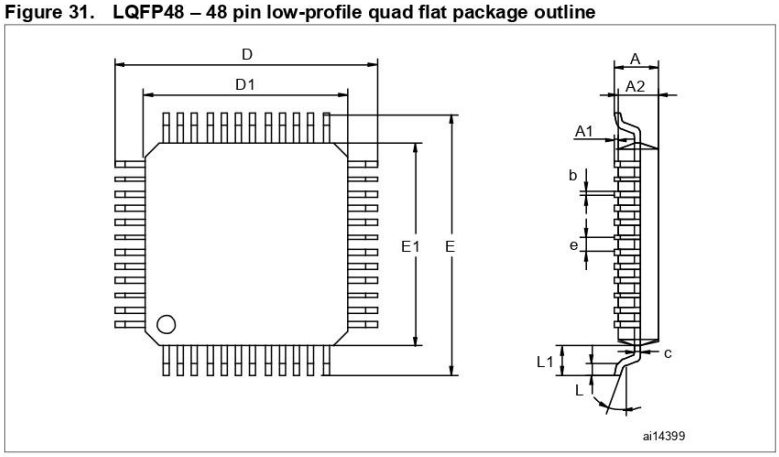


Table 45. LQFP48 – 48 pin low-profile quad flat package mechanical data

Dim.	mm			inches[1]		
	Min	Typ	Max	Min	Typ	Max
A			1.60			0.063
A1	0.05		0.15	0.002		0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
b	0.17	0.22	0.27	0.007	0.009	0.011
C	0.09		0.20	0.004		0.008
D		9.00			0.354	
D1		7.00			0.276	
E		9.00			0.354	
E1		7.00			0.276	
e		0.50			0.020	
θ	0°	3.5°	7°	0°	3.5°	7°
L	0.45	0.60	0.75	0.018	0.024	0.030
L1		1.00			0.039	
	Number of pins					
N	48					

1. Values in inches are converted from mm and rounded to 3 decimal digits.

6.1 Thermal characteristics

The average chip-junction temperature, T_J, in degrees Celsius, may be calculated using the following equation:

$$T_J = T_A + (P_D \times \Theta_{JA}) \quad (1)$$

Where:

- T_A is the Ambient Temperature in °C,
- Θ_{JA} is the Package Junction-to-Ambient Thermal Resistance, in °C/W,
- P_D is the sum of P_{INT} and $P_{I/O}$ ($P_D = P_{INT} + P_{I/O}$),
- P_{INT} is the product of I_{DD} and V_{DD}, expressed in Watts. This is the Chip Internal Power.

$P_{I/O}$ represents the Power Dissipation on Input and Output Pins;

Most of the time for the application $P_{I/O} < P_{INT}$ and can be neglected. On the other hand, $P_{I/O}$ may be significant if the device is configured to drive continuously external modules and/or memories.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is given by:

$$P_D = K / (T_J + 273\ °C) \quad (2)$$

Therefore (solving equations 1 and 2):

$$K = P_D \times (T_A + 273°C) + \Theta_{JA} \times P_D^2 \quad (3)$$

where:

K is a constant for the particular part, which may be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J may be obtained by solving equations (1) and (2) iteratively for any value of T_A.

Table 46. Thermal characteristics

Symbol	Parameter	Value	Unit
Θ_{JA}	**Thermal resistance junction-ambient** LFBGA100 - 10 x 10 mm / 0.5 mm pitch	41	°C/W
	Thermal resistance junction-ambient LQFP100 - 14 x 14 mm / 0.5 mm pitch	46	
	Thermal Resistance Junction-Ambient LQFP64 - 10 x 10 mm / 0.5 mm pitch	45	
	Thermal resistance junction-ambient LQFP48 - 7 x 7 mm / 0.5 mm pitch	55	

7 Order codes

Table 47. Order codes

Part number	Flash program memory Kbytes	SRAM memory Kbytes	Package
STM32F103C6T6	32	10	LQFP48
STM32F103C8T6	64	20	
STM32F103R6T6	32	10	LQFP64
STM32F103R8T6	64	20	
STM32F103RBT6	128	20	
STM32F103V8T6	64	20	LQFP100
STM32F103VBT6	128	20	
STM32F103V8H6	64	20	LFBGA100
STM32F103VBH6	128	20	

7.1 Future family enhancements

Further developments of the STM32F103xx performance line will see an expansion of the current options. Larger packages will soon be available with up to 512KB Flash, 64KB SRAM and with extended features such as EMI support, SDIO, I2S, DAC and additional timers and USARTS.

8 Revision history

Table 48. Document revision history

Date	Revision	Changes
01-jun-2007	1	Initial release.
20-Jul-2007	2	Flash memory size modified in *Note 5*, *Note 4*, *Note 6*, *Note 7* and BGA100 pins added to *Table 3: Pin definitions*. *Figure 5: STM32F103xx performance line BGA100 ballout* added. T_{HSE} changed to T_{LSE} in *Figure 12: Low-speed external clock source AC timing diagram*. V_{BAT} ranged modified in *Power supply schemes*. $t_{SU(LSE)}$ changed to $t_{SU(HSE)}$ in *Table 17: HSE 4-16 MHz oscillator characteristics*. $I_{DD(HSI)}$ max value added to *Table 19: HSI oscillator characteristics*. Sample size modified and machine model removed in *Electrostatic discharge (ESD)*. Number of parts modified and standard reference updated in *Static latch-up*. 25 °C and 85 °C conditions removed and class name modified in *Table 28: Electrical sensitivities*. R_{PU} and R_{PD} min and max values added to *Table 29: I/O static characteristics*. R_{PU} min and max values added to *Table 32: NRST pin characteristics*. *Figure 18: I^2C bus AC waveforms and measurement circuit* and *Figure 17: Recommended NRST pin protection* corrected. Notes removed below *Table 7*, *Table 32*, *Table 37*. I_{DD} typical values changed in *Table 11: Maximum current consumption in Run and Sleep modes*. *Table 33: TIMx characteristics* modified. t_{STAB}, V_{REF+} value, t_{lat} and f_{TRIG} added to *Table 39: ADC characteristics*. In *Table 24: Flash memory endurance and data retention*, typical endurance and data retention for T_A = 85 °C added, data retention for T_A = 25 °C removed. V_{BG} changed to V_{REFINT} in *Table 10: Embedded internal reference voltage*. Document title changed. *Controller area network (CAN)* section modified. *Figure 9: Power supply scheme* modified. *Features on page 1* list optimized. Small text changes.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER'S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com

5. *Encoder* E6A2-CS3E

CHINA YUMO ELECTRIC CO.,LIMITED
YUEQING YING'S IMPORT & EXPORT CO.,LTD
ADD: ZHAOYANG INDUSTRIAL ZONE, LIUSHI TOWN, YUEQING CITY, ZHEJIANG, 325604 CHINA
TEL: 0086-577-62791815 FAX: 0086-577-62791825
Http://www.yumoelectric.com E-mail:sales@yingselectric.com

A6A2 series

Incremental Rotary Encoder

YUMO®

◆**Substitute: E6A2**

●Mini type encoder
●Outside Diameter φ25 Model: A6A2

◆**Features**

Miniature rotary encoder for positioning in space-confined areas
●Wide variety of supply voltages and output forms to match input devices
●Models with zero index function ideal for position applications
●High resolution models（300 or 360 pulses per revolution）substantially improve measuring accuracy
●High response frequency and noise immunity make encoders ideal for factory automation applications

◆Ordering Information

A	6	A	2	-	C	W	Z	6	C
Sequence Number	Outside Diameter φ25			Incremental Rotary Encoder		S: Singleness Output A W: AB90° Phase Difference Z: +Zero signal		1: DC5V 3: DC5~12V 5: DC12~24V 6: DC5~24V	B: PNP Open Collector Output C: NPN Open Collector Output E: Voltage Output GH: NPN Push Pull Output EH: PNP Voltage Output X: Line Driver Output

◆**Note:**
1, Please contact sales representative to confirm the correctness and availability of the part number

◆Technical Specifications

Output Phases	Power Supply Voltage	Output Configuration	Substitution	Resolution (pulses/rotation)
Phase A	5 to 12 VDC	Voltage Output	A6A2-CS3E	10, 20, 60, 100, 200, 300, 360
				500
		Open-collector Output (NPN Output)	A6A2-CS3C	10, 20, 60, 100, 200, 300, 360
				500
	12 to 24 VDC		A6A2-CS5C	10, 20, 60, 100, 200, 300, 360
				500
Phase A and B	5 to 12 VDC	Voltage Output	A6A2-CW3E	100 200 360
				500
		Open-collector Output (NPN Output)	A6A2-CW3C	100 200 360
				500
	12 to 24 VDC		A6A2-CW5C	100 200 360
				500
Phase A, B, and Z	5 to 12 VDC	Voltage Output	A6A2-CWZ3E	100 200 360
				500
		Open-collector Output (NPN Output)	A6A2-CWZ3C	100 200 360
				500
	12 to 24 VDC		A6A2-CWZ5C	100 200 360
				500

CHINA YUMO ELECTRIC CO.,LIMITED
YUEQING YING'S IMPORT & EXPORT CO.,LTD
ADD: ZHAOYANG INDUSTRIAL ZONE, LIUSHI TOWN, YUEQING CITY, ZHEJIANG, 325604 CHINA
TEL: 0086-577-62791815 FAX: 0086-577-62791825
Http://www.yumoelectric.com E-mail:sales@yingselectric.com

◆Ratings and Specifications

Item \ Model	A6A2-CS3E	A6A2-CS3C	A6A2-CS5C	A6A2-CW3E	A6A2-CW3C	A6A2-CW5C	A6A2-CWZ3E	A6A2-CWZ3C	A6A2-CWZ5C
Power supply Voltage	5 VDC-5% to 12 V+10%, ripple (p-p): 5% max.		12 VDC-10% to 24 V+15%, ripple (p-p): 5% max.	5 VDC-5% to 12 V+10%, ripple (p-p): 5% max.		12 VDC-10% to 24 V+15%, ripple (p-p): 5% max.	5 VDC-5% to 12 V+10%, ripple (p-p): 5% max.		12 VDC-10% to 24 V+15%, ripple (p-p): 5% max.
Current Consumption	30mA max	20mA max		30mA max	20mA max			30mA max	
Resolution (See Note1)	10, 20, 60, 100, 200, 300, 360, 500			10, 200, 360, 500					
Output Phases	Phase A			Phase A and B			Phase A, B, and Z		
Output Form	Voltage Output	NPN Open-collector Output		Voltage Output	NPN Open-collector Output		Voltage Output	NPN Open-collector Output	
Output Capacity	Output Resistance: 2KΩ Output Current: 20mA max. Residual Voltage:0.4V max (Output current: 20mA max.)	Applied Voltage: 30VDC max. Sink Current: 30mA max. Residual Voltage: 0.4Vmax (at sink current of 30mA)		Output Resistance: 2KΩ Output Current: 20mA max. Residual Voltage:0.4V max (Output current: 20mA max.)	Applied Voltage: 30VDC max. Sink Current: 30mA max. Residual Voltage: 0.4Vmax (at sink current of 30mA)		Output Resistance: 2KΩ Output Current: 20mA max. Residual Voltage:0.4V max (Output current: 20mA max.)	Applied Voltage: 30VDC max. Sink Current: 30mA max. Residual Voltage: 0.4Vmax (at sink current of 30mA)	
Maximum Response Frequency	30kHZ								
Phase Difference of Output	---			Phase difference between phases A and B: 90°±45°					
Output Waveform Percentage	50±25%			---					
Output Rise And Fall Times	1.0μs max. (Cable length: 2m, Sink current: 10 mA)	1.0μs max. (Cable length: 2m, Control output voltage: 5V, Load resistance 1kΩ)		1.0μs max. (Cable length: 2m, Sink current: 10 mA)	1.0μs max. (Cable length: 2m, Control output voltage: 5V, Load resistance 1kΩ)		1.0μs max. (Cable length: 2m, Sink current: 10 mA)	1.0μs max. (Cable length: 2m, Control output voltage: 5V, Load resistance 1kΩ)	

◆Note:

*1. An inrush current of approximately 9A will flow for approximately 0.3ms when the power is turned ON.
*2. The maximum electrical response revolution is determined by the resolution and maximum response frequency as follows：

Maximum Electrical Response Frequency (Rpm) = Maximum Response Frequency/ Resolution*60

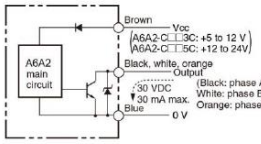
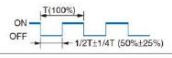
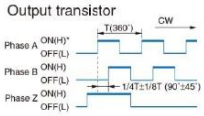
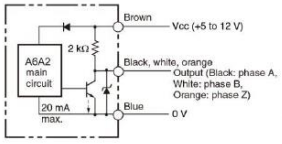
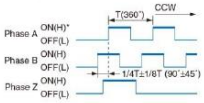
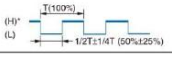
This means that the A6A2 encoder will not operate electrically if its shaft speed exceeds the maximum electrical response revolution
*3. No protection is provided against water or oil.

CHINA YUMO ELECTRIC CO.,LIMITED
YUEQING YING'S IMPORT & EXPORT CO.,LTD
ADD: ZHAOYANG INDUSTRIAL ZONE, LIUSHI TOWN, YUEQING CITY, ZHEJIANG, 325604 CHINA
TEL: 0086-577-62791815 FAX: 0086-577-62791825
Http://www.yumoelectric.com E-mail:sales@yingselectric.com

Mechanical Specifications		
Starting Torque		1mN.m max
Moment Of Inertia		1 x 10^{-7}kg.m^2 max.
Shaft Loading	Radial	10N
	Thrust	50N
Mounting Tolerance		Radial: 0.03mm TIR Max; Axial:0.2mm Max; Shaft Runout: 0.1° Max
Allowable Shaft Load		Radial: 5N; Axial: 3N
Maximum Rotating Speed		5000 rpm
Ambient Temperature Range		Operating: -10 to 55℃ (with no icing), Storage: -25 to 80℃ (with no icing)
Ambient Humidity Range		Operating/storage:35% to 85% (with no condensation)
Insulation Resistance		20 MΩ min. (at 500 VDC) between current-carrying parts and case
Dielectric Strength		500 VAC, 50/60 Hz for 1 min between current-carrying parts and case
Vibration Resistance		Destruction: 10 to 55 Hz, 1.5-mm double amplitude for 2 hours each in X, Y, and Z directions
Shock Resistance		Destruction: 500m/s^2 3 times each in X, Y, and Z directions
Degree Of Protection		IEC 60529 IP50
Connection Method		Pre-wired Models (Standard cable length: 2 m)
Weight		Approx. 60g
Accessories		Coupling, Servo Mounting Bracket (provided with the E6A2-CWZ□), Hexagonal wrench, Instruction manual

---

◆I/O Circuit Diagrams

Model	Output circuits	Output mode	Connection
A6A2-CS3C A6A2-CS5C		Output transistor 	 Note: 1. The white and orange wires of Single Models (A6A2-CS□□) do not output signals (no connection). 2. The white and orange wires of Single Models (A6A2-CS□□) do not output signals (no connection). 3. Voltage Output Models are capable of sinking a maximum current of 20 mA.
A6A2-CW3C A6A2-CW5C		Direction of rotation: CW (as viewed from end of shaft) Output transistor 	
A6A2-CWZ3C A6A2-CWZ5C			
A6A2-CW3E		Direction of rotation: CCW (as viewed from end of shaft) Output transistor 	
A6A2-CWZ3E			
A6A2-CS3E		Output transistor 	

Note: 1. *(H) and (L) indicate the output levels of Voltage Output Models.
2. Output A leads B by 1/4 T±1/8 T when the shaft revolves clockwise, while A lags behind B by 1/4 T±1/8 T when the shaft revolves counterclockwise.

CHINA YUMO ELECTRIC CO.,LIMITED
YUEQING YING'S IMPORT & EXPORT CO.,LTD
ADD: ZHAOYANG INDUSTRIAL ZONE, LIUSHI TOWN, YUEQING CITY, ZHEJIANG, 325604 CHINA
TEL: 0086-577-62791815 FAX: 0086-577-62791825
Http://www.yumoelectric.com E-mail:sales@yingselectric.com

A6A2 series

Incremental Rotary Encoder

YUMO®

◆Dimensions(mm)