Universidad de La Laguna

– TRABAJO DE FIN DE GRADO –

Sistema de localización de robots en interiores

—

Indoor localization system for robots

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA CIVIL E INDUSTRIAL

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Agustín León García

La Laguna, a 9 de septiembre de 2018

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78.698.554-Y profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:
"Sistema de localización de robots en interiores."

ha sido realizada bajo su dirección por D. Agustín León García, con N.I.F. 78.645.734 - V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de septiembre de 2018.

Agradecimientos

Se acaba una etapa.
A todas las personas que han caminado de mi mano durante estos años. Personas que están, que han estado y que estarán siempre a mi lado.

Familia, amigos, compañeros y profesores.
Gracias.

Resumen

Localización y autonomía. El principal objetivo de este proyecto es desarrollar un sistema de localización de robots para interiores y, que gracias a esta sistema en tiempo real, el propio robot pueda moverse de forma autónoma por el entorno. El ultrasonido, la radiofrecuencia y ROS serán los encargados de crear este espacio de trabajo para el robot.

Gracias a un sistema de emisores de ultrasonido comunicados entre sí por radiofrecuencia se creará un ambiente en el que un receptor, con un total de seis módulos de ultrasonido, estará recibiendo en todo momento información de estos emisores. Un algoritmo implementado en el receptor será el encargado de procesar esta información enviada por los emisores y generar unas coordenadas X e Y sobre un plano.

La comunicación del sistema de localización con el robot, que será el que posea el objeto receptor, se ha ejecutado con ROS.

Palabras clave: Arduino, ultrasonido, radiofrecuencia, localización, Raspberry Pi, GopiGo, ROS.

Abstract

Localization and autonomy. The main purpose of this project is to develop an indoor localization system for robots. Helped by this real time system, the robot can move itself autonomously through the environment. Ultrasound, radiofrequency and ROS are responsible to create this workspace for the robot.

Thanks to a system of ultrasound emitters communicated with each other by radiofrequency, an environment will be created, where a receiver with a total of six ultrasound modules, receives information from these emitters throughout. An algorithm implemented in the receiver is the responsible to process this information sent by the emitters and generate some X and Y coordinates on a plane.

The communication of the localization system with our robot, which is the receiver, has been executed with ROS.

Keywords: Arduino, ultrasound, radiofrequency, localization, Raspberry Pi, GopiGo, ROS.

Índice General

Índice de figuras

Capítulo I.

Introducción

En el presente capítulo se hará una pequeña y breve descripción de lo que será el objetivo del proyecto, el desarrollo a seguir y de forma adicional, una pequeña descripción del contenido de la memoria.

1.1 Introducción al proyecto

En este proyecto se tendrá como objetivo principal la siguiente premisa: ¿cómo poder localizar un robot en zonas donde un sistema de localización más común como el GPS no esté disponible? A parte de resolver esta problemática, también se tomará como objetivo secundario que, una vez este robot esté correctamente localizado en un plano, pueda moverse de forma autónoma en el medio hacia una nueva localización.

Un sistemas de localización permite identificar o rastrear automáticamente la localización de objetos o personas en tiempo real. Un sistema de localización GPS, como el que se ha nombrado anteriormente, tiene una precisión bastante buena. De centímetros hasta unos pocos metros, dependiendo del sistema que se utilice. Este método basa su funcionamiento en una red de cómo mínimo 24 satélites en órbita sobre la Tierra a una distancia de 20180 Km. Cuando se quiere obtener una posición, el receptor toma, como mínimo, la localización de 4 satélites, en el caso de una posición tridimensional. En base a estas señales, el dispositivo sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo. De esta manera y gracias al método de trilateración inversa, el cual se basa en determinar la distancia de cada satélite al punto de medición, se consigue medir dicha distancia.

Dicho esto, se tomará como principal influencia el funcionamiento del ya nombrado sistema GPS, pero con algunas modificaciones para superar las dificultades impuestas, tales como precisión, autogestión y autonomía.

El proyecto desarrollado está orientado a casos específicos, como hemos nombrado anteriormente, a aquellos lugares donde no sea posible la

localización por GPS, tales como interiores de casas, oficinas, garajes, etc. En resumen se requerirá que, tanto emisores como receptores, estén continuamente en contacto para poder definir el sistema en tiempo real. En este caso, un sistema de 3 emisores de ultrasonido serán los encargados de generar pulsos de onda para que el receptor/robot sea capaz de recibir estas ondas y así poder localizarse, orientarse y moverse a lo largo y ancho del entorno .

Como se ha comentado, el sistema de localización estará principalmente basado en el uso de sensores de ultrasonido, que serán los encargados de medir las distancias mediante métodos matemáticos, tales como la trilateración. Para facilitar la comunicación entre los emisores y el receptor, se utilizarán diversos componentes electrónicos así como varias infraestructuras. Los ultrasonido irán conectados y codificados en placas Arduino para un fácil manejo e implementación, así como los módulos de radiofrecuencia, que serán los encargados de la comunicación constante.

Una vez se tenga en constante comunicación y envío de información todo el sistema, se implementarán dichos datos vía ROS y odometría para que el emisor/robot pueda moverse de forma autónoma por todo el área de trabajo, el cual es el objetivo final de este proyecto.

Comentar también que este proyecto es una continuación del proyecto presentado en anteriores convocatorias por el alumno Alberto Martínez Chincho y que será una especie de upgrade al trabajo realizado por él, mejorando la comunicación de radiofrecuencia con nuevos módulos más versátiles, mejorando la infraestructura con diseños 3D e implementando el sistema de localización en el robot GopiGo 2.

1.2 Contenido de la Memoria

En este apartado se expone una breve sinopsis del contenido de esta memoria, enumerando cada uno de los capítulos y su contenido.

- Capítulo 1: INTRODUCCIÓN

Descripción de los objetivos del proyecto y del desarrollo a seguir. También se describe a modo de resumen alguno de los componentes utilizados. Se detalla brevemente también los capítulos incluidos en la memoria.

- Capítulo 2: MARCO TEÓRICO

Descripción de los métodos y componentes utilizados para la realización del proyecto, desde métodos de localización hasta una amplia definición de módulos de ultrasonido, radiofrecuencia y Arduino utilizados, así como del robot y su entorno de aplicación. Se explica también el por que de la elección de dichos componentes y modos de uso en el proyecto. Además, se comenta brevemente los entornos de trabajo utilizados e implementaciones necesarias para el uso del robot manera autónoma.

- Capítulo 3: MARCO PRÁCTICO

Descripción de manera detallada los procedimientos llevados a cabo para la realización, paso por paso, de nuestros objetivos. Se describen los montajes realizados, desde las pruebas iniciales hasta el montaje final, comprobando en todo momento el correcto funcionamiento de cada uno de los montajes. Se expone también el desarrollo del montaje final una vez incluido en el robot para su aplicación de forma autónoma en el entorno mediante ROS, obteniendo datos de odometría provenientes del objeto receptor.

- Capítulo 4: CONCLUSIONES Y LINEAS FUTURAS

Balance objetivo del trabajo realizado y objetivos cumplidos así como posibles formas de mejorar el proyecto en una siguiente fase de este.

- Capítulo 5: PRESUPUESTO

Estimación de costes y presupuesto de acuerdo al tiempo de realización y a materiales y componentes empleados en este proyecto.

- Anexo A: CÓDIGOS

Se exponen los códigos utilizados para la implementación de los componentes en Arduino, así como comandos de instalación de paquetes tanto en la Raspberry como en Linux.

- Anexo B: DOCUMENTACIÓN

Se expone un listado con los datasheet de los componentes electrónicos utilizados en el desarrollo hardware del proyecto.

- Anexo C: DISEÑOS 3D

Se exponen los diseños tridimensionales diseñados para la implementación final del trabajo.

- Anexo C: BIBLIOGRAFÍA

Se expone la bibliografía utilizada en el desarrollo del proyecto.

Capítulo II.
Marco Teórico

A lo largo del presente capítulo se intentará desarrollar y explicar con detenimiento, todos y cada uno de los componentes y métodos utilizados durante la ejecución práctica del proyecto, de manera que se puntualizará cuales son las principales características del componente y el porqué se ha elegido este.

Diferentes métodos de localización como la triangulación o la trilateración han sido revisados a lo largo del desarrollo, así como diferentes módulos de radiofrecuencia y ultrasonido.

2.1 Localización

Del latín "*locus*" que indica lugar, la localización hace referencia a una ubicación espacial, comúnmente usado en el campo de la geografía para identificar donde están situados ciudades, países incluso lugares más concretos como puntos, objetos o incluso personas.

Los sistemas de localización en tiempo real (del inglés Real-time locating system o RTLS) son los sistemas utilizados para el proyecto ya que, utilizando como capa física la radio frecuencia simultáneamente con la señal acústica del ultrasonido, identifican automáticamente la localización del objeto en tiempo real dentro de un recinto o área cerrada.

Para este proyecto, la principal problemática a resolver es el tipo de localización que se desea conseguir así como cual será el método de estimación o aproximación para conocer la posición de nuestro objeto, siendo este último, de vital importancia para el cálculo final de la posición sobre el plano de nuestro objeto.

2.1.1 Tipos de localización

Como ha sido citado en el apartado anterior, será de vital importancia saber qué tipo de localización será la adecuada para el proyecto, definiendo los siguientes tipos:

Localización local: basada en una estimación de la posición del objeto teniendo en cuenta la posición inicial. Aquí es donde entra en juego la odometría, definida como el estudio de la estimación de la posición de objeto

durante su navegación. En objetos móviles, como en el caso de este proyecto, se utiliza para estimar su posición relativa a su localización inicial.

Localización global: basada en una estimación de la posición del objeto sin tener en cuenta información previa acerca de este, normalmente porque se desconoce dicha información, a diferencia de la local, en la que se conoce la posición inicial.

Pero, ¿y cómo será la determinación de la posición, los puntos en el plano? Para ello se tomará en cuenta métodos comúnmente utilizados en el ámbito empresarial, basados en la toma de decisiones, pero que serán de gran ayuda para el desarrollo del proyecto. Estos métodos, llamados deterministas y estocásticos, se rigen según el número y el tipo de variables conocidas.

Modelo determinista: Es un modelo matemático basado en que las mismas condiciones iniciales producirán repetidamente las mismas salidas o resultados, no contemplándose la existencia de causas externas que puedan enturbiar el cálculo. Esto es que, basado en datos por ejemplo, de los sensores utilizados, se pueda estimar o predecir una posición futura.

Modelo estocástico o probabilístico: en contraposición al modelo determinista, este modelo se basa en variables tomadas al azar o de forma aleatoria que intervienen en la localización, cuya relación se predice por medio de funciones probabilísticas. Sirven por lo general para realizar grandes series de muestreos donde la información es escasa, quitando tiempo de procesado.

2.1.2 El ultrasonido

Como se describe en este proyecto, a la hora de localizar un objeto o estimar su posición en un plano, se usará el ultrasonido, por lo que resulta de vital importancia conocer un poco acerca de este tipo de onda, así como sus principales características.

Los ultrasonidos son ondas mecánicas, cuya frecuencia ese encuentra por encima del umbral de percepción para el oído humano y de la cual se necesitará tener claro los siguientes conceptos:

- Es una onda mecánica, por lo que solo se propagará a través de un medio material que sea capaz de permitir la transmisión de las vibraciones de sus partículas.
- Es una onda longitudinal por lo que el movimiento de las partículas que transporta será en la misma dirección de propagación de la onda.
- Es una onda esférica, por lo que debe estar claro que su onda se desplazará en las tres dimensiones describiendo esferas radiales tomando como punto de partida su foco emisor.

Como principal característica, se debe tener en cuenta que la velocidad de propagación de la onda dependerá del medio en el que transmita. Para aproximar la velocidad del sonido en el aire, que será donde se desarrolle el proyecto, se podrá utilizar la siguiente ecuación:

$$c = \sqrt{\frac{\gamma RT}{M}}$$

Figura 2.1: Ecuación de la velocidad del sonido

, donde:

- γ es el coeficiente de dilatación adiabática (γ =1,4 para el aire).
- R es la constante universal de los gases (R= 8,314 J/molK).
- T es la temperatura en grados kelvin (T = 293,15 K).
- M es la masa molar del gas (M = 0,029 kg/mol para el aire).

Para el desarrollo del proyecto, se tomará como valor constante de la velocidad del sonido **344 m/s**. Así mismo, se tendrá en cuenta el fenómeno de la reflexión sin el cual el efecto eco sería imposible y la medición de distancias a través de este sistema también. Este fenómeno implica que, cuando una onda choca con una superficie, esta es rebotada al medio del que proviene con un ángulo de reflexión igual al de incidencia.

Una técnica comúnmente conocida y que utiliza este método de localización es el SONAR, que usa la propagación del sonido bajo el agua para navegar, comunicarse o detectar objetos sumergidos y que, a diferencia del RADAR que utiliza ondas electromagnéticas, este sí utiliza ondas mecánicas.

2.1.3 Estimación de la posición

Conocer la posición de nuestro objeto para a posteriori desarrollar el sistema de localización es de vital importancia para el proyecto, tanto para conocer dicha localización, como tener presente y conocer los movimientos de nuestro objeto. Es por esto que cuanto más preciso sea el método de estimación de posición, mejor resultado se obtendrá en operaciones futuras.

Diferentes métodos de estimación de posición están presentes en este ámbito. Únicamente se tendrá que escoger el que más se adecúe a las necesidades y objetivos del proyecto. Tanto el método de triangulación como el de trilateración son los que, aparentemente, se adecúan más al desarrollo del proyecto, por lo que serán estos métodos los analizados para posteriormente, elegir el que mejor y más eficaz resultado provea.

2.1.3.1 Triangulación

La triangulación es un método trigonométrico basado en la determinación de una posición o punto a partir de ángulos y distancias. Se trata de ir resolviendo triángulos hasta dar con las distancias deseadas. A pesar de poder medir distancias con los sensores de ultrasonido implementados, la obtención de los ángulos de dichos triángulos se hará más tediosa, por lo que también significará la implementación de algún otro sensor que proporcione esta información, complicando de manera obvia la obtención de la posición final, siendo esta una de las principales desventajas a simple vista de este método.

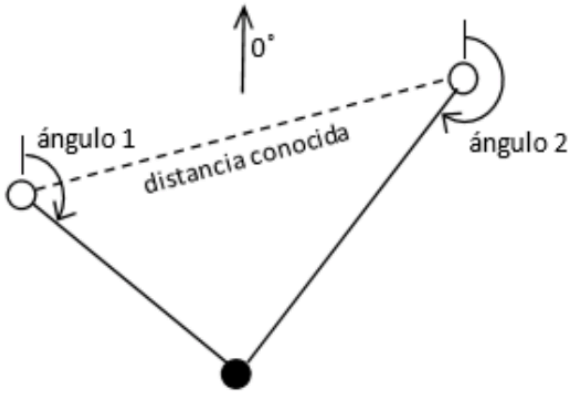


Figura 2.2: Triangulación

2.1.3.2 Trilateración

La trilateración es un método matemático basado en la determinación de una posición o punto a partir de la distancia entre el referente y el objeto. A diferencia de la triangulación, usa las distancias medidas entre el objeto y cada punto de referencia, en este caso, los emisores de onda ultrasónica. La existencia de tres puntos de referencia será condición indudable para la resolución del sistema que proporcione la posición del objeto en un plano. En el caso de necesitar una posición en el espacio, un punto de referencia extra, y por consiguiente, una distancia medida extra, será necesaria, pero no será el caso de este proyecto.

La posición del plano bidimensional vendrá dada después de una serie de cálculos matemáticos conociendo, previamente, la distancia a la que se encuentran los tres emisores o puntos de referencia. Más adelante se explicará en que consisten estos cálculos matemáticos para finalmente poder implementarlos en el objeto receptor.

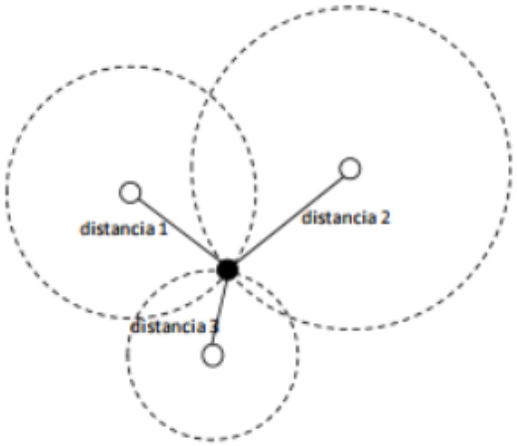


Figura 2.3: Trilateración

2.1.4 Método de localización elegido

Dados los métodos de localización citados en el apartado, se ha elegido el método de trilateración debido a las ventajas que ofrece frente a la triangulación, tales como un coste operacional y computacional menor, así como la facilidad y trivialidad que brinda este sistema con respecto a la propia triangulación. Asimismo, para cálculos de orientación, al tener el objeto receptor un total de seis sensores ultrasónicos, se podrá resolver la problemática de la orientación simplemente haciendo balance de los datos recibidos y haciendo una comparación de estos.

2.1.5 Método de trilateración y sus ecuaciones

Como se ha comentado en apartados anteriores, la trilateración se basa en la determinación de una posición o punto a partir de la distancia entre el referente y el objeto. Para el cálculo de esta posición, el método de la trilateración ofrece una serie de ecuaciones que, en caso de este proyecto en particular, se puede basar en los radios de las circunferencias descritas por los emisores. Estos radios serán básicamente las distancias obtenidas desde el emisor al receptor. También se tomará en cuenta la disposición de los emisores, así como sus coordenadas x e y.

Utilizando la ecuación de la distancia, siendo las distancias las que hay desde los emisores (cuyas coordenadas están predefinidas) hasta el receptor, obtenemos la posición (x, y). Partiendo entonces de las siguientes tres ecuaciones:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$
$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$

Figura 2.4: Ecuaciones de la distancia

Que desarrollando los cuadrados, se obtiene lo siguiente:

$$x^2 + x_1^2 - 2x_1x + y^2 + y_1^2 - 2y_1y = r_1^2$$
$$x^2 + x_2^2 - 2x_2x + y^2 + y_2^2 - 2y_2y = r_2^2$$
$$x^2 + x_3^2 - 2x_3x + y^2 + y_3^2 - 2y_3y = r_3^2$$

Si se resta la segunda ecuación a la primera y la tercera a la segunda, se obtiene:

$$(-2x_1 + 2x_2)x + (-2y_1 + 2y_2)y = r_1^2 - r_2^2 - x_1^2 - x_2^2 - y_1^2 - y_2^2$$
$$(-2x_2 + 2x_3)x + (-2y_2 + 2y_3)y = r_2^2 - r_3^2 - x_2^2 - x_3^2 - y_2^2 - y_3^2$$

Obteniendo el siguiente sistema de ecuaciones de dos incógnitas:

$$Ax + By = C$$
$$Dx + Ey = F$$

La solución de este sistemas de ecuaciones de trilateración proporcionará los valores en x e y de posición del objeto receptor, quedando resuelto como se muestra en la Figura 2.5.

$$x = \frac{CE - FB}{EA - BD}$$

$$y = \frac{CD - AF}{BD - AE}$$

Figura 2.5: Sistema de ecuaciones de la trilateración

A la hora de la programación del objeto receptor como se comentará en apartados posteriores, se deberá implementar dichas ecuaciones para que el programa devuelva explícitamente los valores de x e y, que corresponderán con la posición del objeto.

2.2 Arduino

Arduino es una comunidad de desarrolladores cuya principal función es el diseño y producción de placas de desarrollo de hardware que puedan controlar objetos del mundo real mediante infinidad de sensores que pueden ser conectados a esta placa.

Los diseños de las placas Arduino usan una amplia gama de microcontroladores y microprocesadores, normalmente de la marca Atmel AVR, conectado sobre una placa de circuito impreso a la que se le pueden conectar shields a través de los diferentes puertos de entrada y salida que encontramos en dicha placa. La inmensa mayoría de estas placas son alimentadas por un puerto USB o un puerto barrel Jack de 2.5mm. La programación se realiza a través del puerto Serial que incorporan haciendo uso del Bootloader que traen programado por defecto.

En cuanto al software de la placa, se pueden puntualizar dos características principales: un entorno de desarrollo (IDE), y en el cargador de arranque (Bootloader) que es ejecutado de forma automática una vez la placa recibe alimentación para su funcionamiento. Las placas Arduino se programan mediante el puerto Serial de estas, usando un ordenador, teniendo Arduino su propio software de programación, que será de gran utilidad para el desarrollo del proyecto. Cabe destacar que todas los modelos de placa Arduino se pueden programar y compilar bajo el IDE predeterminado de Arduino gracias a que este compila el código original a la versión de la placa seleccionada en nuestro menú de configuración.

Para el desarrollo del proyecto se ha utilizado un total de cuatro placas Arduino, tres de ella del tipo Arduino UNO para los diferentes emisores y una ultima placa de tipo Arduino MEGA para el receptor, debido a que si se utilizaba otra Arduino UNO, limitaba las funciones de interrupción del receptor debido a que los puertos destinados a estas, no eran suficientes, como señalares más adelante.

2.2.1 Arduino UNO

Figura 2.6: Arduino UNO

La placa Arduino Uno es una placa electrónica basada en el chip de Atmel ATmega328. Tiene 14 pines digitales de entrada/salida, de los cuales seis pueden ser utilizados como salidas PWM y otras 6 entradas analógicas. Tiene también dos pines donde conectar los sensores y actuadores que necesiten de señales digitales que pueden ser utilizados para las interrupciones de la placa. Es por esto que para el receptor necesitamos utilizar una placa con seis pines para las interrupciones, uno por cada sensor de ultrasonido incorporado a dicho receptor. Posee también un oscilador de

cristal de 16 MHz, una conexión USB, un conector de alimentación, una cabecera ICSP y un botón de reset. Algunas de las características principales son las siguientes:

- Microcontrolador ATmega328 (5V)
- 7-12 V de alimentación recomendada
- 14 Pines digitales I/O
- 6 Pines PWM
- 6 entrada analógicas
- 40 mA de corriente máxima por pin
- 32 KB de memoria Flash (0.5 KB para el bootloader)
- 2 KB de SRAM
- 1 KB de EEPROM
- 16 MHz de Velocidad de oscilación

2.2.2 Arduino MEGA

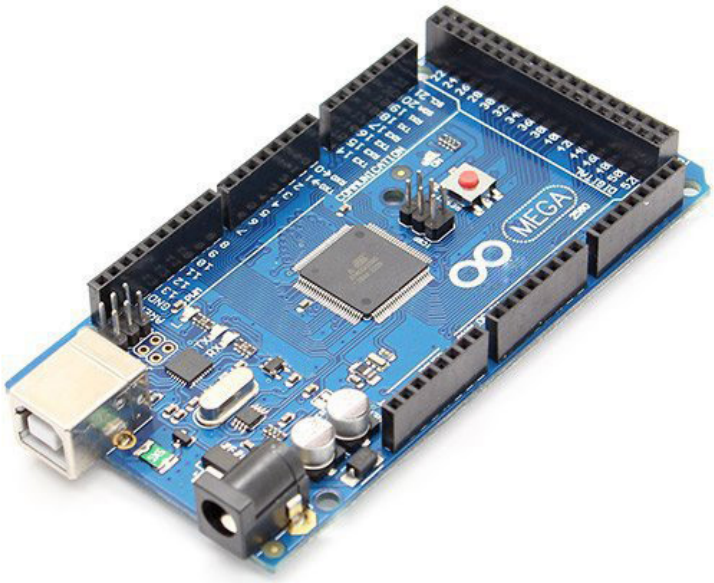


Figura 2.7: Arduino MEGA

El Arduino Mega es una placa electrónica basada en un microcontrolador modelo ATmega2560. Posee 54 pines digitales que funcionan como entrada/salida, 14 de los cuales pueden ser utilizadas como salidas PWM, 16 entradas análogas, 4 UARTs (puertos serial por hardware), un cristal oscilador de 16 MHz, una conexión USB, un botón de reset y una entrada Jack de alimentación de la placa. Además esta placa cuenta con seis pines donde conectar los sensores y actuadores que necesiten de señales digitales que pueden ser utilizados para las interrupciones de la placa, razón principal por la que se ha elegido dicha placa para el receptor, uno por cada sensor de ultrasonido incorporado a dicho receptor y sus correspondientes interrupciones. Algunas de las características principales son las siguientes:

- Microcontrolador ATmega2560
- 7-12 V de alimentación recomendada
- 54 pines digitales I/O
- 14 Pines PWM
- 16 entradas análogicas
- 40 mA de corriente máxima por pin
- 50 mA de corriente máxima en el pin 3.3V
- 256KB de memoria Flash (8 KB para el bootloader)
- 8 KB de SRAM
- 4 KB de EEPROM
- 16 MHz de Velocidad de oscilación

2.3 Sensor de ultrasonido HC-SR04

A la hora de medir distancias con una base de Arduino, es amplio el rango de opciones que existen en el mercado, desde infrarrojos, láser o ultrasonido. Debido al bajo coste y la extremada sencillez de uso e implementación, se ha optado para este proyecto el uso de sensores ultrasónicos, más concretamente en el HC-SR04.

Figura 2.8: Módulo HC-SR04

Su funcionamiento se resume básicamente en el envío de un pulso de alta frecuencia, tal frecuencia que ni siquiera es audible por el ser humano. Este pulso rebota en los objetos cercanos y es reflejado hacia el sensor, que dispone de un micrófono adecuado para esa frecuencia. Una vez recibido este pulso de vuelta, midiendo el tiempo entre pulsos y conociendo la velocidad del sonido, podemos estimar la distancia del objeto en el cual ha rebotado dicha onda de ultrasonido.

El rango de medición teórico del sensor escogido es de 2cm a 400 cm aunque en la práctica, el rango es ligeramente menor, oscilando entre los 20cm y los 2 metros.

2.3.1 Funcionamiento del sensor

Como se ha comentado en el apartado anterior, el sensor funciona emitiendo impulsos de ultrasonidos imperceptibles para el ser humado. Los impulsos emitidos viajan a la velocidad del sonido hasta alcanzar un objeto, entonces el sonido rebota y es recibido de nuevo por el receptor de ultrasonidos.

El módulo de ultrasonido incorpora un controlador que permite a este enviar una ráfaga de impulsos para, posteriormente, contar el tiempo que tarda en llegar el Echo. Este tiempo se traduce en un pulso de Echo de anchura proporcional a la distancia a la que se encuentra el objeto.

Básicamente, lo que se hace es proporcionar un pulso corto en el pin de Trigger para iniciar la medición y posteriormente, una ráfaga de ocho pulsos ultrasónicos a 40kHz es enviada. Momento en el que el pin de Echo se pone a 1 (estado HIGH) como se puede observar en la Figura 2.9. Cuando se detecta otro pulso en el pin de Trigger, lo que significa que la onda ha sido recibida de vuelta, el pin de Echo volverá a 0 (estado LOW). La anchura de este pulso Echo será equivalente a la distancia a la que se encuentra el objeto en el que ha rebotado la onda. En el caso de que no se encuentre ningún objeto en el rango de alcance y por lo tanto el pin de Echo no vuelva a 0, este pulso tendrá una longitud aproximada de 36 ms. Un retarde de 10 ms desde

que se hace la lectura hasta que se realiza la siguiente será recomendable para un correcto funcionamiento del sensor y conseguir un circuito estable.

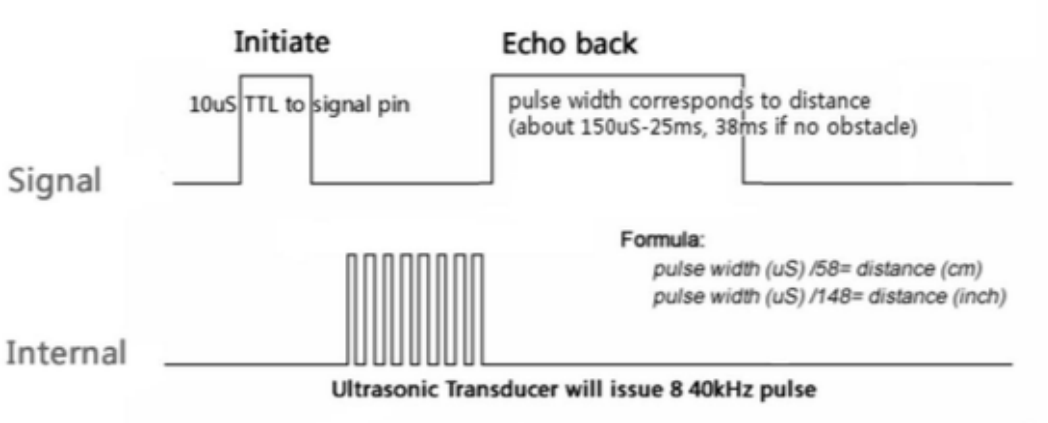


Figura 2.9: Diagrama de pulso del módulo HC-SR04

En el caso especifico para el proyecto a realizar, los pines de conexión para el módulo de ultrasonido serán los tres: el pin de VCC que se conectará al VCC de la placa Arduino, generando 5V, el pin de tierra que de igual forma irá conectado a un puerto GND de la placa y el pin TRIG, que irá conectado al pin número 6 correspondiente a un pin Digital I/O de nuestro Arduino. Puesto que en nuestro objetos emisores no se necesita una señal de Echo, que solamente recibirá el objeto receptor, este pin no será necesario conectarlo.

2.3.2 Cálculo de la distancia

Basándose en las leyes de la cinemática y el movimiento rectilíneo uniforme, la distancia recorrida por el objeto, siendo su posición inicial $x_0 = 0$ viene dada por la función:

$$d = v \cdot t$$

Figura 2.10: Ecuación del M.R.U

, donde d es la distancia recorrida, v es la velocidad, en este caso del sonido y t es el tiempo transcurrido, tomando esta variación desde el envío del pulso hasta su recepción.

Si se admite que la velocidad del sonido es constante y por tanto, tomando el movimiento del sonido rectilíneo uniforme, a una velocidad de 340 m/s, la distancia será el resultado de multiplicar dicha velocidad por el tiempo en que se tarde en enviar y recibir el pulso. No obstante, se debe de tener en cuanta que haciendo esta operación, el resultado final sería la distancia de emisión/recepción, lo que significa que para obtener directamente la distancia al objeto, habría que variar esta fórmula de la siguiente manera:

$$d = \frac{340\ m/s\ \cdot\ t}{2}$$

Figura 2.11: Ecuación de la distancia para el módulo HC-SR04

2.3.3 Características del módulo de ultrasonido

Como cualquier componente electrónico que se desee usar en este proyecto, viene acompañado de una hoja de datos donde se podrá encontrar las características de dicho módulo. En los siguientes apartados se hará un

breve resumen de las que se consideran características a tener en cuenta a la hora de realizar tanto el proyecto físico como sus cálculos.

- Tensión de alimentación: 5 Vcc
- Frecuencia de trabajo: 40 KHz
- Rango máximo: 4 m
- Rango mínimo: 2 cm
- Apertura del pulso ultrasónico: 15º
- Duración mínima del pulso de disparo (nivel TTL): 10 μS.
- Duración del pulso eco de salida (nivel TTL): 100-25000 μS.
- Tiempo mínimo de espera entre una medida y el inicio de otra 20 mS.
- Dimensiones del circuito: 43 x 20 x 17 mm

En términos de ángulo o rango efectivo, este sensor presenta una gráfica similar a la de un patrón de radiación. La propiedad más relevante en este tipo de gráfica es la distribución de energía, es decir cuántos dB tenemos en función de un punto de referencia a lo largo de un radio que no cambia. Para este caso se comenta a qué distancia se puede detectar un objeto en función del ángulo.

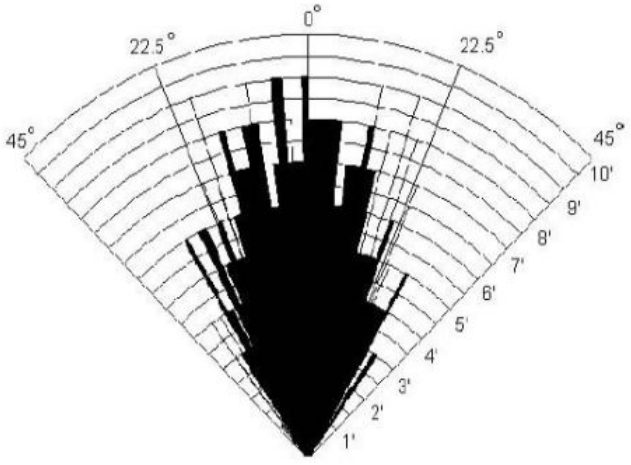


Figura 2.12: Diagrama de ángulo efectivo

El ángulo efectivo para este sensor es de 30°, aunque cabe destacar que, como bien indica el gráfico, el sensor podrá detectar objetos a un rango de aproximadamente 60°, aunque no con la misma precisión que dentro de estos 30º efectivos como muestra la Figura 2.12.

Otra de las características interesantes a tener en cuenta es el ángulo muerto del haz de ultrasonido. Estos módulos tienen una zona muerta, como se muestra en el diagrama de la Figura 2.13, en la cual no pueden obtener con total precisión información acerca del objeto a detectar. Esta es la distancia entre la membrana del sensor y el mínimo rango de sensibilidad. En casos de extrema cercanía, la señal ultrasónica puede rebotar en el objeto antes de que esta haya dejado el sensor, por lo que no sería información válida para nuestro sensor.

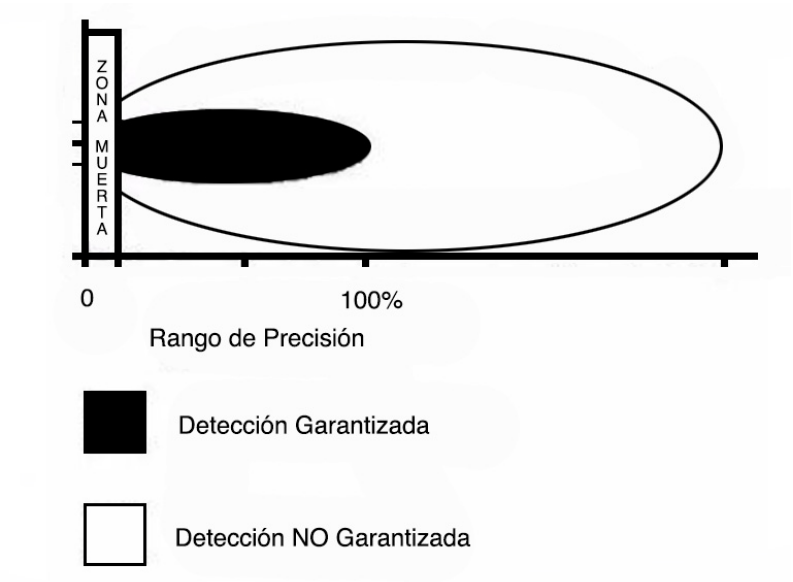


Figura 2.13: Diagrama de precisión y zona muerta

2.3.4 Inclinación del haz de ultrasonido

En cuanto a la inclinación del haz de ultrasonido se debe destacar también ciertos aspectos. Si un objeto es inclinado más de 3º con respecto a la normal al eje del haz de emisión, la señal de ultrasonido es desviada del sensor y por lo tanto, la distancia de detección disminuye, siendo aún permisible esta inclinación. Sin embargo, para objetos pequeños situados cerca del sensor, la desviación respecto a la normal puede aumentar hasta un valor no superior de los 10º. Cuanto más cerca tengamos el objeto, obviamente, mayor inclinación podremos ejercer, asumiendo siempre errores de funcionamiento.

Con la ayuda de un potenciómetro, es posible ajustar la intensidad de onda, para así tener un haz más o menos ancho, aunque se debe tener en cuenta que esta modificación alterará otras características, como la distancia de detección o la zona muerta. Asimismo, el encuentro del haz con una superficie inclinada o no regular, también supondrá una señal de Echo ligeramente menor.

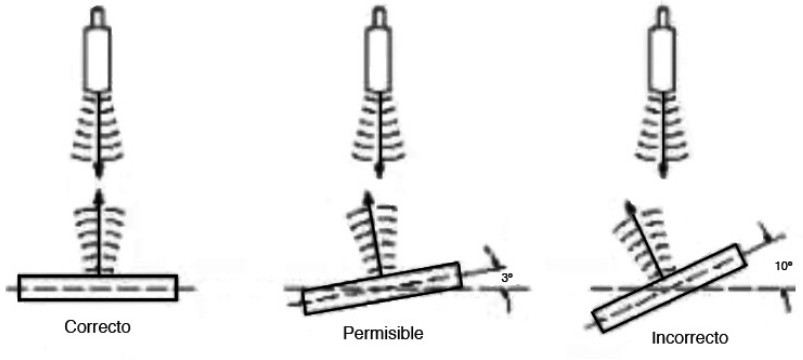


Figura 2.14: Permisividad de ángulo de inclinación

2.4 Módulos de radiofrecuencia

Para facilitar la comunicación entre cada uno de los emisores así como de los emisores con el receptor, se ha dotado a cada uno de los objetos de un módulo de radiofrecuencia NRF24L01. A diferencia del módulo RF433MHz, este posee la característica de integrar tanto el transmisor como el receptor en un mismo módulo, a una frecuencia entre 2.4GHz. La velocidad de transmisión es configurable entre 250 Kbps, 1Mbps, y 2 Mbps y permite la conexión simultánea con hasta 6 dispositivos.

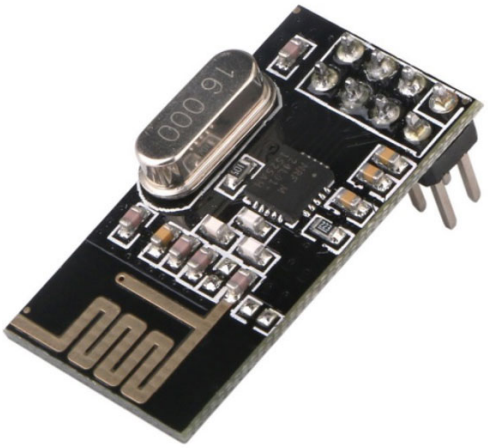


Figura 2.15: Módulo NRF24L01

Otra característica a destacar del NRF24L01 es que también incorpora la lógica necesaria para una comunicación robusta, por lo que el problema de ruido que se obtenía con el RF433MHz queda parcialmente solucionado. El control del módulo se realiza a través de bus SPI, por lo que es sencillo controlarlo desde un procesador como Arduino.

2.4.1 Características del módulo de radiofrecuencia

La tensión de alimentación del NRF24L01 es de 1.9 a 3.6V, por lo que se usará el pin de 3,3V de Arduino para dar alimentación al módulo. El consumo eléctrico en standby es bajo, y de unos 15mA durante el envío y recepción por lo que no se prestará demasiada atención a este apartado, quedando así resuelto. Cabe destacar que, a pesar de usar el módulo básico para el proyecto, con antena integrada y un alcance de 30 metros aproximadamente, existe también un módulo de alta potencia que incorpora amplificador y antena externa, con un alcance máximo de 700-1000 metros. Esto es teóricamente según indica su fabricante. Desgraciadamente, en la práctica, el alcance real se verá limitado por muchos factores. El principal factor será la alimentación, aunque se podrá conseguir un buen alcance alimentando el módulo con los 3,3V descritos anteriormente, siempre y cuando sea una fuente estable y con potencia suficiente.

La principal razón por la que se han elegido estos módulos son su bajo precio, buenas características y fácil implementación en Arduino. Solamente se necesitará añadir una nueva librería llamada “RF24-master”.

2.4.2 Funciones del módulo RF

A continuación se enumeran algunas de las funciones más utilizadas para el envío y recepción de datos durante la realización del proyecto:

RF24 radio (pinCE, pinCSN) → Crea una nueva instancia de este dispositivo. Hará falta también especificar los pines de control que están conectados al módulo.

radio.begin ( ) → Inicializa el objeto creado, generalmente se llama a esta función en setup ( ).

radio.openWritingPipe (pipe) → Abre un canal de comunicación de escritura. Hará falta también especifica la dirección del canal. Su único parámetro será la dirección del canal para abrir.

radio.openReadingPipe (1, pipe) → Abre un canal de comunicación de lectura. Hará falta también especificar la dirección del canal. Sus parámetros serán el número de canal a abrir y su dirección.

radio.startListening ( ) → Se empieza a escuchar por los canales definidos como lectura. Después de llamar a esta función no podemos hacer escrituras, por lo que debemos hacer uso de la función radio.stopListening ( ).

radio.available ( ) → Comprueba si hay bytes disponibles para ser leídos. Esta función devolverá TRUE si existen datos disponibles en el canal de lectura y FALSE si no hay ningún dato recibido.

radio.read (mensaje, sizeof(mensaje)) → Lee un dato por el canal definido. Sus parámetros serán el dato a enviar y el número de bytes a enviar.

radio.write (mensaje, sizeof(mensaje)) → Envía un dato por el canal definido. Sus parámetros serán el dato a enviar y el número de bytes a enviar.

2.5 Placa Raspberry Pi3 Model B

Raspberry Pi es un ordenador de placa simple (SBC) de bajo coste desarrollado en el Reino Unido por la Fundación Raspberry Pi, cuyo principal objetivo es el de apoyar y amenizar la enseñanza de materias informáticas en colegios y universidades. Para este proyecto, y con la principal intención de conseguir una simulación en tiempo real por medio de la estructura para el desarrollo de software para robots ROS, se implementará

el uso de la Raspberry Pi3 Model B basado en el último sistema operativo desarrollado expresamente para este fin: Raspbian Stretch.

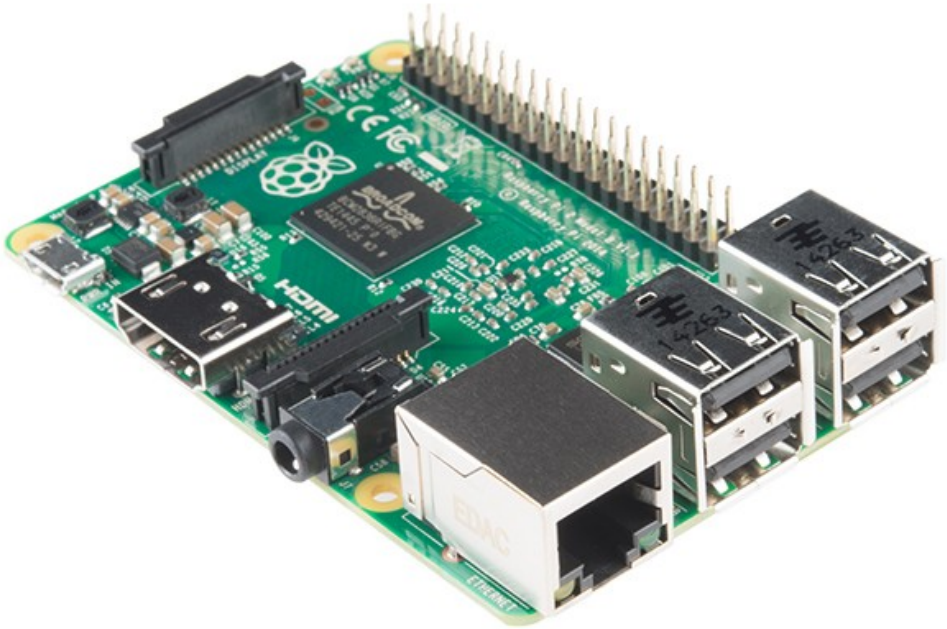


Figura 2.16: Raspberry Pi3 Model B

2.5.1 Características de la placa

La características de este pequeño ordenador son las siguientes:

- Procesador:
- Chipset Broadcom BCM2387.
- 1,2 GHz de cuatro núcleos ARM Cortex-A53
- GPU
- Dual Core VideoCore IV ® Multimedia Co-procesador. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware, y 1080p30 H.264 de alto perfil de decodificación.

- Capaz de 1 Gpixel / s, 1.5Gtexel / s o 24 GFLOPs con el filtrado de texturas y la infraestructura DMA

• RAM: 1GB LPDDR2.

• Conectividad

- Ethernet socket Ethernet 10/100 BaseT

- 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE)

• Salida de vídeo

- HDMI rev 1.3 y 1.4

- RCA compuesto (PAL y NTSC)

• Salida de audio

- jack de 3,5 mm de salida de audio, HDMI

• USB 4 x Conector USB 2.0

• Conector GPIO

- 40-clavijas de 2,54 mm (100 milésimas de pulgada) de expansión: 2x20 tira

- Proporcionar 27 pines GPIO, así como 3,3 V, +5 V y GND líneas de suministro

• Conector de la cámara de 15 pines cámara MIPI interfaz en serie (CSI-2)

• Pantalla de visualización Conector de la interfaz de serie (DSI) Conector de 15 vías plana flex cable con dos carriles de datos y un carril de reloj

• Ranura de tarjeta de memoria Empuje / tire MicroSD

2.6 Dexter Industries GoPiGo

Una vez finalizado el sistema de localización, la idea es implementar este sobre un robot móvil y que se pueda localizar y mover alrededor del plano de trabajo de manera autónoma. Para ello se ha elegido el robot educativo GoPiGo, desarrollado por la compañía Dexter Industries. Se trata de un robot en forma de "coche" que proporcionará una plataforma móvil, donde dos ruedas motorizadas y una tercera rueda loca de apoyo serán las que den soporte al robot. Sobre esta plataforma se posiciona el resto de elementos del robot.

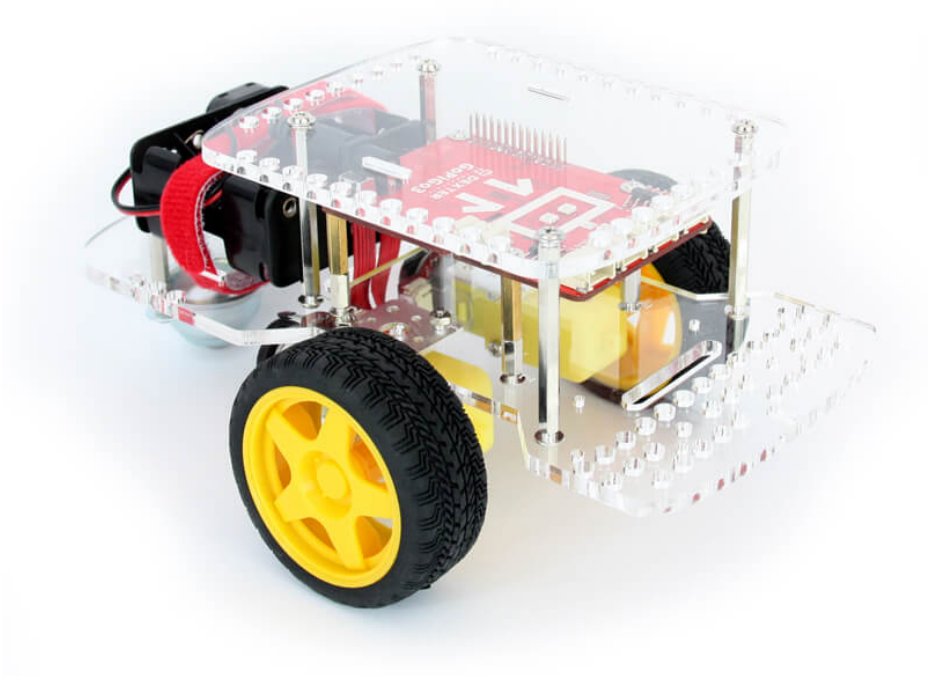


Figura 2.16: Dexter Industries GopiGo 2

Dicho robot estará formado por dos placas, una de propia manufactura por parte de Dexter Industries, la cual hará de interfaz entre los actuadores y la segunda placa, que será la anteriormente nombrada Raspberry Pi3.

La interfaz de programación de aplicaciones (API), en diferentes lenguajes de programación tales como Python o Scratch, facilitará el desarrollo de programación que se necesite en el robot.

2.6.1 Características técnicas del robot

Estas son algunas de las características de dicho robot:

- Un servomotor
- Dos motores donde se acoplaran las ruedas del robot y permitirán el movimiento de este en ambos sentidos.
- Dos encoders: Los cuales se pueden acoplar fácilmente a los motores del robot. Gracias a ellos será posible realizar los cálculos de odometría, los cuales se explicarán en mayor detalle más adelante, para poder así calcular la posición del robot.
- Una tercera rueda loca de apoyo.

2.6.2 Especificaciones de la placa

Estas son algunas de las especificaciones de dicha placa:

- Un puerto analógico la placa dispone de un puerto analógico el cual permite leer valores de voltaje entre 0v y 5v.
- Un puerto serie que puede ser utilizado para conectar diferentes tipos de sensores.
- Un puerto digital/PWM: utilizado para leer o escribir datos digitales.
- Un puerto I2C

• Dos encoders ópticos: la placa contiene dos encoders ópticos para medir la rotación de cada una de las ruedas con una precisión de 18 pulsos de cuenta por cada rotación.

• Dos conectores, uno para cada motor de las ruedas.

• Un conector para un motor servo

• Dos conectores para poder conectar dos sensores LED.

• Microcontrolador Atmega 328: todo el control de los motores y escritura y lectura de los sensores es realizada por este microcontrolador. El microcontrolador actúa de intérprete entre la Raspberry y el GoPiGo.

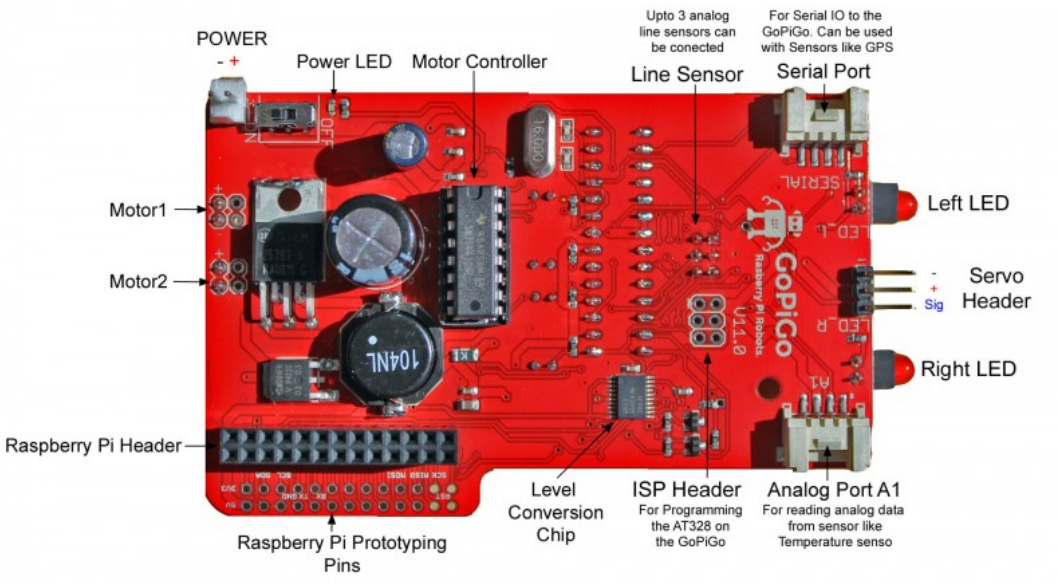


Figura 2.17: Placa GopiGo

2.7 ROS (Robot Operating System)

ROS, del inglés Robot Operating System, es un framework que permite la interactuación de entre el usuario y el software para utilizar y programar

robots. ROS tiene soporte para gran variedad de robots, entre los cuales se encuentra el GopiGo2, que será el utilizado en este proyecto.

ROS utiliza un sistema basado en nodos, mensajes y topics. Los nodos son unidades de computación que ejecutan una tarea simple. Un nodo puede enviar o recibir mensajes. Para ello, un nodo debe suscribirse o publicar en un topic el cual es un canal de comunicación entre los nodos. Un nodo publicará mensajes en un topic y otros nodos se suscribirán a ese topic si quieren recibir los mensajes.

Para obtener más información acerca de los topics y mensajes se puede utilizar el comando rostopic. Esta instrucción muestra información acerca de los topics. Cuenta con diferentes opciones como:

- *rostopic echo*: muestra los datos publicados sobre un tema. Se debe ejecutar este comando por ejemplo si se quiere visualizar los datos provenientes de un topic.
- *rostopic list:* devuelve una lista de todos los Topics que se están utilizando en ese momento.
- *rostopic pub:* publica datos en el topic que se especifica. Por ejemplo

En nuestro proyecto utilizaremos rosserial, que proporciona un protocolo de comunicaciones en ROS que permite trabajar con Arduino. Mediante este paquete podemos usar ROS con el IDE de Arduino. Permite al Arduino crear un nodo, el cual pueda publicar o subscribir mensajes ROS, transformaciones TF y obtener parámetros del sistema ROS.

Para instalar la librería de ros_lib en Arduino, bastará con copiar las librerías descargadas desde en enlace Github de la librería, e incluirlo dentro de la carpeta libraries de Arduino en el ordenador en el que se desarrolle el

código de Arduino, de tal modo que se verá reflejado como muestra la Figura 2.18.

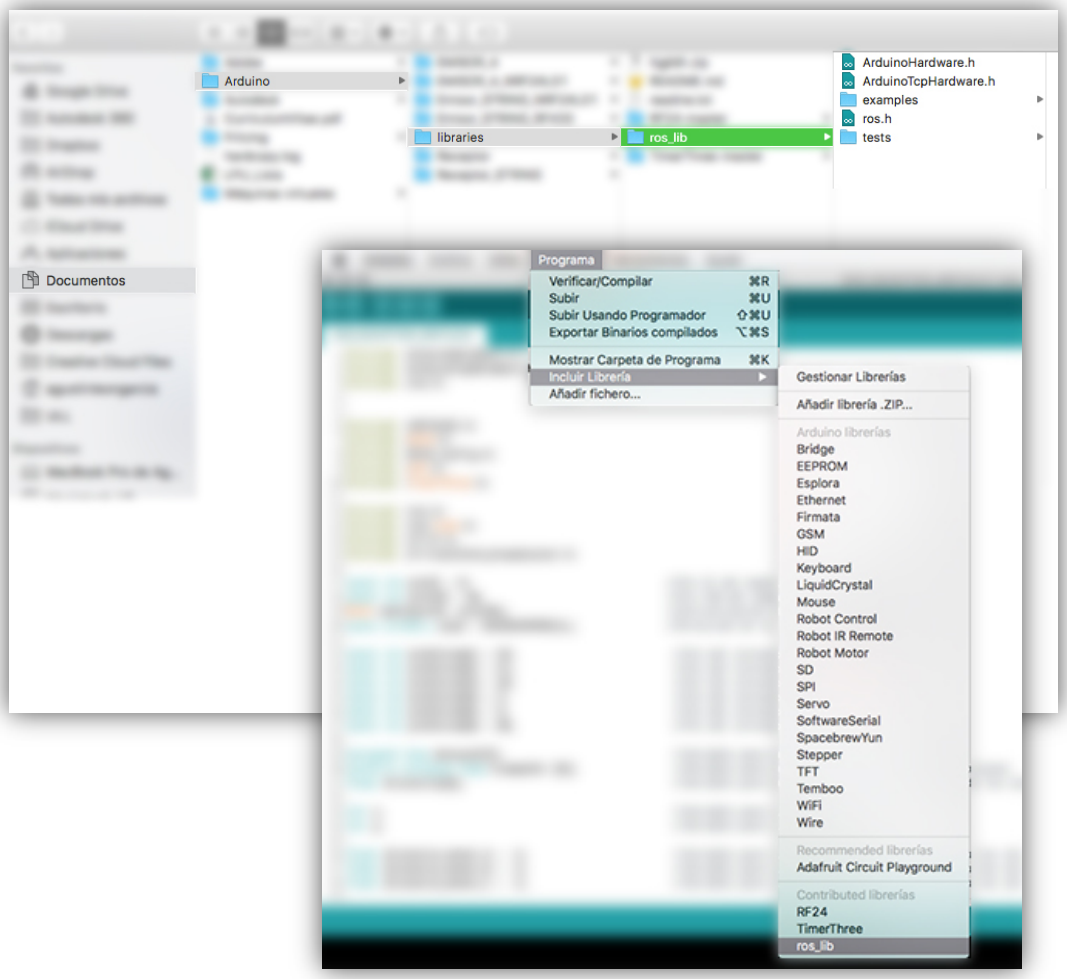


Figura 2.18: Ubicación de librería ros_lib

Para la representación de la odometría y posterior control del robot sobre estos datos utilizaremos ROS Kinetic, una versión de ROS que fue lanzada hace poco más de un año, primeramente para Ubuntu Xenial, aunque posteriormente se ha buscado compatibilidad con otros sistemas operativos con distinto grado de soporte. El entorno de trabajo sobre el que se trabajará será *catkin_ws*, predeterminado por ROS. Lo primero es crear un directorio con ese nombre y a su vez uno dentro del mismo que se debe de llamar src.

Una vez creados los directorios, entramos en el src y ejecutamos el comando que inicializa el workspace. A partir de este momento, se podrá empezar a crear paquetes dentro del directorio src y compilar desde el nivel superior con el comando catkin_make, que se deberá ejecutar cada vez que se desee compilar uno de estos paquetes.

Capítulo III.
Marco Práctico

En el siguiente capítulo se describirá, paso a paso, el desarrollo práctico del proyecto. Se comentará detalladamente desde las primeras implementaciones de los módulos RF y ultrasonido hasta el montaje final, incluido métodos y cálculos realizados para la realización de cada uno de los montajes.

3.1 Desarrollo del proyecto. Hardware

Como bien se ha comentado a lo largo de la redacción de esta memoria, la principal función de este proyecto será dotar de localización a un objeto receptor colocado en cualquier punto de un plano horizontal, ayudado de tres objetos emisores, que serán los encargados de proporcionar señales ultrasónicas con las que se llevará a cabo esta funcionalidad. Una vez resuelto este primer problema que se presenta, se implementará este objeto receptor en un robot Gopigo que, mediante odometría, será capaz de localizarse por sí mismo para que, más adelante y de forma autónoma, pueda moverse por este plano de trabajo hasta unas coordenadas impuestas.

3.1.1 Primer montaje. Objeto emisor

En este primer montaje, el principal objetivo a alcanzar es el de conocer y dominar el funcionamiento del ultrasonido. Para ello, se ha conectado un módulo de ultrasonido a la placa Arduino y, mediante la programación de un código específico, se ha dotado de señal ultrasónica al sensor. Ha sido una prueba bastante trivial pero de suma importancia para comprobar el correcto funcionamiento de estos módulos.

En la Figura 3.1 se pueden observar ambas señales en el osciloscopio, tanto la de TRIGGER como la de Echo, tomadas directamente de dichos pines del ultrasonido.

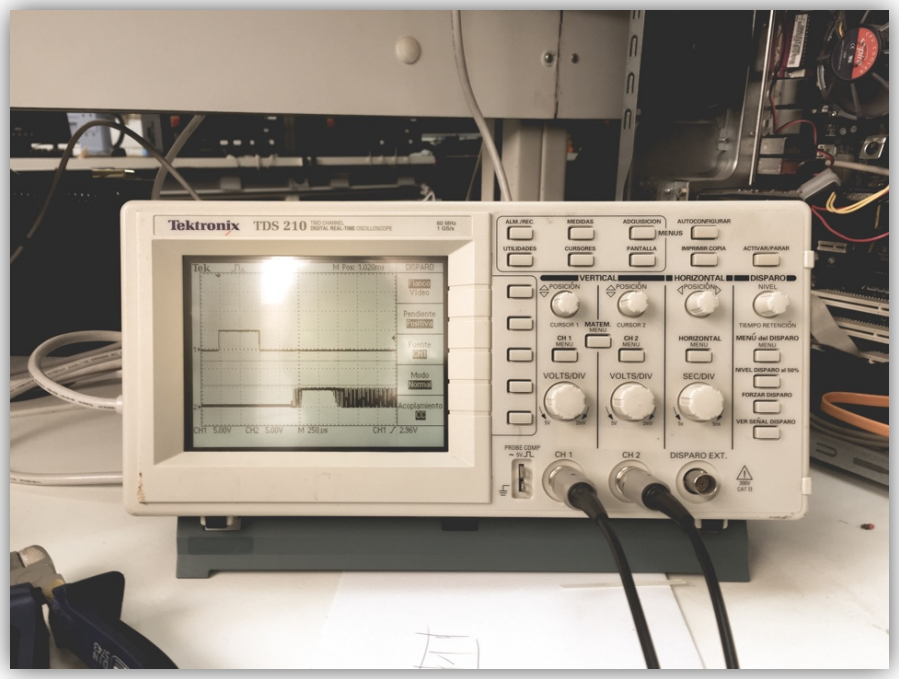


Figura 3.1: Imagen Channel 1 & 2 del primer montaje

3.1.2 Segundo montaje. Objeto emisor + configuración de RF

Una vez comprobado el correcto funcionamiento de los módulos de ultrasonido, se ha pasado a comprobar los de radiofrecuencia. Como se ha descrito anteriormente, son módulos transceptores por lo que no se necesitará dos módulos como en desarrollos anteriores, sino que con un módulo será suficiente para la emisión y recepción de información.

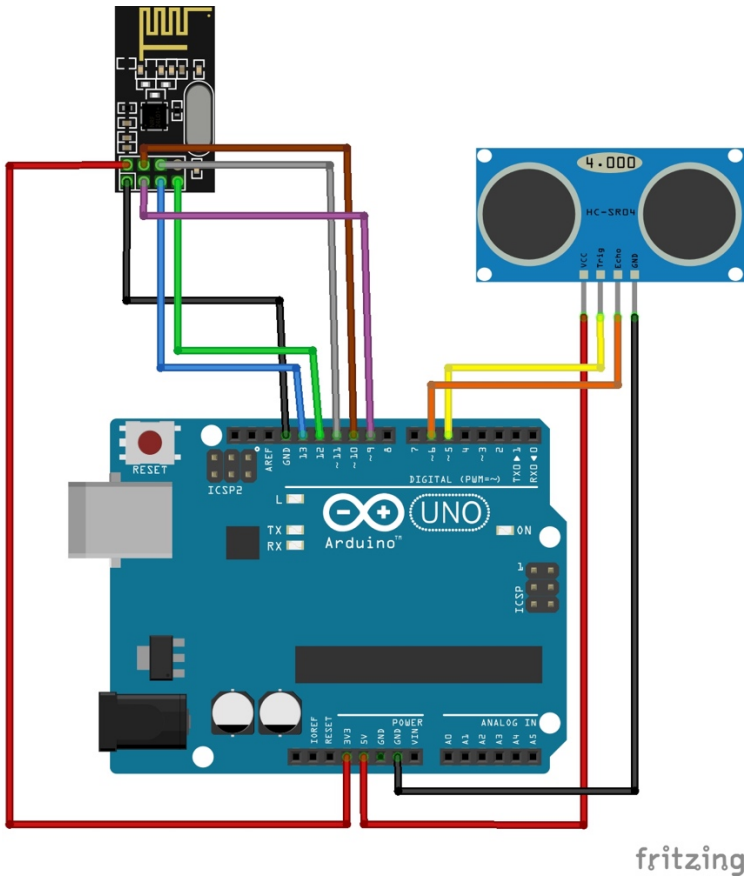


Figura 3.2: Segundo montaje (Objeto emisor + configuración de RF)

En este montaje, y puesto que aún no se tiene el circuito de emisores totalmente montado, lo que se ha hecho ha sido básicamente generar un mensaje y enviarlo una vez se genere un pulso de ultrasonido. Esto servirá más adelante para continuar la comunicación, puesto que desde que el primer emisor envíe el mensaje, el segundo emisor recibirá este aviso y hará lo propio con su mensaje. Y así sucesivamente.

Cabe recordar que, para la correcta implementación de los módulos dentro del código, se ha añadido la librería RF24-master, que permitirá el uso de funciones especificas del módulo transceptor, algunas de las cuales han sido descritas en capítulos anteriores.

Para comprobar que se escribe bien el mensaje y está preparado para ser mandado, se ha creado un código que imprima por pantalla serial el mensaje enviado después de cada pulso, como muestra la Figura 3.3, por lo que, aunque no se tenga el circuito cerrado, se verá claramente que el módulo está actuando de manera correcta.

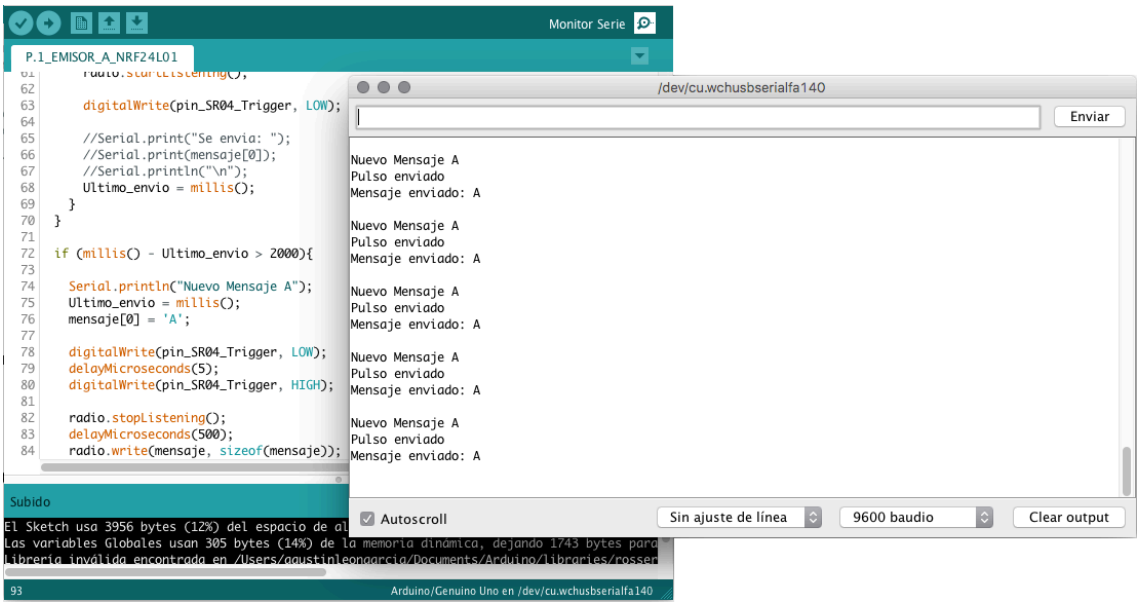


Figura 3.3: Mensaje de pulso enviado

3.1.3 Tercer montaje. Emisor A + Emisor B

En este montaje, y una vez comprobado que el primer emisor escribe correctamente el mensaje deseado, se ha añadido un segundo emisor al circuito para comprobar que la comunicación de radiofrecuencia se realiza correctamente entre más de un objeto emisor.

Para ello, un segundo objeto emisor ha sido montado con exactamente las mismas condiciones que el primero.

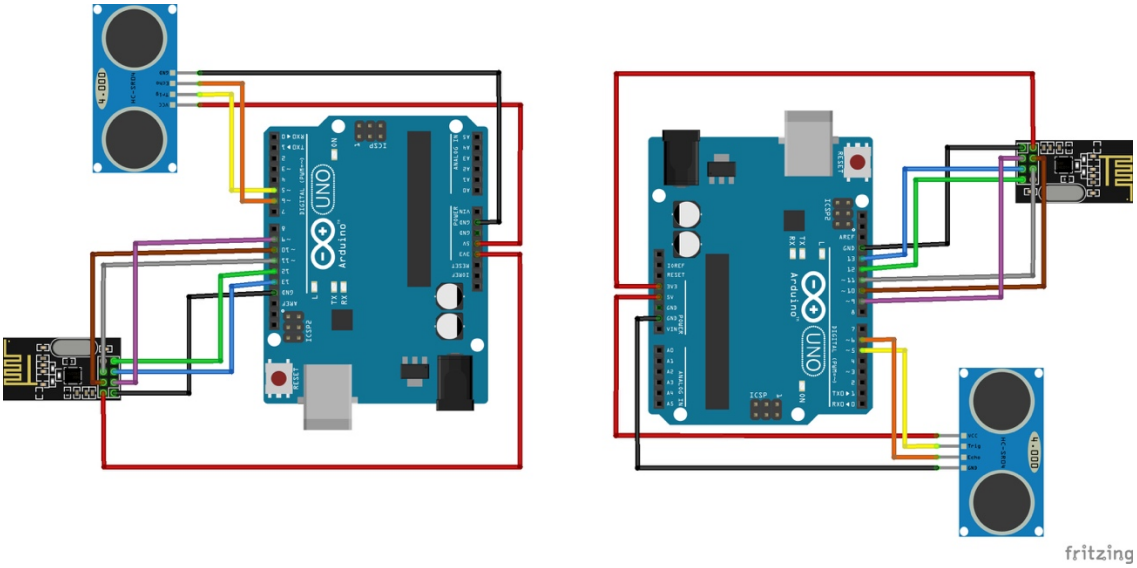


Figura 3.4: Tercer montaje (Emisor A + Emisor B)

En la práctica, la función a realizar será la siguiente: Una vez el emisor A mande el pulso de ultrasonido y escriba un mensaje, el emisor B tiene que ser capaz de reconocer este mensaje proveniente del emisor A, enviar un pulso de ultrasonido y escribir su propio mensaje para que el siguiente objeto emisor pueda reconocerlo y así seguir con la transmisión del mensaje, tal y como se muestra en la Figura 3.5.

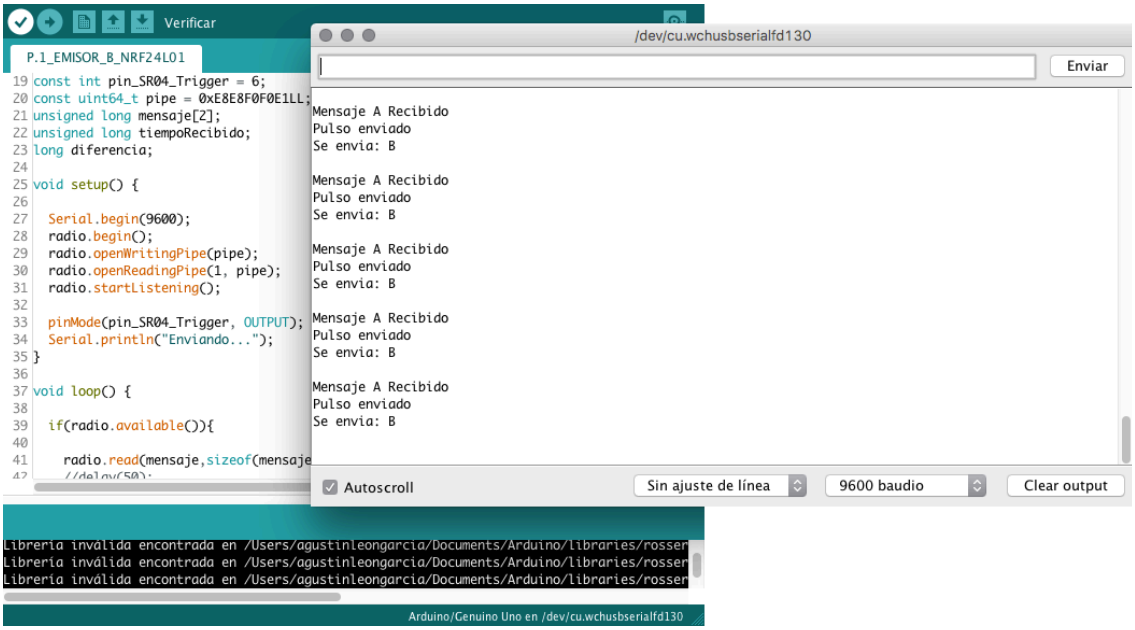


Figura 3.5: Mensaje recibido y pulso enviado por segundo emisor

Cabe recordar que, tanto en el emisor A como en el B para un correcto funcionamiento, han sido añadidas las librerías RF24-master en ambos códigos de Arduino.

3.1.4 Cuarto montaje. Circuito de objetos emisores

El siguiente paso será el de cerrar el circuito de comunicación. Para ello, se han montado los tres emisores que serán los encargados de enviar los tres haces de ultrasonido y se han montado todos de la misma forma, cada uno con su Arduino, su sensor de ultrasonido y su transceptor de radiofrecuencia.

El proceso es el mismo. El emisor A envía el pulso y escribe el mensaje, que es recogido por el emisor B. Acto seguido, este manda su pulso y escribe

su mensaje. Una vez el emisor C reciba el mensaje del emisor B, mandará su pulso y escribirá su mensaje para que el emisor A lo reciba y se inicie de nuevo el circuito de emisión como se puede ver en la Figura 3.6. El propio código estará adecuadamente implementado para que ningún emisor envíe pulso sin recibir el mensaje que le corresponde, puesto que los tres emisores estarán escuchando todos los mensajes escritos por cada uno de los emisores.

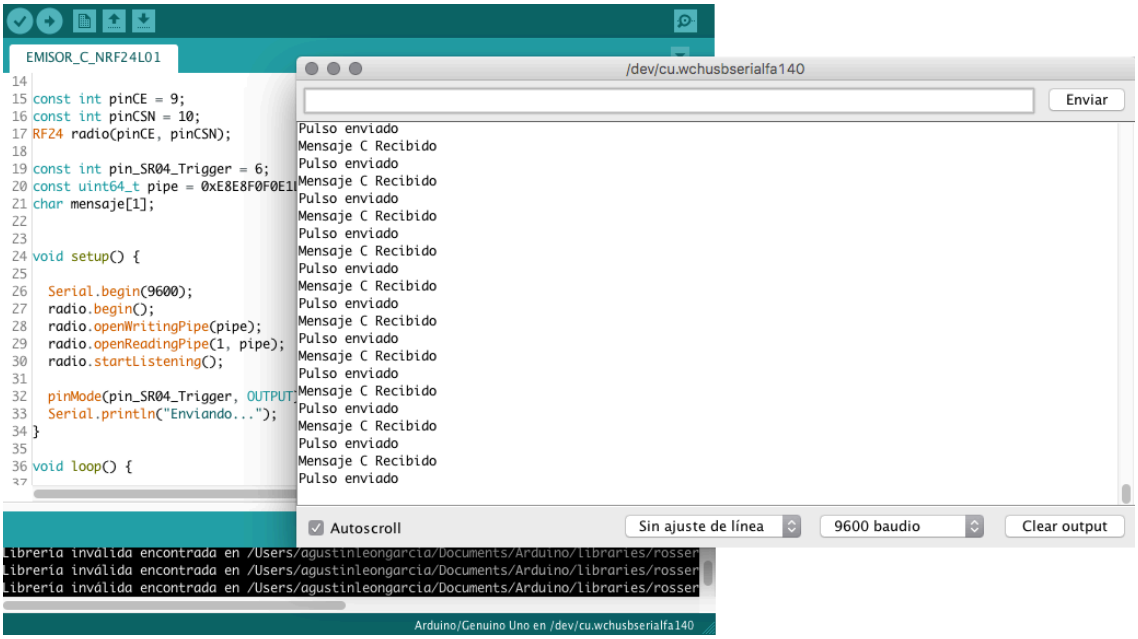


Figura 3.6: Mensaje recibido en el tercer emisor

Cabe destacar que en el código del primer emisor se han añadido unas líneas que permitirían reiniciar la comunicación, en caso de que esta se quede colgada al cabo de cierto tiempo, por ejemplo 10 segundos.

3.1.5 Quinto montaje. Emisor + receptor sin interrupciones

Una vez el circuito de objetos emisores esté en funcionamiento, el siguiente paso será medir distancias entre uno de los emisores y el receptor.

En esta primera prueba, no se ha tenido en cuenta las interrupciones, por lo que solo uno de los seis sensores ultrasónicos será el que esté conectado al Arduino Mega.

Antes de proceder a la explicación del montaje y dejando constancia para cada uno de los montajes en los que el objeto receptor tenga un papel principal en el montaje, hay que comentar que los módulos de ultrasonido utilizados para este están levemente modificados de la siguiente forma:

Cada uno de los sensores posee un microcontrolador (a parte de los amplificadores de señal) que es el encargado de indicar al pin Echo que los 8 pulsos de Trigger han sido enviados. En el caso del proyecto, los módulos del receptor en ningún momento emiten señal de Trigger, por lo que si se conectara el cable del Echo directamente al pin de salida de Echo del propio módulo, nunca tendríamos un pulso a 1 en ese pin. Es por esto que se ha decidido soldar directamente al pin Echo (antes de entrar a este microprocesador) el cable que nos devuelva la señal de Echo en crudo, por llamarla de alguna forma. De esta forma, cualquier pulso enviado por cualquiera de los emisores podrá ser recogida y enviada al Arduino sin mayor complicación.

Para la realización de la prueba, se ha tomado el emisor A en modo “sin respuesta”. Este es el modo que hemos conectado en el apartado anterior en el que, si este no recibía ningún mensaje en un determinado tiempo, se enviaría de nuevo el mensaje de inicio de ciclo, por lo que siempre se tendrá un pulso constante cada 5 segundos por ejemplo.

El código del receptor se ha realizado en base a la función millis() del Arduino. El código tomará dos valores de micros() tomados en dos

momentos diferentes: uno cuando el emisor manda el mensaje (y por ende pulso) y es recibido por el objeto receptor y un segundo valor tomado cuando el pin Eeho del receptor pasa de 0 a 1, lo que significará que se ha recibido la señal. La diferencia de ambos valores será el tiempo, en microsegundos, que ha tardado la señal en llegar desde que es enviada por el emisor hasta que es recibida en el Echo del objeto receptor.

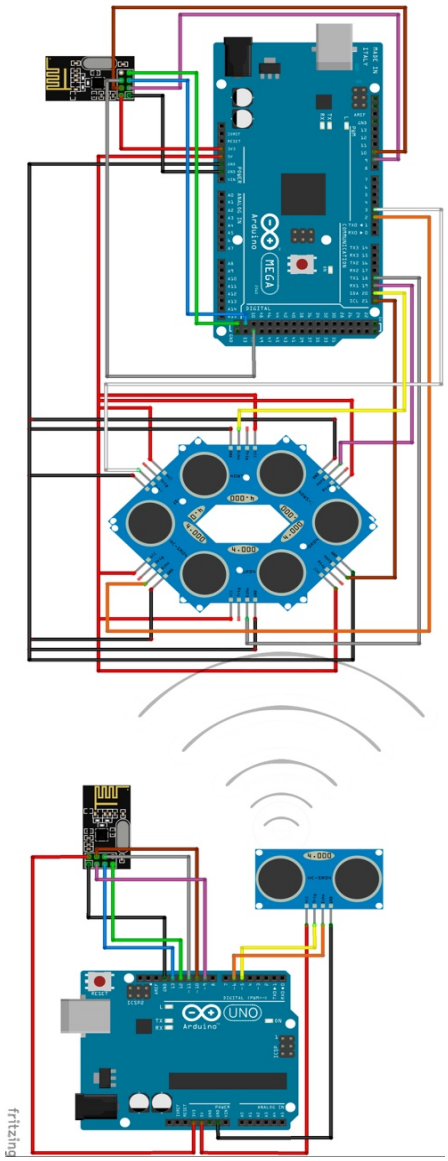


Figura 3.7: Quinto montaje (Emisor + receptor sin interrupciones)

Para el cálculo de la función que relaciona este tiempo con la distancia a la que se encuentran ambos objetos, se ha utilizado una tabla de Excel y se han tomado datos desde los 20 centímetros hasta los 2 metros y 20 centímetros usando el montaje experimental mostrado en la Figura 3.10. Una vez obtenidos estos datos y plasmados en una gráfica de carácter lineal, se ha obtenido la ecuación de la recta que relacionará el valor en microsegundos con una distancia en centímetros.

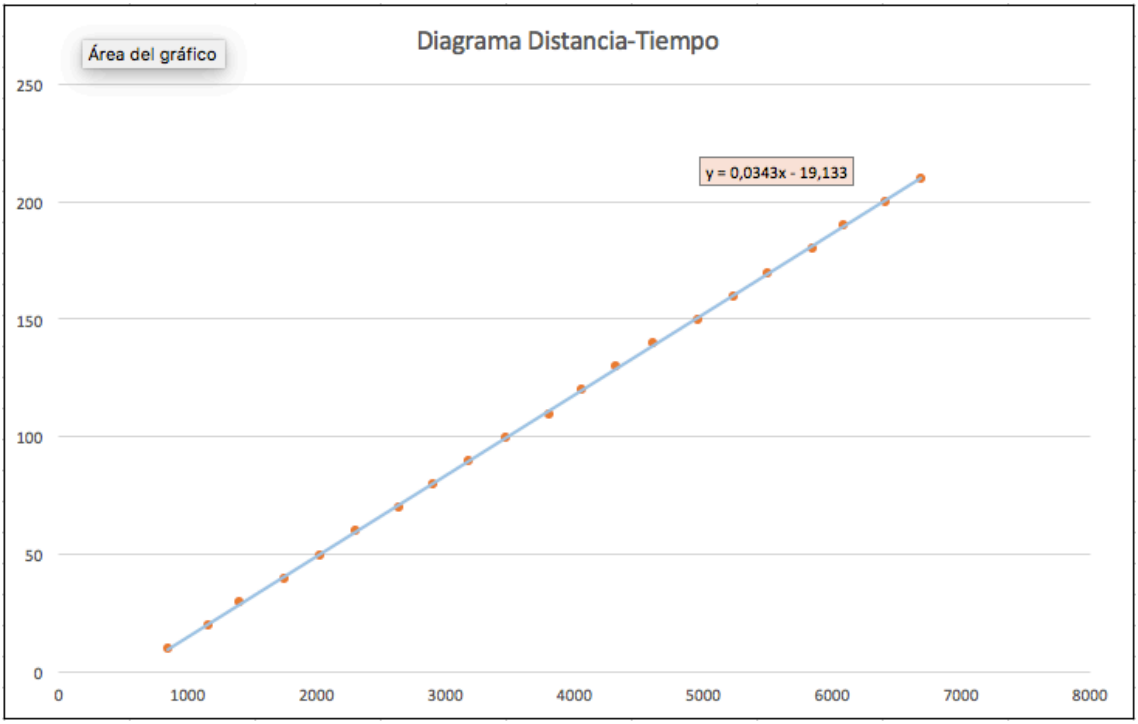


Figura 3.8: Diagrama distancia-Tiempo del quinto montaje

Para el cálculo de la distancia, se ha implementado un código que hará lo siguiente: el emisor envía el mensaje y en ese mismo instante, envía el pulso. En ese preciso momento en el que el receptor recibe el mensaje desde el emisor, se almacena el valor de la función micros() en una variable.

Seguidamente, en cuanto el receptor recibe la señal de ultrasonido y su pin Echo es puesto a uno, se almacena el valor de la función micros() en otra variable. Finalmente, para obtener el tiempo que ha tardado la señal en llegar desde que es enviada hasta que es recibida por el receptor, se hace una diferencia matemática entre el último valor de micros() y el primero, obteniendo así el valor buscado.

3.1.6 Sexto montaje. Emisor + Receptor con interrupciones

En esta ocasión, el principal objetivo es el de obtener la distancia de nuevo, pero con un código basado en interrupciones, que es como finalmente se implementará el código en el objeto receptor.

El procedimiento ha sido el mismo que en la prueba anterior, utilizando tan solo uno de los sensores ultrasónicos del objeto receptor y con el receptor A en modo "sin respuesta", con un pulso estable de ultrasonido cada 5 segundos.

En cuanto al código, se han usado las interrupciones para el pin del ultrasonido conectado. Esta interrupción se activará una vez se reciba el pulso de ultrasonido en el pin del Echo. Puesto que el timer de la interrupción siempre estará activos hay que cerciorarse de ponerlo a 0 una vez recibamos el mensaje. Para cuando el pin del Echo pase de FALSE a TRUE, el contador tendrá un valor que será almacenado en una variable. Esta variable será el valor de tiempo que ha pasado desde que se ha enviado el pulso hasta que se ha recibido en el objeto receptor.

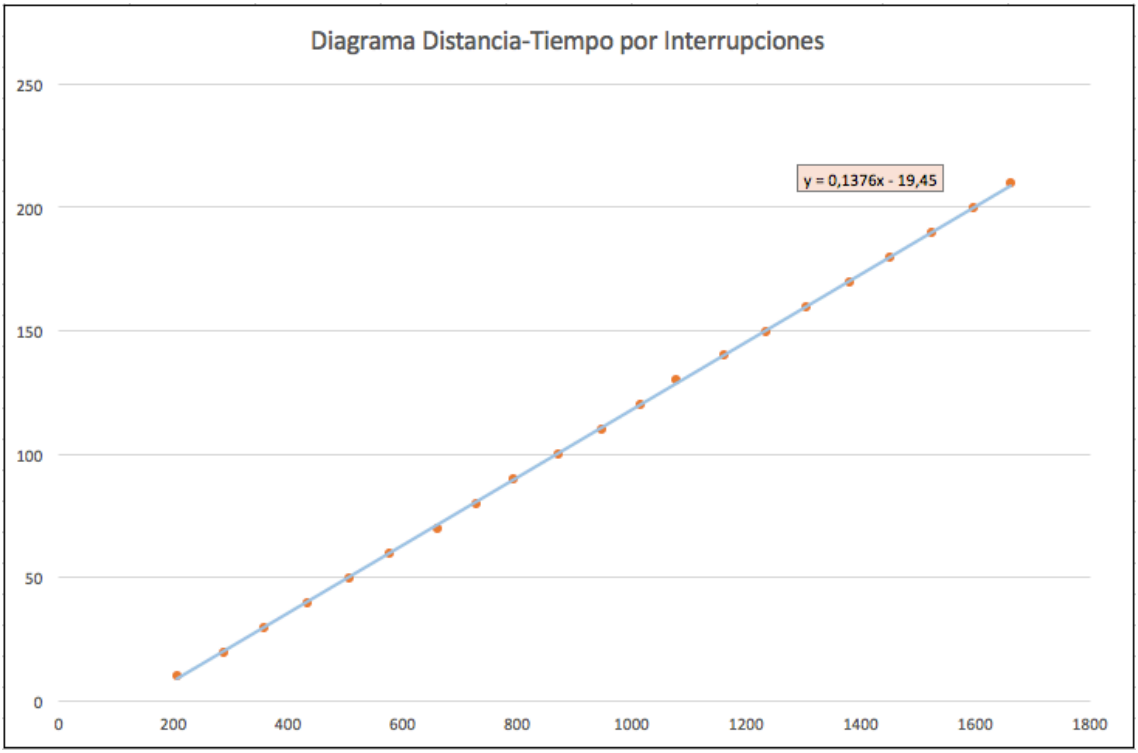


Figura 3.9: Diagrama Distancia-Tiempo con interrupciones

Para el cálculo de la función que relaciona este tiempo con la distancia a la que se encuentran ambos objetos el proceso es el mismo que en el anterior montaje. Se ha utilizado una tabla de Excel y se han tomado datos desde los 20 centímetros hasta los 2 metro y 20 centímetros usando el montaje experimental mostrado en la Figura 3.10. Una vez obtenidos estos datos y plasmados en una gráfica de carácter lineal, se ha obtenido la ecuación de la recta que relacionará el valor en microsegundos con una distancia en centímetros.

Para el cálculo de la distancia, se ha implementado un código que hará lo siguiente: el emisor envía el mensaje y en ese mismo instante, envía el pulso. En ese preciso momento en el que el receptor recibe el mensaje desde el emisor, se almacena pone a 0 el timer de la interrupción. Seguidamente,

en cuanto el receptor recibe la señal de ultrasonido y su pin Echo es puesto a uno, se activará la interrupción, la cual posee un timer interno que nos indica el tiempo que lleva ejecutándose desde su puesta a 0. Finalmente, para obtener el tiempo que ha tardado la señal en llegar desde que es enviada hasta que es recibida por el receptor, se toma este valor del timer de la interrupción, que es almacenado en una variable.

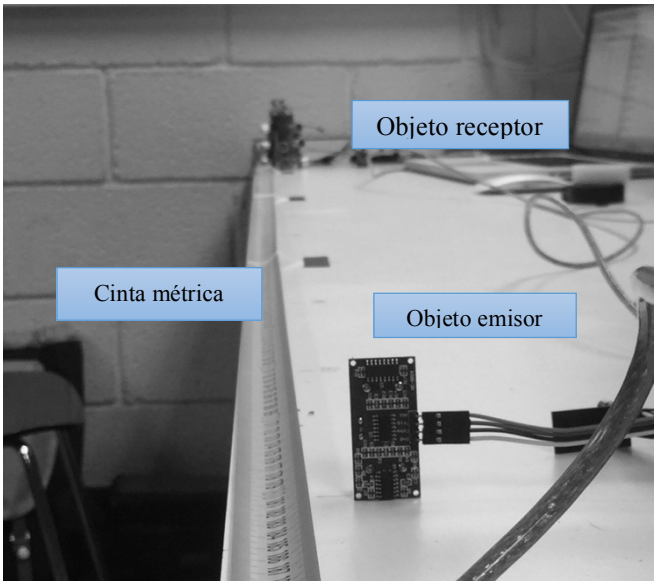


Figura 3.10: Montaje experimental para medida de distancia

3.1.7 Montaje final. Circuito de emisores + objeto receptor

Para finalizar el montaje del equipo de localización, lo que se ha hecho ha sido tomar finalmente todas y cada una de las partes del proyecto.

Se ha cargado en los tres objetos emisores su programa correspondiente de emisión para crear el circuito de emisión de ultrasonido que teníamos previamente funcionando. Se han conectado todos y cada uno de los ultrasonidos a los pin de I/O del Arduino MEGA destinados a las

interrupciones y se ha modificado el código para que estas interrupciones afecten a todos y cada uno de los ultrasonidos.

Para el cálculo de la distancia, se ha utilizado la ecuación de la recta obtenida en el sexto montaje ya que es la que más se acerca al montaje que se tendrá finalmente en el proyecto, incluyendo el código con las interrupciones para cada ultrasonido.

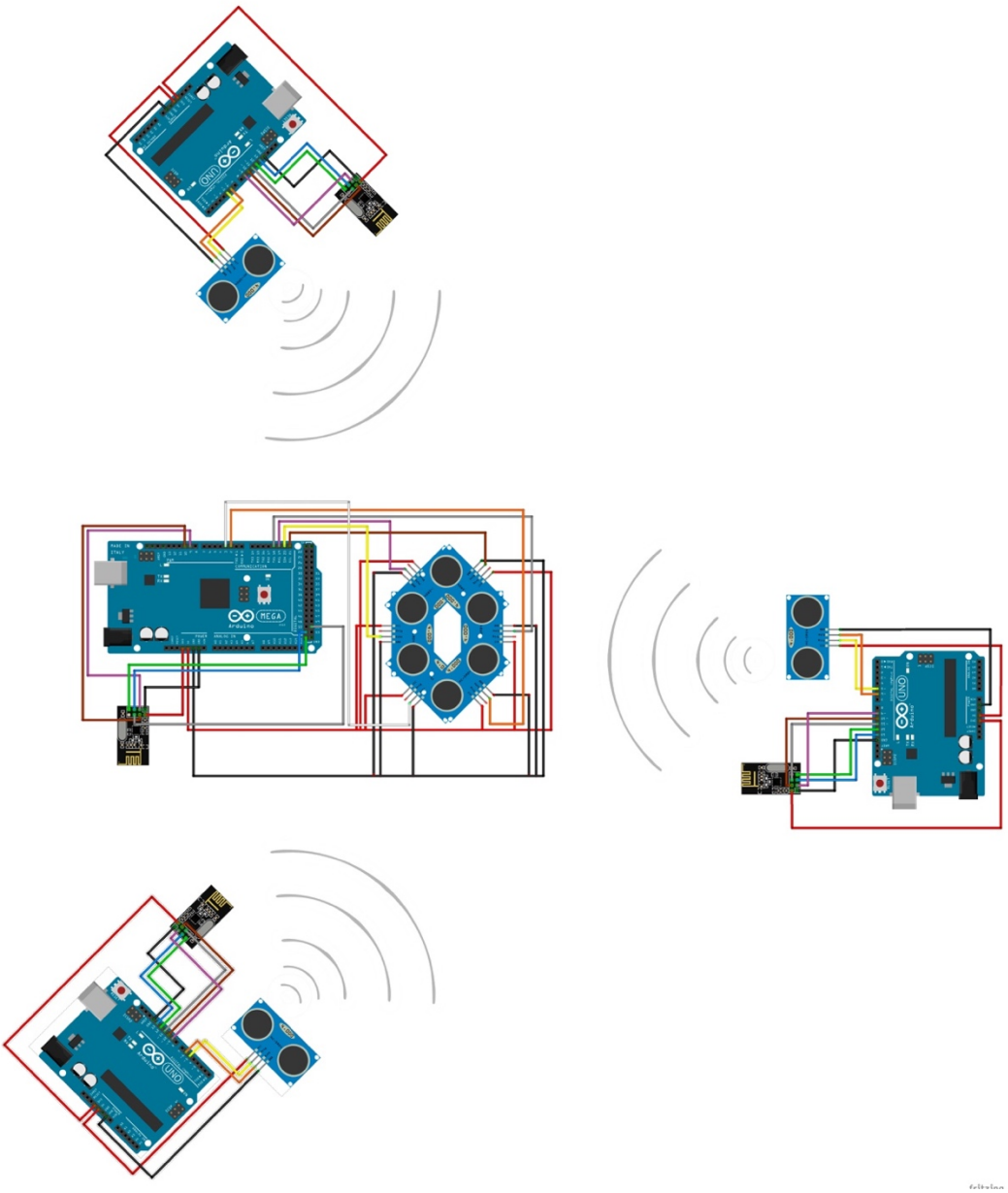


Figura 3.10: Montaje experimental final

3.2 Obtención de ángulo de señal

Uno de los problemas presentados también a lo largo del proyecto era el de cómo saber en qué ángulo se encuentra el robot con respecto al emisor y cómo se podía obtener este ángulo. La solución que se ha presentado para remediar este inconveniente ha sido la de estudiar como se comportan los sensores mediante el tiempo de llegada de la onda según el ángulo de giro.

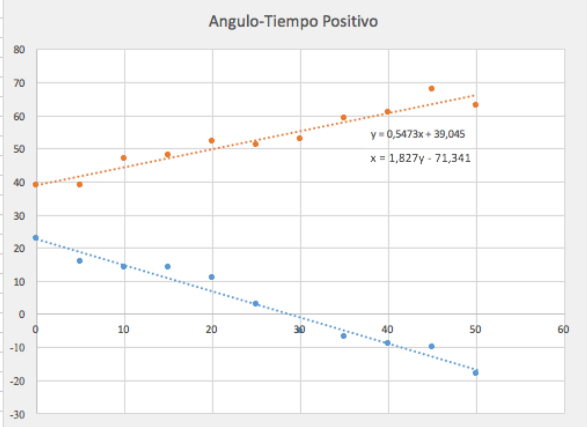
La prueba se ha realizado utilizando tres de los sensores de ultrasonido del objeto receptor y un objeto emisor que sería el encargado de emitir la onda. Haciendo una primera prueba con el módulo receptor central totalmente en paralelo con el emisor, se ha ido girando un total de 50º en ambos sentidos en intervalos de 5º, obteniendo en cada uno de estos intervalos, el timing de cada uno de los tres sensores.

Según el sentido de giro, se ha observado como dos de los emisores se iban alejando más del emisor (tardaban más tiempo en recibir la señal) y otro de ellos, se acercaba hasta quedar totalmente en paralelo con el emisor, obteniendo el timing inicial del receptor central. Una vez obtenidos los datos y presentado una tabla con ellos, se ha operado estos resultados, haciendo una diferencia entre el módulo central y los módulos de los extremos respectivamente.

En la figura 3.11 se observa cómo, según el sentido de giro, una de estas diferencias va en aumento, y la otra como es obvio, en decrecimiento.

Grado de giro	Sensor 1	Sensor 2	Sensor 3	T1-T2	T3-T2
0	441	418	457	23	39
5	436	420	459	16	39
10	438	424	471	14	47
15	441	427	475	14	48
20	433	422	474	11	52
25	432	429	480	3	51
30	429	434	487	-5	53
35	425	432	491	-7	59
40	422	431	492	-9	61
45	420	430	498	-10	68
50	418	436	499	-18	63

Grado de giro	Sensor 1	Sensor 2	Sensor 3	T1-T2	T3-T2
0	441	418	457	23	39
-5	447	421	449	26	28
-10	450	423	444	27	21
-15	457	423	442	34	19
-20	468	431	440	37	9
-25	475	435	437	40	2
-30	478	437	435	41	-2
-35	481	435	432	46	-3
-40	481	438	429	43	-9
-45	488	441	425	47	-16
-50	494	444	423	50	-21



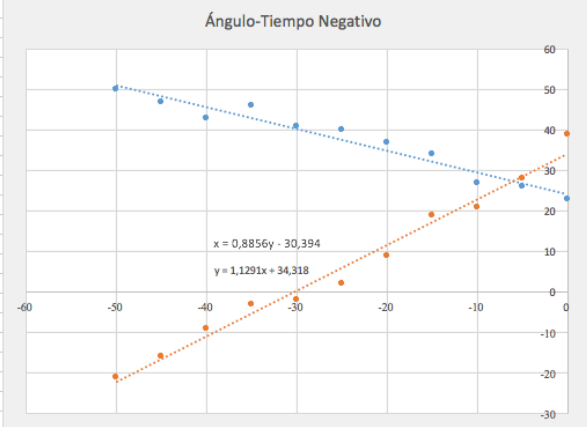


Figura 3.11: Tablas y diagramas de Tiempo-Ángulo

Una vez hecho esto se podrá tener una idea aproximada de cuál es la dirección o ángulo de orientación del robot tomando los valores del sensor que se precise y comparándolos en dicha tabla, obteniendo un valor de hasta 50º de rotación, tanto en el eje positivo como en el negativo.

Para el cálculo del ángulo, se ha implementado un código que hará lo siguiente: se han utilizado únicamente 3 sensores para la prueba, por lo que uno de ellos, el central, deberá estar totalmente en paralelo para iniciar la prueba y así tomar ese valor de tiempo como ángulo 0º. Utilizando la medida de distancia mediante interrupciones que se ha comentado en el apartado 3.1.6 de esta memoria, se han tomado datos de los timers de las interrupciones en este momento y se han tomado como ángulo 0º, como se

ha comentado. A partir de aquí, y girando el receptor de 5º en 5º, se ha hecho la misma operación para observar cómo variaban los timers en cada sensor.

En el código final, como se puede ver en el anexo A.4, lo que se ha hecho es comparar los sensores de dos en dos. En primer lugar se ha comparado el sensor más cercano, obtenido dentro del código mediante comparaciones de los timers de cada sensor, con su posterior y su anterior respectivamente. Tomando la diferencia de timers obtenida en la prueba como ángulo 0º, en este caso 39 ms, se ha comparado estos sensores para comprobar si la diferencia era menor o mayor, para así saber el signo del ángulo y aplicar una de las dos fórmulas obtenidas en el gráfico 3.11.

Una vez esta comparación haya sido realizada, se calculará el ángulo de giro con una u otra ecuación según convenga, obteniendo el ángulo de giro con respecto a este sensor más cercano y por cada uno de los tres emisores (cada emisor tendrá un sensor más cercano y serán distintos en cada uno de estos emisores).

3.3 Diseño 3D

Para la creación del ambiente de trabajo se ha optado por la creación de módulos en PLA impresos en 3D para poder albergar tanto las placas de Arduino como los sensores de ultrasonido y radiofrecuencia.

Estos modelos se han diseñado en SketchUp partiendo desde cero y con un acabado sencillo para las placas de Arduino y algo más complejo para los sensores. Los modelos realizados han sido 3:

- Diseño para los emisores. Serán capaz de alojar tanto la placa Arduino como los sensores de ultrasonido y radiofrecuencia.

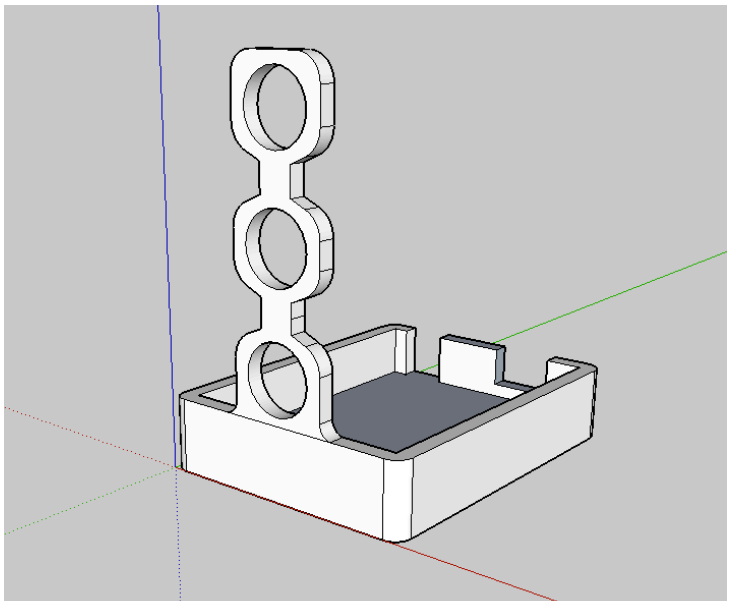


Figura 3.12: Diseño para objeto emisor

- Diseño para Arduino Mega. Será capaz de alojar la placa de Arduino Mega que irá encima del robot GopiGo, así como su módulo de RF.

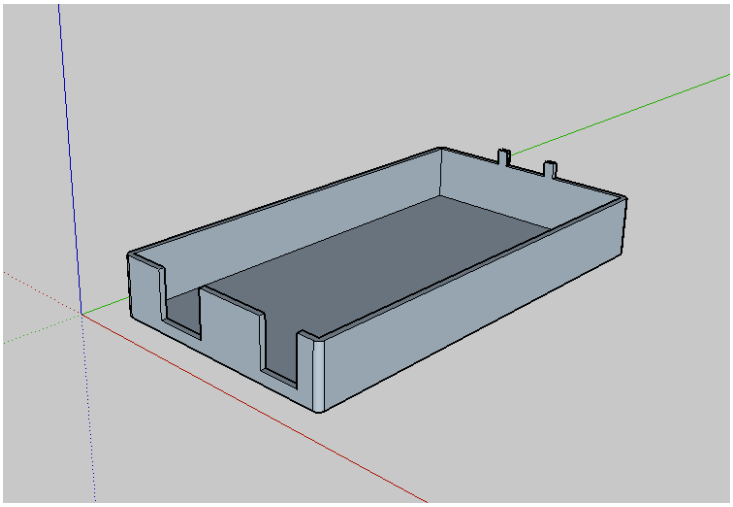


Figura 3.13: Diseño para placa Arduino MEGA (Receptor)

- Diseño para el receptor. Será el encargado de albergar los seis sensores que conforman el objeto emisor. Estarán dispuestos en posición hexagonal, uno en cada cara del hexágono. El interior y parte baja hueco será para alojar los cables y que no puedan interferir en el funcionamiento de los sensores.

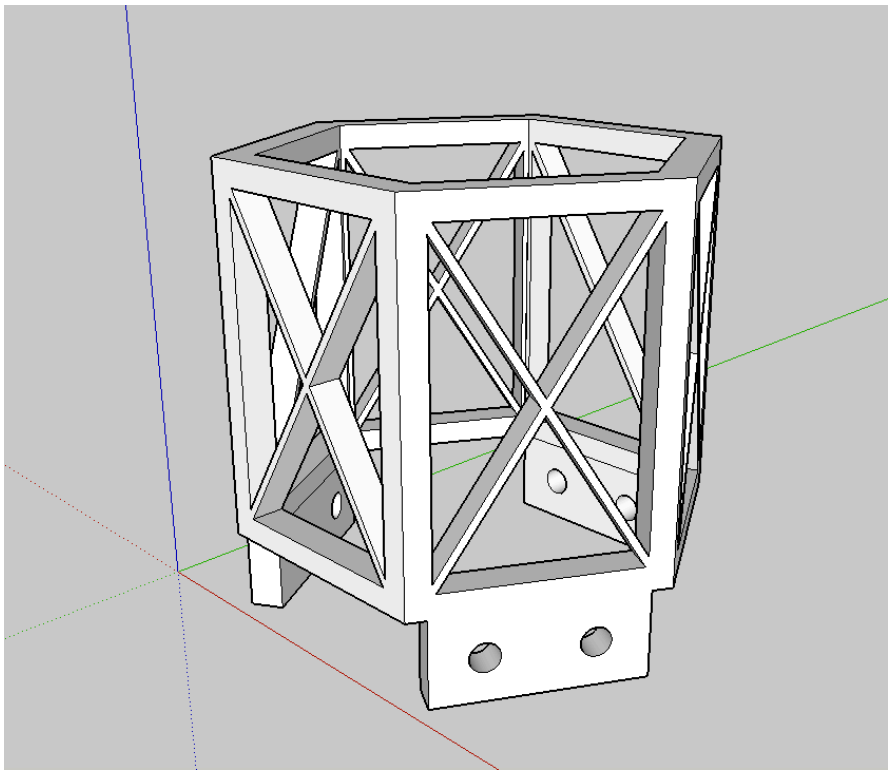


Figura 3.14: Diseño para objeto receptor

Una vez realizados los diseños en 3D, se han exportado en formato STL para un fácil manejo en el programa UltimakerCura. Una vez importado el archivo en el programa, se podrá definir patrones como la altura de capa y el porcentaje de relleno que se desea en la pieza, factores que irán ligados a la calidad del resultado final.

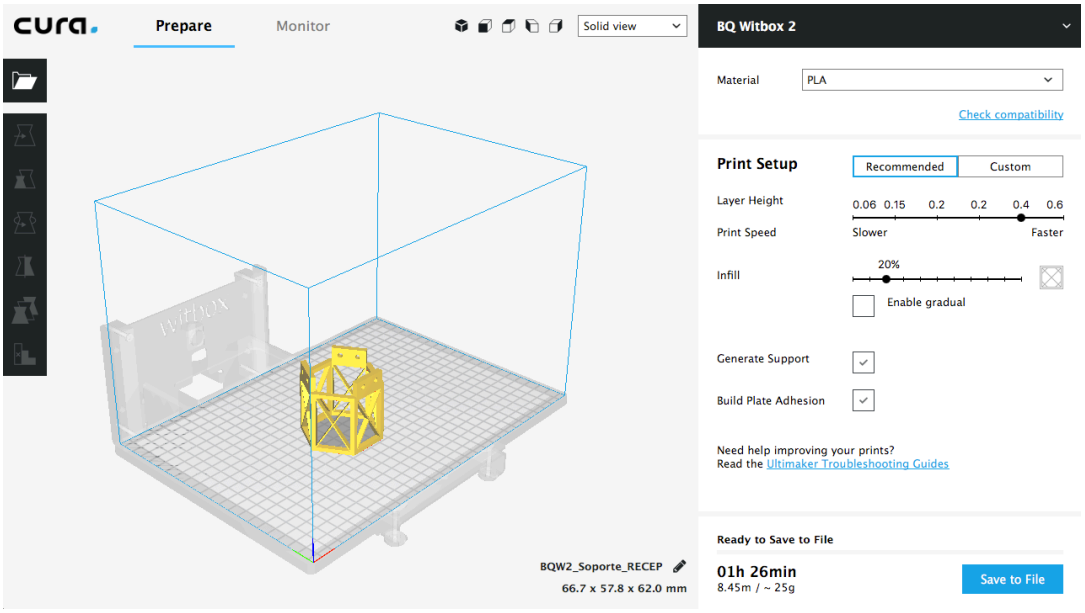


Figura 3.15: UltimakerCura con el diseño del objeto receptor

Una vez se han impreso las piezas, se ha pasado al montaje final del robot y el objeto emisor, obteniendo como resultado el mostrado en la Figura 3.16. para el objeto emisor y Figura 3.17 para el Gopigo2 con el objeto receptor y su correspondiente Arduino MEGA ya montado.

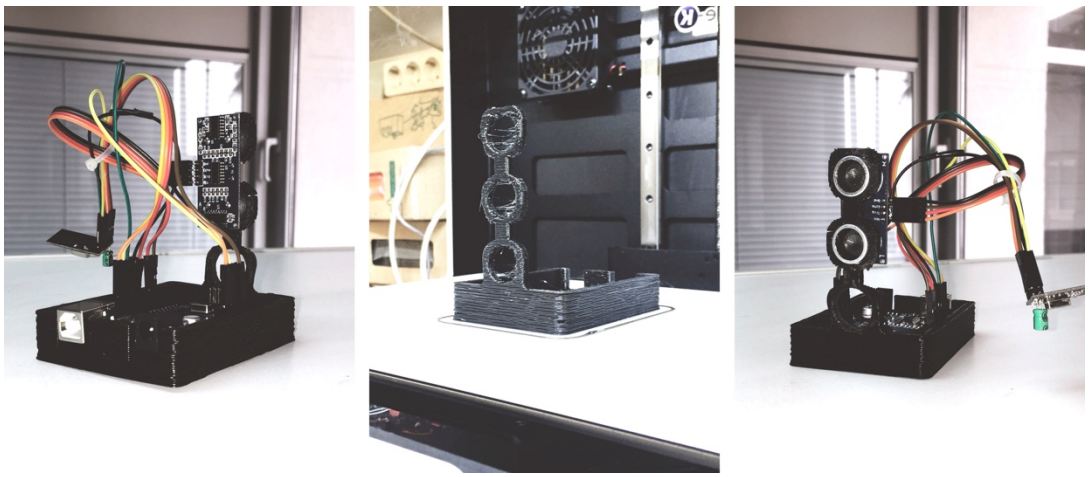


Figura 3.16: Montaje final del objeto emisor

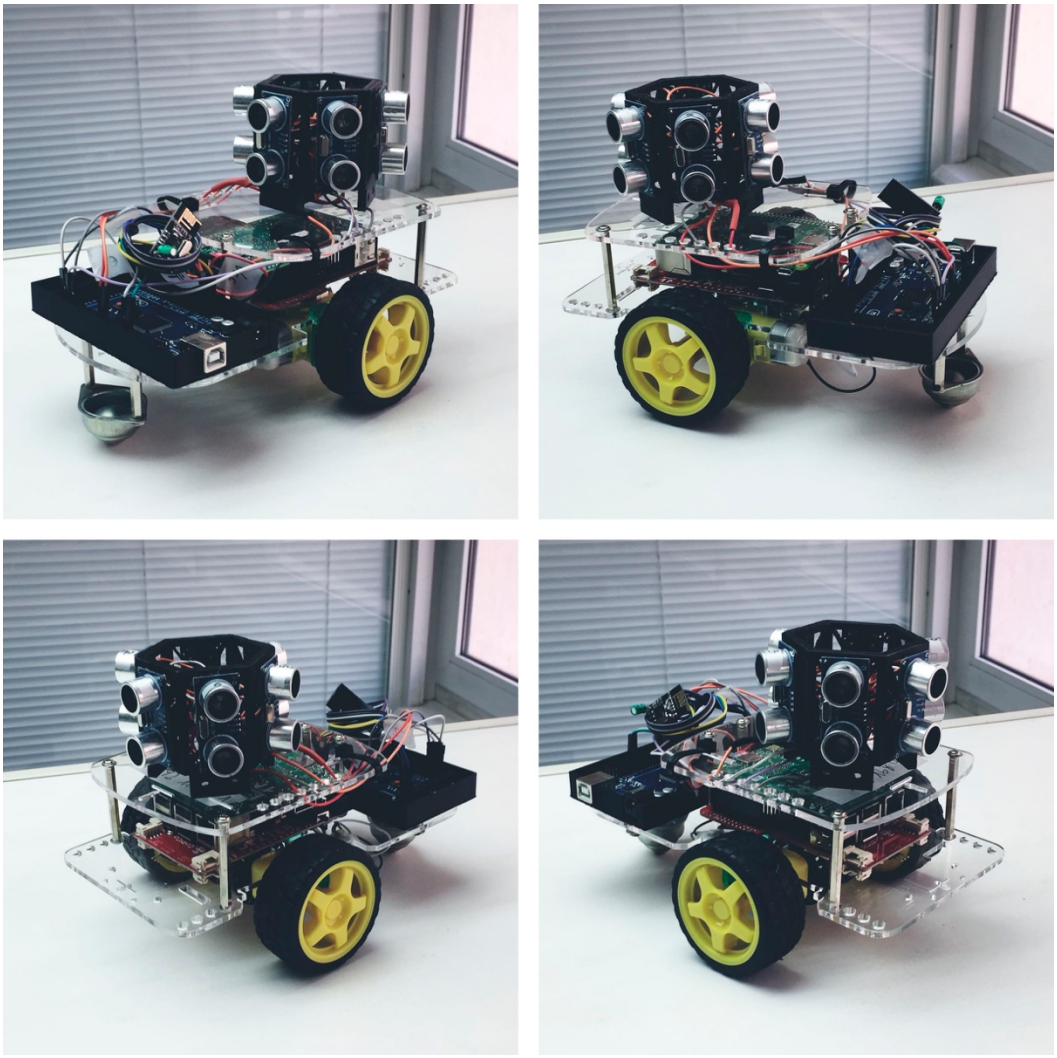


Figura 3.17: Montaje final del robot

3.4 Desarrollo de la comunicación. Raspberry y ROS

En el siguiente apartado del marco práctico se comentará el desarrollo que se ha ejecutado para llevar a cabo la comunicación entre el objeto receptor y el robot, todo esto a través de la plataforma Raspberry Pi.

3.4.1 Instalación de paquetes y SO

En este apartado se van a diferenciar dos medios de comunicación. El primero será las Raspberry, que recibirá los datos del objeto receptor y enviará el paquete con la odometría al segundo medio, que será un ordenador con Linux como sistema operativo y que tendrá también instalado ROS Kintetic.

En primer lugar se ha instalado en la Raspberry una versión basada en Debian y por lo tanto en Linux orientada al uso en la Raspberry Pi llamada Raspbian. En este caso, se ha instalado la ultima versión Raspbian Stretch en su versión June 2018, que como se ha comentado, esta basado en Debian Stretch. Esto permitirá crear un entorno intuitivo y de fácil uso para la implementación a posteriori de ROS Kinetic. Para la instalación de este sistema operativo en la Raspberry Pi se ha optado por el método más directo, que es el de quemar la imagen del SO directamente en la tarjeta MicroSD que se utilizará en la Raspberry con la ayuda del programa Etcher, tal y como se recomienda en la propia web de Dexter Industries

Figura 3.18: Etcher

Una vez instalado en la tarjeta, este es el aspecto que posee este sistema operativo dentro de la Raspberry Pi.

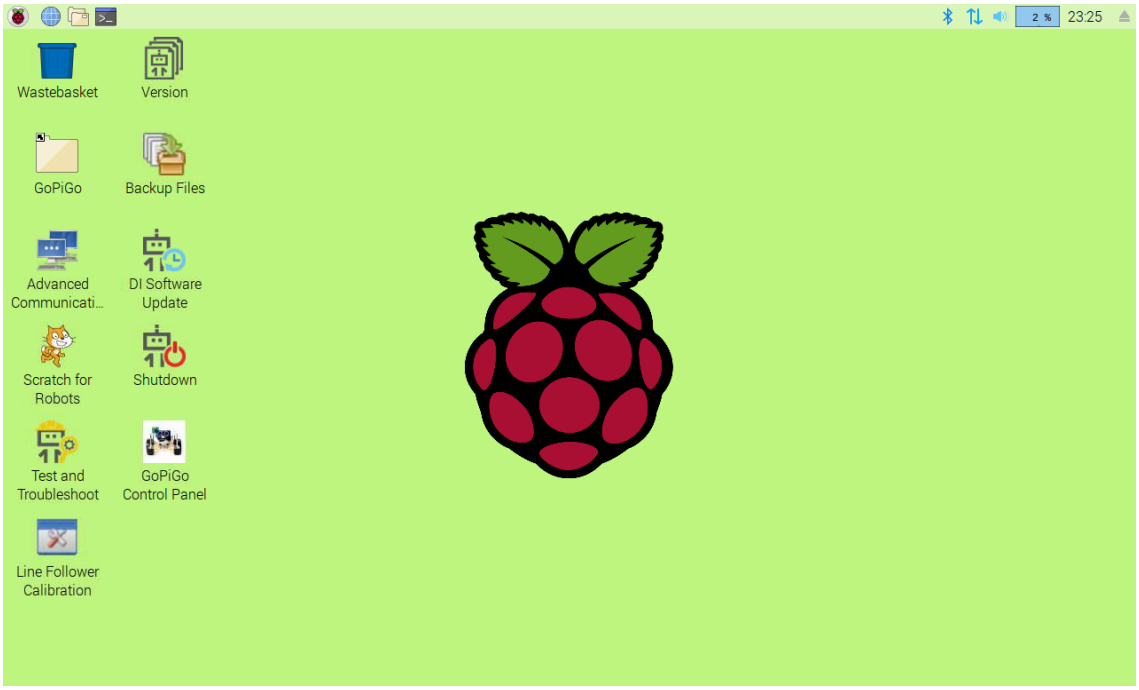


Figura 3.19: Escritorio Raspbian Stretch

Es el momento de instalar ROS Kinetic en la Raspberry ya que será de gran ayuda para la comunicación entre el robot y el ordenador externo. Para ello se ha seguido un tutorial en el que se indica paso por paso como descargar los paquetes, crear el espacio de trabajo de ROS (catkin_ws) y construir y compilar los paquetes y librerías una vez instalados.

Cabe destacar también que se ha configurado la Raspberry como punto de acceso WiFi, para así poder operar con ella sin la necesidad de usar todo el rato una pantalla externa. Para ello simplemente se ha modificado ciertos parámetros del network interfaces de la raspberry, como muestra la figura 3.20, para hacerla accesible con una IP estática. Dos de los paquetes necesarios para esta modificación son los siguientes:

- **hostapd** que será el paquete que permita usar el módulo WiFi que posee la propia Raspberry Pi como punto de acceso.
- **dnsmasq** que es una combinación del protocolo DHCP y el servidor DNS que será de gran ayuda a la hora de configurar la interface Wlan.

Una vez se tenga configurado correctamente dichos paquetes y se inicien los servicios de hostapd y dnsmasq, se podrá acceder a la Raspberry mediante su propio punto de acceso WiFi.

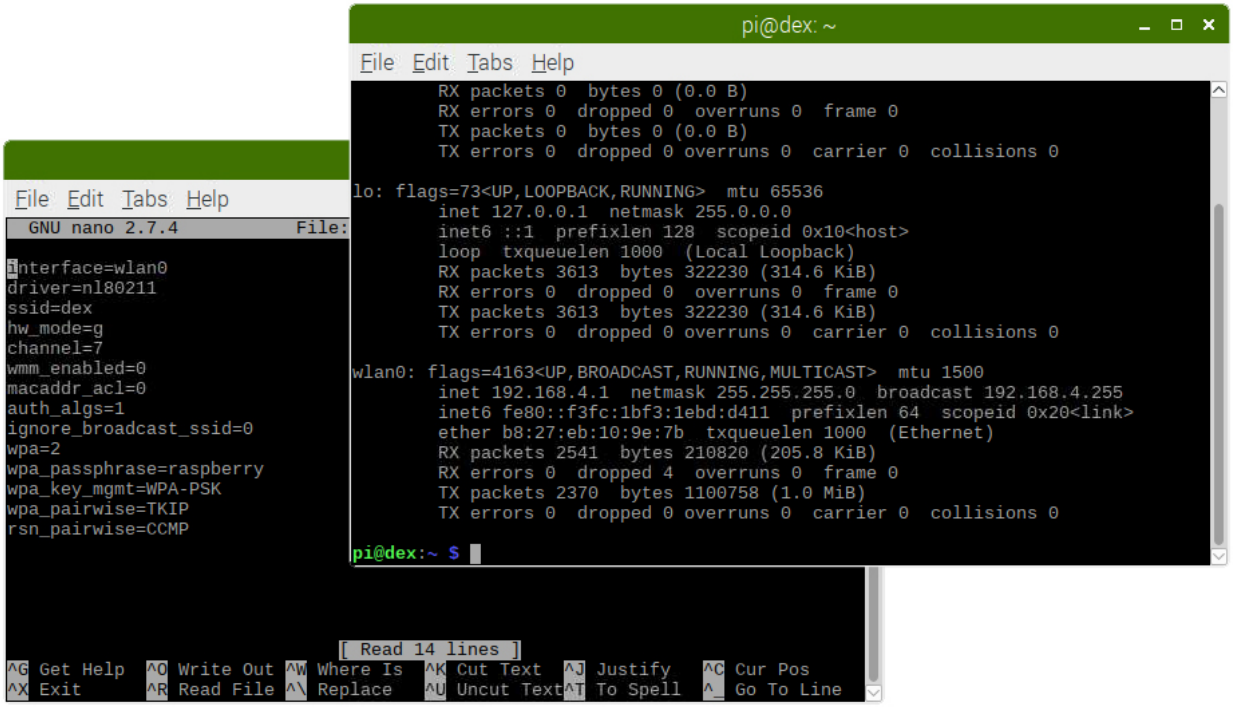


Figura 3.20: Configuración network de Raspberry Pi3

El segundo medio de comunicación, un ordenador personal, ha sido dotado de la última versión de Ubuntu. Puesto que el ordenador utilizado ya poseía un sistema operativo nativo, se ha optado por el uso de una máquina virtual. Una vez descargada la imagen en extensión .iso desde la web oficial

de Ubuntu, la instalación de la máquina virtual ha sido muy fácil e intuitiva gracias a la aplicación VMware Fusion, la cuál se muestra en la Figura 3.21.

La versión instalada ha sido Ubuntu Desktop Xenial Xerus (16.04 LTS) x64 y se ha dotado a la máquina virtual de 2 núcleos de procesador, 1532Mb de memoria RAM y 40GB de SSD (SCSI) entre otras características.

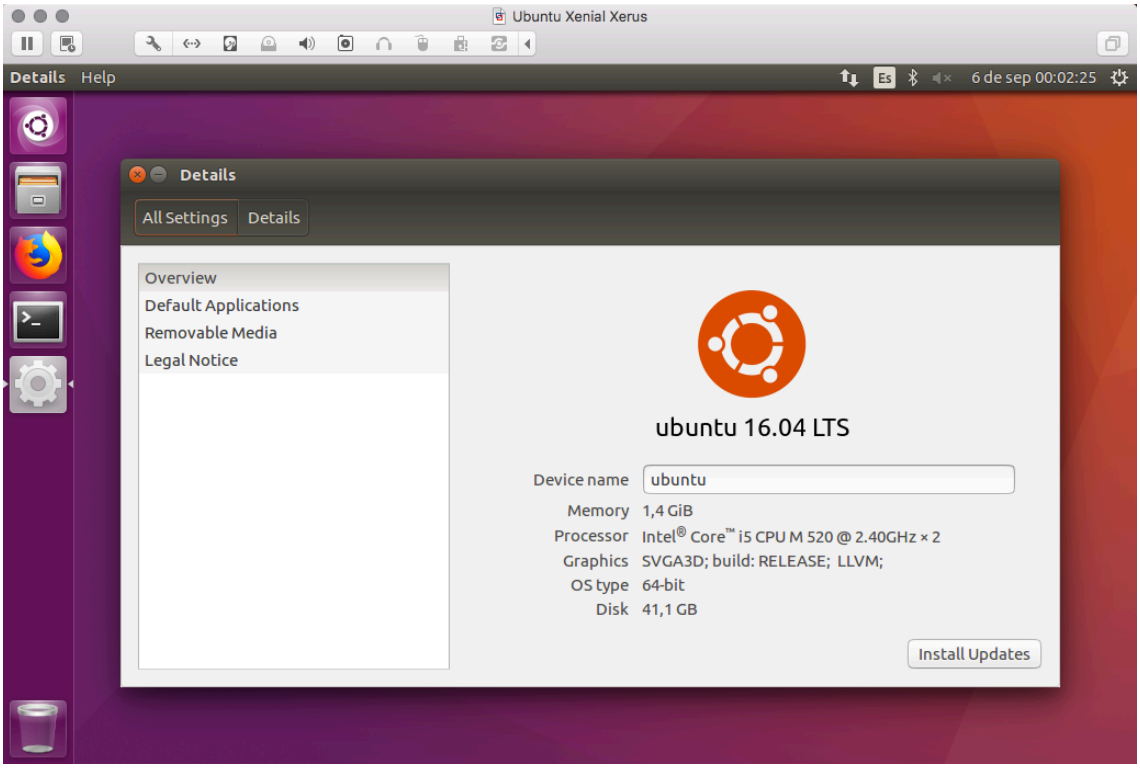


Figura 3.21: VMware Fusion ejecutando Ubuntu Xenial

Capítulo IV.

Conclusiones y líneas futuras

Se expone en este capítulo un balance objetivo del trabajo realizado y objetivos cumplidos así como posibles formas de mejorar el proyecto en una siguiente fase.

4.1 Conclusiones y líneas futuras

Como conclusión, se puede decir que este trabajo de fin de grado ha servido para desarrollar un sistema de localización basado en emisores de ultrasonido y conexiones de radiofrecuencia, que son ámbitos en los que no acostumbra a moverse un ingeniero electrónico. Planteado esto, se puede decir que se ha sabido resolver la mayoría de problemas que han ido apareciendo a lo largo del desarrollo del proyecto. Así pues, se ha adquirido un conocimiento, tanto teórico como de trabajo, al que no se estaba acostumbrado.

Como se ha visto a lo largo de la memoria, en ambas partes, hardware y software, se ha tenido que realizar modificaciones, tanto en módulos hardware como en códigos, modelos 3D, etc. Estas modificaciones, a parte de mejorar de forma sustancial el proyecto, también han servido para que el resultado final se adecuara a lo que se quería.

Durante el desarrollo de la parte de ROS se han encontrado muchas dificultades a la hora de su implementación, por lo que no se ha desarrollado del todo esta función, quedando así como una posible mejora para que finalmente pueda realizar su función al completo.

Así pues, como líneas futuras, se podría desarrollar esta función comentada de una mejor manera, así como optimizaciones del software en la programación de los objetos, tanto emisores como receptores, para una mejora del entorno. Cabe decir también, que el apartado de la orientación del robot no ha tenido el resultado esperado, si bien el robot es capaz de orientarse, la implementación del código y su algoritmo de orientación podría mejorarse en un próximo desarrollo.

4.2 Conclusions and open lines

In conclusion, this final project has served to develop a localization system based on ultrasound emitters and radiofrequency connections, which are areas where an electronic engineer does not usually move. It can be said that it has been known to solve most of the problems that have been appearing throughout the development of the project. So, a knowledge have been acquired both theoretical and practical.

As it has been seen throughout the memory, in both parts, hardware and software, it has been necessary to make modifications: hardware modules, programming codes, 3D models, etc. These modifications, apart from substantially improving the project, have also served to ensure that the final result matches what was wanted.

During the development of the ROS part, many difficulties have been encountered when it was implemented, so this function has not been fully developed, leaving a possible improvement so that it can finally perform its function.

Therefore, as open lines, this function could be developed in a better way, as well as programming software optimizations, both emitters and receivers, for an improvement of the environment. Moreover, the orientation system of the robot has not had the expected result, although the robot is able to orient itself, the implementation of the code and its orientation algorithm could be improved in the next development.

Capítulo V.

Presupuesto

En el siguiente capítulo se detallará el presupuesto de los costes estimados del proyecto en su totalidad. Se expondrá un primer presupuesto de los materiales y componentes utilizados y un segundo presupuesto que expone los costes de desarrollo y mano de obra del equipo de trabajo. Finalmente, se hará un balance total.

5.1 Presupuesto de materiales y componentes

Material/Componente	Precio unitario	Cantidad	Total
Módulos HC-SR04	3,37 €	9	30,33 €
Módulo NRF24L01	1,96 €	4	7,84 €
Arduino UNO	19,95 €	3	59,85 €
Arduino MEGA	32,86 €	1	32,86 €
Raspberry Pi 3 Model B	34,14 €	1	34,14 €
Gopigo 2 Base Kit	119,99 €	1	119,99 €
Sandisk MicroSDHC 32GB	13,99 €	1	13,99 €
		TOTAL	299,00 €

Figura 6.1 Presupuesto de materiales y componentes

5.2 Costes de desarrollo

Trabajo realizado	Precio/hora	Horas totales	Total
Análisis y documentación	10,00 €	20	200,00 €
Montaje del hardware	15,00 €	15	225,00 €
Programación	25,00 €	145	3.625,00 €
Pruebas	15,00 €	40	600,00 €
Documentación	10,00 €	45	450,00 €
		TOTAL	5.100,00 €

Figura 6.2 Costes desarrollo

5.3 Balance total y presupuesto final

Concepto		Total
Material/Componentes		299,00 €
Trabajo Realizado		5.100,00 €
	TOTAL	5.399,00 €

Figura 6.3: Presupuesto final

Anexo A.

Códigos

En este anexo se expondrán cada uno de los códigos utilizados en el montaje final. Serán un total de cuatro códigos. Tres códigos para los objetos emisores A, B y C respectivamente y uno para el objeto receptor. Cada uno de los códigos estará adecuadamente comentado para una mejor comprensión del mismo.

A.1 Código del objeto emisor A

```
/* EMISOR (A)
 *  Codigo que envia un caracter para poder identificar el emisor.
 *  Esto se enviara al mismo tiempo que se envia el pulso del ultrasonido emisor
 *
 * Autor: Agustin Leon Garcia
 * Universidad de La Laguna*/

//libreria necesaria para los modulos RF

#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>

const int pinCE = 9;
const int pinCSN = 10;
RF24 radio(pinCE, pinCSN);

const int pin_SR04_Trigger = 6;                 //Pin del ultrasonido (solo
trigger)
const uint64_t pipe = 0xE8E8F0F0E1LL;           //Direcci√≥n de la conexion de
radio
double Ultimo_envio = 0;                        //Para saber el ultimo envio que
realiza
char mensaje[1];                                //Creamos un mensaje

void setup() {
  Serial.begin(9600);                           //Abre el puerto serie para la
comunicaci√≥n con el ordenador (9600 baudios)
  radio.begin();
  radio.openWritingPipe(pipe);
  radio.openReadingPipe(1, pipe);
  radio.startListening();

  pinMode(pin_SR04_Trigger, OUTPUT);            //Activamos el pin del
ultrasonido como Modo de trabajo = Salida
  Serial.println("Enviando...");
}

void loop() {

  if(radio.available()){

    radio.read(mensaje,sizeof(mensaje[1]));

    if(mensaje[0] == 'C'){                      //Emitimos cuando recibamos el
mensaje del emisor C

      Serial.println("Mensaje C Recibido");
      //delay(100);
      mensaje[0] = 'A';                         //Mensaje identificativo que
enviaremos al receptor

      digitalWrite(pin_SR04_Trigger, LOW);      //Para estabilizar el sensor
      delayMicroseconds(5);
      digitalWrite(pin_SR04_Trigger, HIGH);     //Activamos el envio del pulso
      Serial.print("Pulso enviado\n");
```

```
      radio.stopListening();
      delay(50);
      radio.write(mensaje, sizeof(mensaje[1]));
      radio.startListening();

      Ultimo_envio = millis();                  //Devuelve los milisegundos
    }
  }

  if (millis() - Ultimo_envio > 10000){

    Serial.println("Nuevo Mensaje A");
    delay(1000);
    Ultimo_envio = millis();
    mensaje[0] = 'A';                           //Mensaje identificativo que
enviaremos al receptor

    digitalWrite(pin_SR04_Trigger, LOW);        //Para estabilizar el sensor
    delayMicroseconds(5);
    digitalWrite(pin_SR04_Trigger, HIGH);       //Activamos el envio del pulso
    Serial.print("Pulso enviado\n");

    radio.stopListening();
    delay(50);
    radio.write(mensaje, sizeof(mensaje[1]));
    radio.startListening();
  }
}
```

A.2 Código del objeto emisor B

```
/* EMISOR (B)
 *  Codigo que envia un caractrer para poder identificar el emisor.
 *  Esto se enviara al mismo tiempo que se envia el pulso del ultrasonido
emisor
 *
 * Autor: Agustin Leon Garcia
 * Universidad de La Laguna */

//Libreria necesaria para los modulos RF
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>

const int pinCE = 9;
const int pinCSN = 10;
RF24 radio(pinCE, pinCSN);

const int pin_SR04_Trigger = 6;           //Pin del ultrasonido (solo
trigger)
const uint64_t pipe = 0xE8E8F0F0E1LL;     //Direcci√≥n de la conexion de
radio
char mensaje[1];                          //Creamos un mensaje


void setup() {

  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(pipe);
  radio.openReadingPipe(1, pipe);
  radio.startListening();

  pinMode(pin_SR04_Trigger, OUTPUT);      //Activamos el pin del ultrasonido
como salida
  Serial.println("Enviando...");
}

void loop() {

  if(radio.available()){

    radio.read(mensaje,sizeof(mensaje));
    //Serial.print("Se recibe: ");
    //Serial.print(mensaje[0]);
    //Serial.print("\n");

    if(mensaje[0] == 'A'){ //Emitimos cuando recibamos el mensaje del emisor
A

      Serial.print("Mensaje A Recibido\n");
      //delay(100);
      mensaje[0] = 'B'; //mensaje identificativo que enviaremos al receptor

      digitalWrite(pin_SR04_Trigger, LOW); //Para estabilizar el sensor
```

```
      delayMicroseconds(50);
      digitalWrite(pin_SR04_Trigger, HIGH); //Activamos el envio del pulso
      Serial.print("Pulso enviado\n");

      radio.stopListening();
      delayMicroseconds(500);
      radio.write(mensaje, sizeof(mensaje));
      radio.startListening();

      digitalWrite(pin_SR04_Trigger, LOW);

      Serial.print("Se envia: ");
      Serial.print(mensaje[0]);
      Serial.println("\n");
    }
  }
}
```

A.3 Código del objeto emisor C

```
/* EMISOR (C)
 *  Codigo que envia un caracter para poder identificar el emisor.
 *  Esto se enviara al mismo tiempo que se envia el pulso del
ultrasonido emisor
 *
 * Autor: Agustin Leon Garcia
 * Universidad de La Laguna*/

//Libreria necesaria para los modulos RF
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>

const int pinCE = 9;
const int pinCSN = 10;
RF24 radio(pinCE, pinCSN);

const int pin_SR04_Trigger = 6;           //Pin del ultrasonido
(solo trigger)
const uint64_t pipe = 0xE8E8F0F0E1LL;     //Direcci√≥n de la
conexion de radio
char mensaje[1];                          //Creamos un mensaje


void setup() {

  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(pipe);
  radio.openReadingPipe(1, pipe);
  radio.startListening();

  pinMode(pin_SR04_Trigger, OUTPUT);      //Activamos el pin del
ultrasonido como salida
  Serial.println("Enviando...");
}

void loop() {

  if(radio.available()){

    radio.read(mensaje,sizeof(mensaje));
    //Serial.print("Se recibe: ");
    //Serial.print(mensaje[0]);
    //Serial.print("\n");

    if(mensaje[0] == 'B'){ //Emitimos cuando recibamos el mensaje
del emisor B

      Serial.print("Mensaje B Recibido\n");
      //delay(100);
      mensaje[0] = 'C'; //mensaje identificativo que enviaremos al
receptor
```

```
      digitalWrite(pin_SR04_Trigger, LOW); //Para estabilizar el
sensor
      delayMicroseconds(50);
      digitalWrite(pin_SR04_Trigger, HIGH); //Activamos el envio
del pulso
      Serial.print("Pulso enviado\n");

      radio.stopListening();
      delayMicroseconds(500);
      radio.write(mensaje, sizeof(mensaje));
      radio.startListening();

      digitalWrite(pin_SR04_Trigger, LOW);

      //Serial.print("Se envia: ");
      //Serial.print(mensaje[0]);
      //Serial.println("\n");
      }
    }
  }
```

A.4 Código del objeto receptor

```
/* RECEPTOR
 *  Codigo que recibe los datos de los emisores para calcular la
 *  trilateración, iniciar la conexión con ROS y publicar la
odometría
 *
 * Autor: Agustin Leon Garcia
 * Universidad de La Laguna*/

//Libreria necesaria para los modulos RF
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>
#include <TimerThree.h>

#include <ros.h>
#include <ros/time.h>
#include <tf/tf.h>
#include <tf/transform_broadcaster.h>

const int pinCE = 9;                                  //Pin CE del
modulo RF
const int pinCSN = 10;                                //Pin CSN del
modulo RF
RF24 radio(pinCE, pinCSN);
//Inicializaci√≥n de la conexion de radio
const uint64_t pipe = 0xE8E8F0F0E1LL;                 //Direcci√≥n de
la conexion de radio

const int pinEntrada1 = 19;                            //Pin del
ultrasonido Echo
const int pinEntrada2 = 21;                            //Pin del
ultrasonido Echo
const int pinEntrada3 = 18;                            //Pin del
ultrasonido Echo
const int pinEntrada4 = 2;                             //Pin del
ultrasonido Echo
const int pinEntrada5 = 3;                             //Pin del
ultrasonido Echo
const int pinEntrada6 = 20;                            //Pin del
ultrasonido Echo

unsigned long mensaje[2];                              //Variable para
almacenar mensajes de radio
volatile unsigned long tiempoInt [6];                  //Variable para
almacenar tiempo de las interrupciones
float distancia[6];                                    //Variable para
almacenar distancia a cada uno de los sensores del receptor
unsigned long dif_angulo_A;
unsigned long dif_angulo_B;
unsigned long dif_angulo_C;

int i;                                                 //Variable para
inicializaci√≥n de bucle
```

```
int j;                                                  //Variable para
inicializaci√≥n de bucle

float distancia_menor_A = -1;                           //Variable para
inicializaci√≥n de distancia para los calculos
float distancia_menor_B = -1;                           //Variable para
inicializaci√≥n de distancia para los calculos
float distancia_menor_C = -1;                           //Variable para
inicializaci√≥n de distancia para los calculos


//Variables tomadas para los calculos de la trilateraci√≥n
float a;
float b;
float c;
float d;
float e;
float f;
float x1;
float y1;
float x2;
float y2;
float x3;
float y3;
float x_total;
float y_total;
float pos_1;
float pos_2;

//ROS
ros::NodeHandle nh;                                     //Creamos el
Nodo Objeto para ROS
geometry_msgs::TransformStamped t;                      //Instancia de
un mensaje para la comunicacion
tf::TransformBroadcaster broadcaster;                   //radiodifusor
para la comunicacion

char baseFrame[] = "/base_link";                        //Marcos sobre
los que se va a realizar la tranformacion
char odomFrame[] = "/odom";


void setup() {

  Serial.begin(57600);
  radio.begin();
  radio.openReadingPipe(1,pipe);
  radio.startListening();
  Serial.println("Escuchando...");

  nh.initNode();
  broadcaster.init(nh);

  pinMode(pinEntrada1, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al
  pinMode(pinEntrada2, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al
```

```
  pinMode(pinEntrada3, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al
  pinMode(pinEntrada4, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al
  pinMode(pinEntrada5, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al
  pinMode(pinEntrada6, INPUT); //activamos el pin del ultrasonido
como entrada de la se√±al

  Timer3.initialize(500000); //Preescalado para el timer
  attachInterrupt(digitalPinToInterrupt(pinEntrada1),
calcDistancia1, RISING);
  attachInterrupt(digitalPinToInterrupt(pinEntrada2),
calcDistancia2, RISING);
  attachInterrupt(digitalPinToInterrupt(pinEntrada3),
calcDistancia3, RISING);
  attachInterrupt(digitalPinToInterrupt(pinEntrada4),
calcDistancia4, RISING);
  attachInterrupt(digitalPinToInterrupt(pinEntrada5),
calcDistancia5, RISING);
  attachInterrupt(digitalPinToInterrupt(pinEntrada6),
calcDistancia6, RISING);
}

void calcDistancia1(void){
  if (tiempoInt[0] == 0)
    tiempoInt[0] = TCNT3;
}

void calcDistancia2(void){
  if (tiempoInt[1] == 0)
    tiempoInt[1] = TCNT3;
}

void calcDistancia3(void){
  if (tiempoInt[2] == 0)
    tiempoInt[2] = TCNT3;
}

void calcDistancia4(void){
  if (tiempoInt[3] == 0)
    tiempoInt[3] = TCNT3;
}

void calcDistancia5(void){
  if (tiempoInt[4] == 0)
    tiempoInt[4] = TCNT3;
}

void calcDistancia6(void){
  if (tiempoInt[5] == 0)
    tiempoInt[5] = TCNT3;
}
void loop() {

   if(radio.available()){
    radio.read(mensaje,sizeof(mensaje));
```

```
    Timer3.restart();

    for (int i = 0; i < 6; i ++){
      tiempoInt[i] = 0;
    }

    if((char)mensaje[0] == 'A'){ //Comprobamos que la se√±al
provenga del primer sensor
      delay(50);

      int indice_A;
      float menor_A = 999999;

       for (int j = 0; j < 6; j ++){
            if((tiempoInt[j] < menor_A) && (tiempoInt[j] > 0 )){
              menor_A = tiempoInt[j];
              indice_A = j +1;
            }
       }

       int angulo_A = 0;

       if ((tiempoInt[j+1] > 0) && (tiempoInt[j+1]-menor_A > 39)){

        dif_angulo_A = ((tiempoInt[(j+1)%6]) - menor_A);
        angulo_A = ((1.827*dif_angulo_A) - 71.341);
       }else{
        angulo_A = ((0.8856*dif_angulo_A) - 30.394);
       }

        if ((tiempoInt[j+1] <= 0) && (tiempoInt[j-1]-menor_A >
39)){

        dif_angulo_A = ((tiempoInt[(j-1)%6]) - menor_A);
        angulo_A = ((-1.2615*dif_angulo_A) + 28.6703);
       }else{
        angulo_A = ((-1.86428*dif_angulo_A) + 45.1659);
       }

        for (int i = 0; i < 6; i ++){

          distancia[i] = ((0.1376*(tiempoInt[i])) - 19.45);

          Serial.print("Interrupcion ");
          Serial.print(i+1);
          Serial.print(": ");
          Serial.print(tiempoInt[i]);
          Serial.print(". Distancia al emisor A: ");
          Serial.print(distancia[i]);
          Serial.print("cm.");
          Serial.print("\n");
        }

        distancia_menor_A = ((0.1376*(menor_A)) - 19.45);
        Serial.print("El sensor mas cercano es el ");
        Serial.print(indice_A);
        Serial.print(" y la distancia es de: ");
        Serial.print(distancia_menor_A);
```

```
        Serial.print(" cm\n");

        Serial.print("El angulo de giro con respecto al sensor ");
        Serial.print(indice_A);
        Serial.print(" es: ");
        Serial.print(angulo_A);
        Serial.print(" ¬ſ\n");
    }

     if((char)mensaje[0] == 'B'){ //Comprobamos que la se√±al
provenga del segundo sensor
      delay(50);
      int indice_B;
      float menor_B = 999999;

       for (int j = 0; j < 6; j ++){
            if((tiempoInt[j] < menor_B) && (tiempoInt[j] > 0 )){
              menor_B = tiempoInt[j];
              indice_B = j +1;
            }
       }

       int angulo_B = 0;

       if ((tiempoInt[j+1] > 0) && (tiempoInt[j+1]-menor_B > 39)){

        dif_angulo_B = ((tiempoInt[(j+1)%6]) - menor_B);
        angulo_B = ((1.827*dif_angulo_B) - 71.341);
       }else{
        angulo_B = ((0.8856*dif_angulo_B) - 30.394);
       }

        if ((tiempoInt[j+1] <= 0) && (tiempoInt[j-1]-menor_B >
39)){

        dif_angulo_B = ((tiempoInt[(j-1)%6]) - menor_B);
        angulo_B = ((-1.2615*dif_angulo_B) + 28.6703);
       }else{
        angulo_B = ((-1.86428*dif_angulo_B) + 45.1659);
       }

        for (int i = 0; i < 6; i ++){

          distancia[i] = ((0.1376*(tiempoInt[i])) - 19.45);

          Serial.print("Interrupcion ");
          Serial.print(i+1);
          Serial.print(": ");
          Serial.print(tiempoInt[i]);
          Serial.print(". Distancia al emisor B: ");
          Serial.print(distancia[i]);
          Serial.print("cm.");
          Serial.print("\n");
        }

        distancia_menor_B = ((0.1376*(menor_B)) - 19.45);
        Serial.print("El sensor mas cercano es el ");
        Serial.print(indice_B);
```

```
        Serial.print(" y la distancia es de: ");
        Serial.print(distancia_menor_B);
        Serial.print(" cm\n");
    }

     if((char)mensaje[0] == 'C'){ //Comprobamos que la se√±al
provenga del tercer sensor
      delay(50);
      int indice_C;
      float menor_C = 999999;

       for (int j = 0; j < 6; j ++){
            if((tiempoInt[j] < menor_C) && (tiempoInt[j] > 0 )){
              menor_C = tiempoInt[j];
              indice_C = j +1;
            }
       }

       int angulo_C = 0;

       if ((tiempoInt[j+1] > 0) && (tiempoInt[j+1]-menor_C > 39)){

        dif_angulo_C = ((tiempoInt[(j+1)%6]) - menor_C);
        angulo_C = ((1.827*dif_angulo_C) - 71.341);
       }else{
        angulo_C = ((0.8856*dif_angulo_C) - 30.394);
       }

        if ((tiempoInt[j+1] <= 0) && (tiempoInt[j-1]-menor_C >
39)){

        dif_angulo_C = ((tiempoInt[(j-1)%6]) - menor_C);
        angulo_C = ((-1.2615*dif_angulo_C) + 28.6703);
       }else{
        angulo_C = ((-1.86428*dif_angulo_C) + 45.1659);

        for (int i = 0; i < 6; i ++){

          distancia[i] = ((0.1376*(tiempoInt[i])) - 19.45);

          Serial.print("Interrupcion ");
          Serial.print(i+1);
          Serial.print(": ");
          Serial.print(tiempoInt[i]);
          Serial.print(". Distancia al emisor C: ");
          Serial.print(distancia[i]);
          Serial.print("cm.");
          Serial.print("\n");
        }
        distancia_menor_C = ((0.1376*(menor_C)) - 19.45);
        Serial.print("El sensor mas cercano es el ");
        Serial.print(indice_C);
        Serial.print(" y la distancia es de: ");
        Serial.print(distancia_menor_C);
        Serial.print(" cm\n");
    }
```

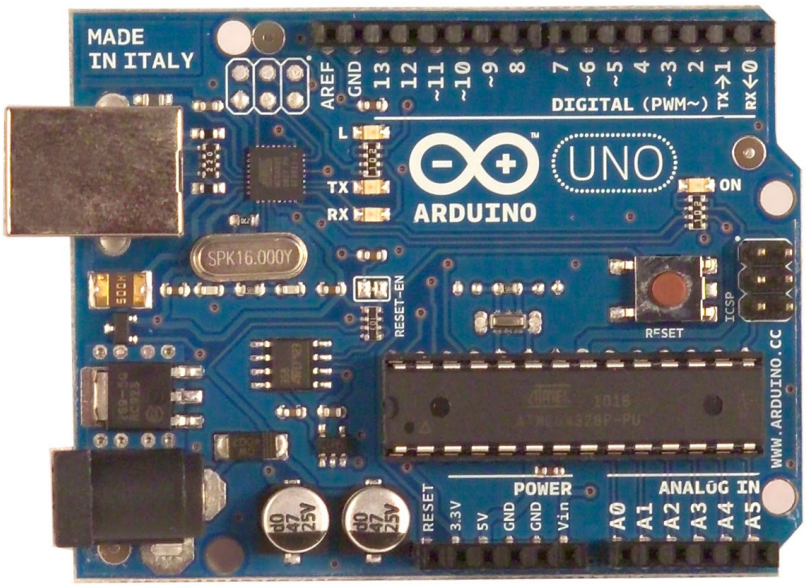
```
   Serial.println(distancia_menor_A);
   Serial.println(distancia_menor_B);
   Serial.println(distancia_menor_C);
   }
   if ((distancia_menor_A > 0) && (distancia_menor_B > 0) &&
(distancia_menor_C > 0)){

    x1 = -30;
    y1 = 0.5;
    x2 = -3;
    y2 = 46;
    x3 = 32;
    y3 = 2;

    /*Calculos de los sistemas de ecuaciones*/
    a = (-2*x1)+(2*x2);
    b = (-2*y1)+(2*y2);
    c = (distancia_menor_A*distancia_menor_A)-
(distancia_menor_B*distancia_menor_B)-(x1*x1)+(x2*x2)-
(y1*y1)+(y2*y2);
    d = (-2*x2)+(2*x3);
    e = (-2*y2)+(2*y3);
    f = (distancia_menor_B*distancia_menor_B)-
(distancia_menor_C*distancia_menor_C)-(x2*x2)+(x3*x3)-
(y2*y2)+(y3*y3);

    x_total = ((c*e)-(f*b))/((e*a)-(b*d));
    y_total = ((c*d)-(a*f))/((b*d)-(a*e));

    /*Mostramos la posici√≥n actual del robot*/
    pos_1 = x_total;
    pos_2 = y_total;

    Serial.print("Posici√≥n aproximada: ");
    Serial.print(pos_1);
    Serial.print(", ");
    Serial.println(pos_2);

    /*Apartir de aqui utilizaremos las funciones de ROS para poder
establecer una comunicacion*/
    //Rellenamos los campos de nuestra transformaci√≥n (tf odom-
>base_link)
    t.header.frame_id = "/odom";
    t.child_frame_id = "/base_link";
    t.transform.translation.x = x_total;
    t.transform.translation.y = y_total;
    t.transform.translation.z = 0.0;
    t.header.stamp = nh.now();

    //Finalmente publicamos la transformacion y esperamos un poco
antes de volver hacerlo
    broadcaster.sendTransform(t);
    nh.spinOnce();
    delay(10);
   }
}
```

Anexo B.

Documentación

En este anexo se expondrán cada uno de los datasheet de los componentes utilizados en el montaje final, así como cualquier otra documentación que haya sido de especial importancia a lo largo del desarrollo del proyecto.

Arduino UNO



Product Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduno, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Index

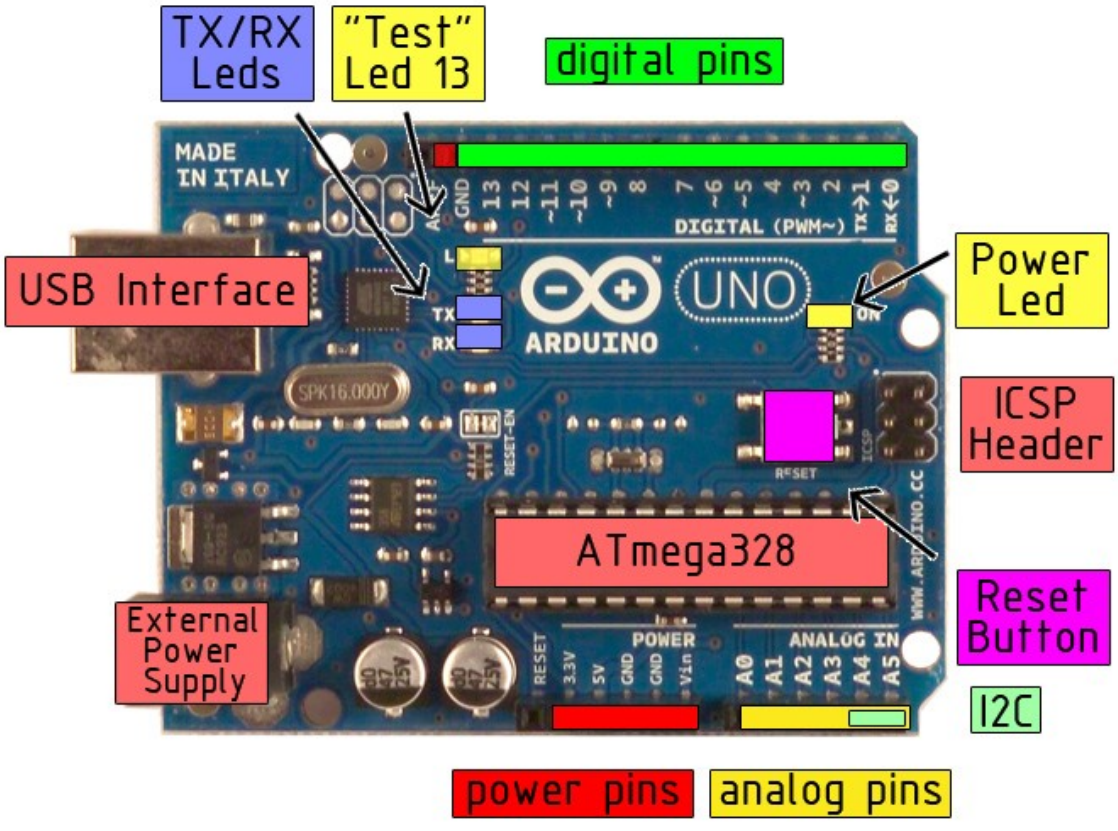
Technical Specification

EAGLE files: arduino-duemilanove-uno-design.zip Schematic: arduino-uno-schematic.pdf

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

the board



Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); It has also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. TThese pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **I²C: 4 (SDA) and 5 (SCL).** Support I²C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and Atmega328 ports.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, an *.inf file is required..

The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.

The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. To use the SPI communication, please see the ATmega328 datasheet.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno w/ ATmega328" from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

The ATmega8U2 firmware source code is available . The ATmega8U2 is loaded with a DFU bootloader, which can be activated by connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2. You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader).

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

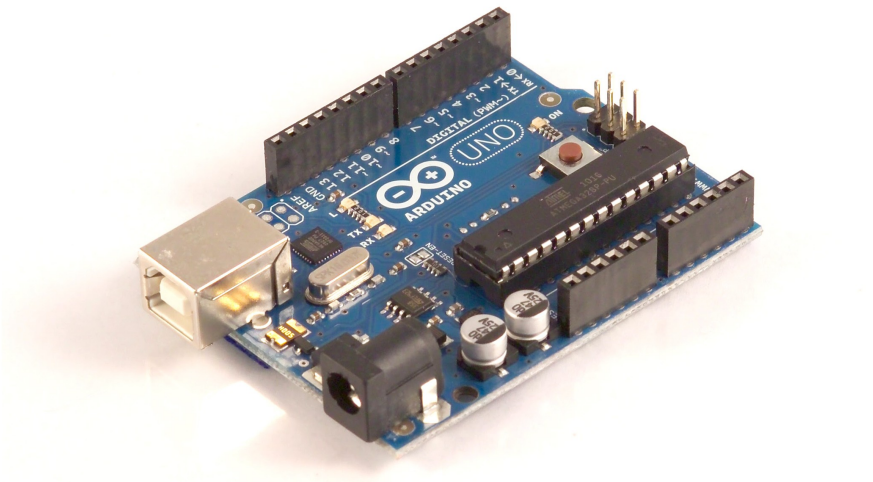
The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.



How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install Windows Install Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

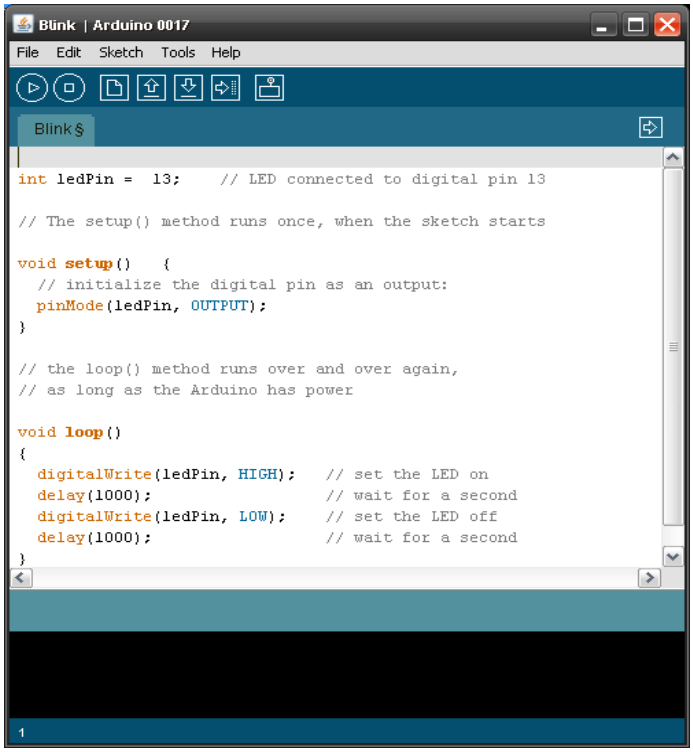
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

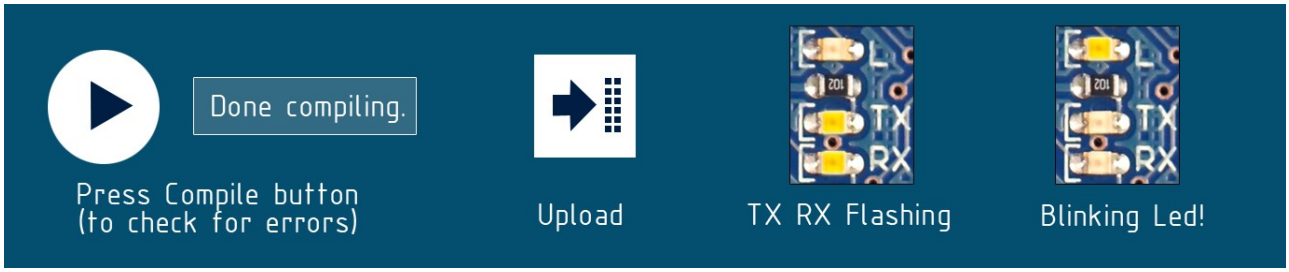
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

Once you have your skecth you'll see something very close to the screenshot on the right.

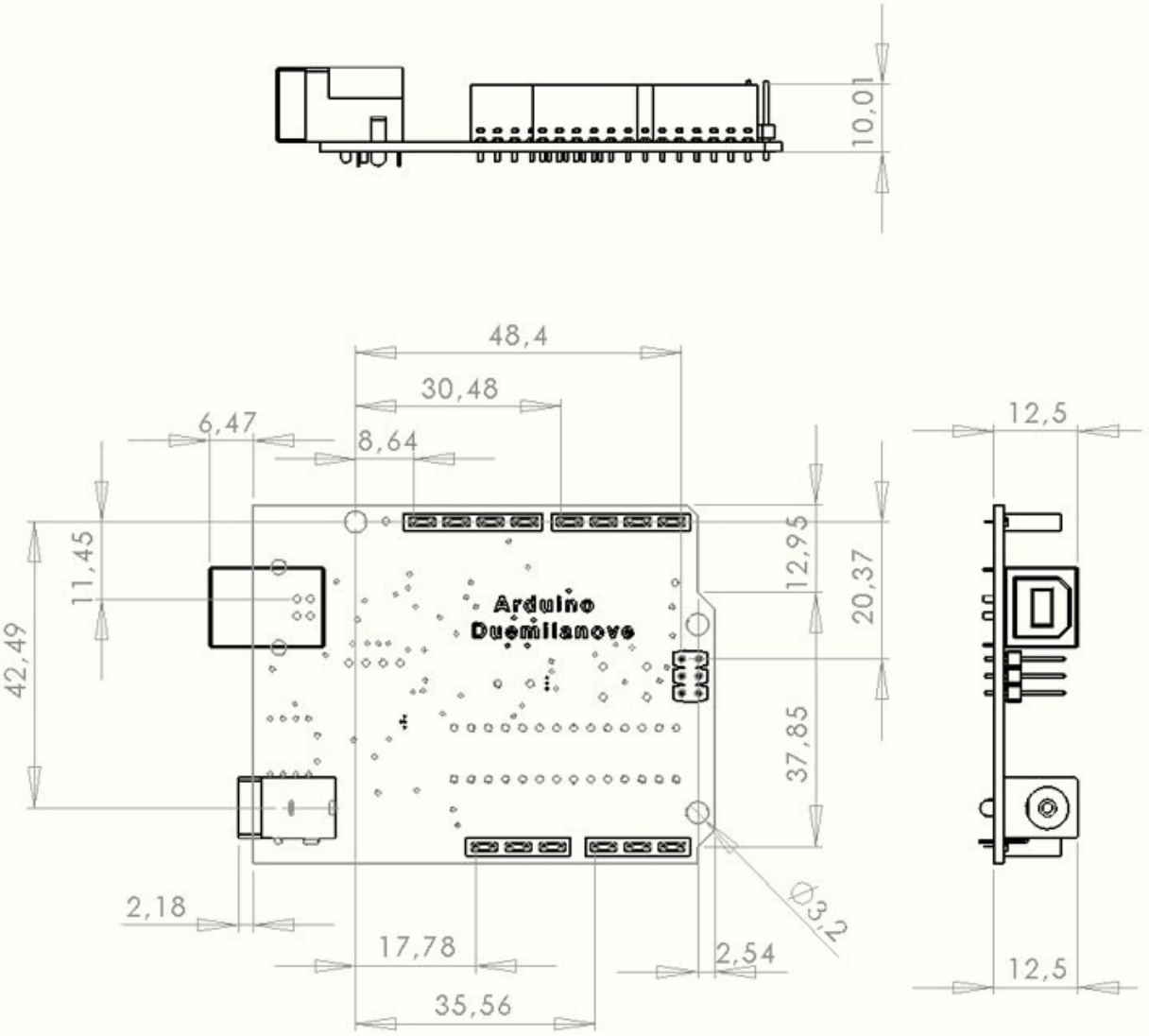
In **Tools>Board** select

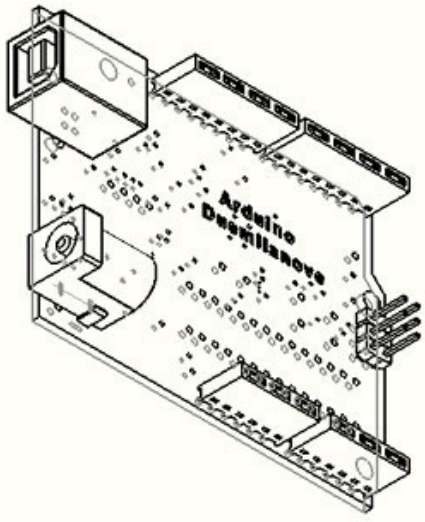
Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.





Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

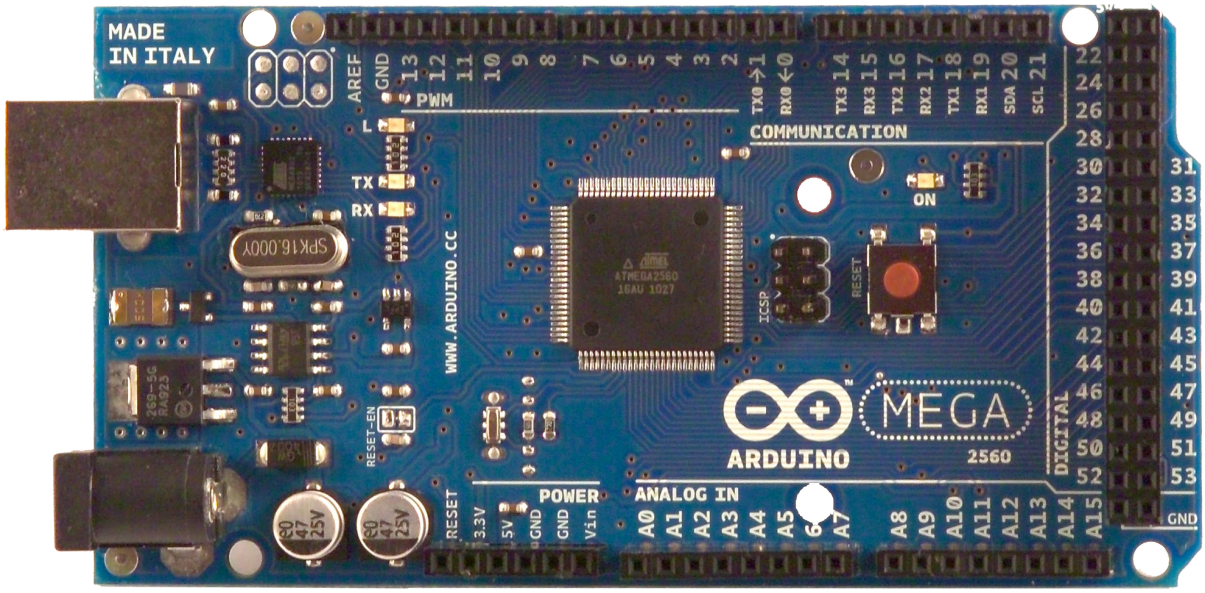
4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

Arduino MEGA 2560



Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

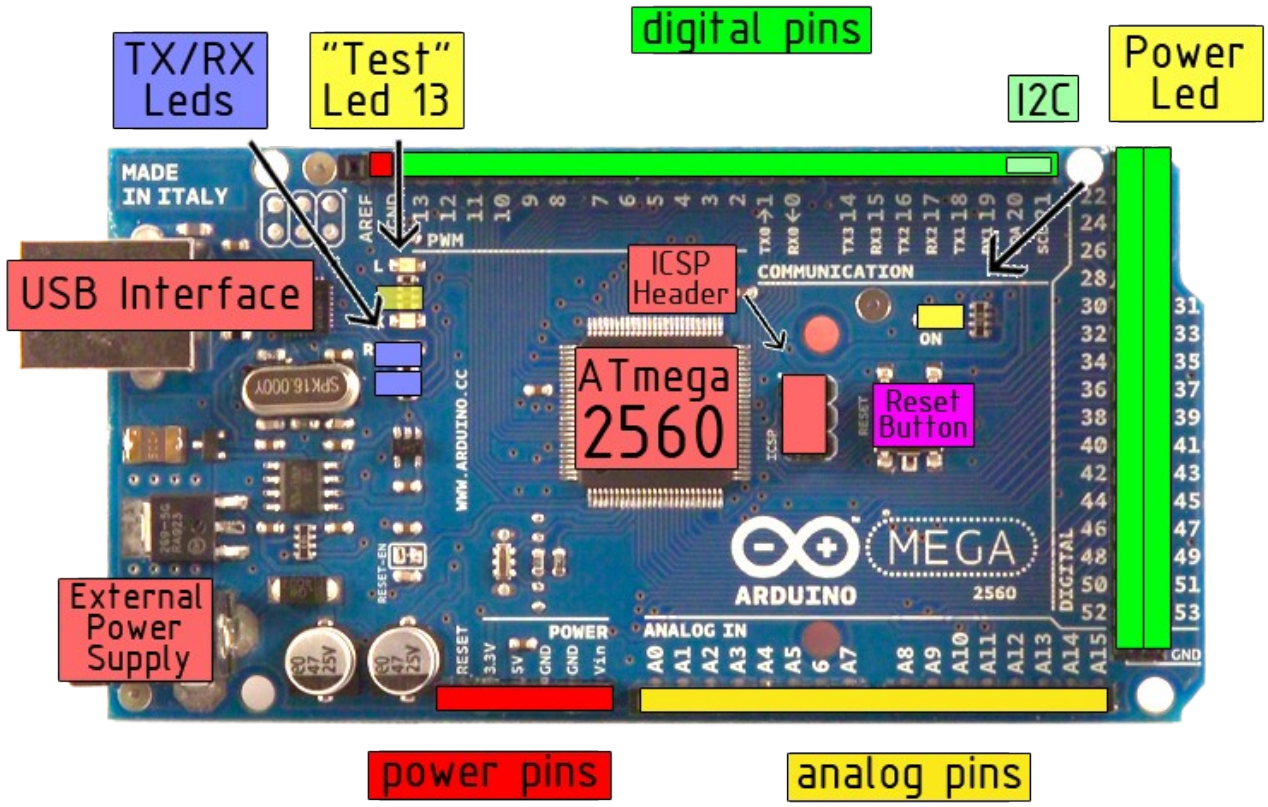
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

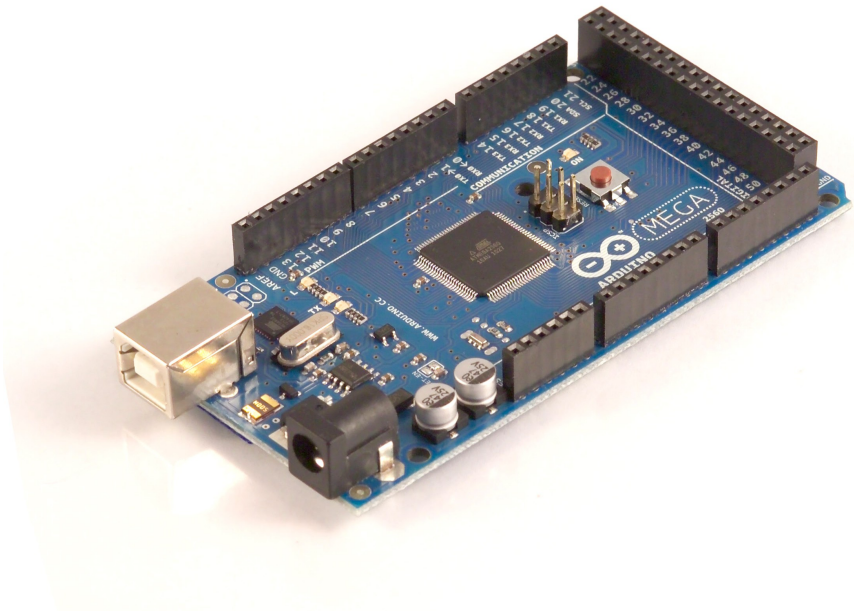
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

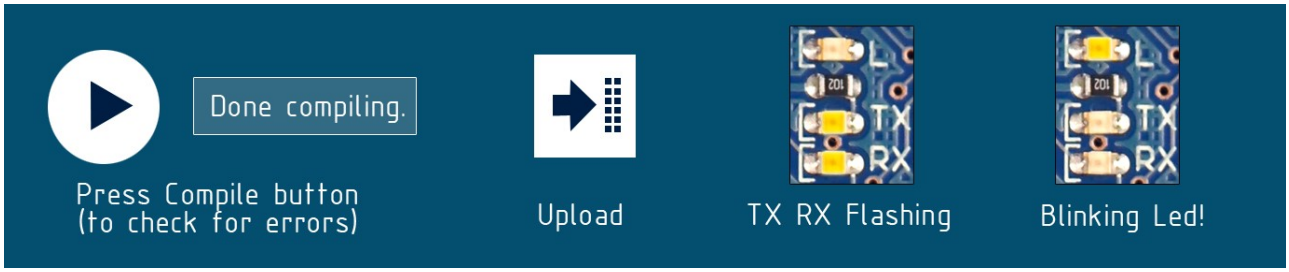
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

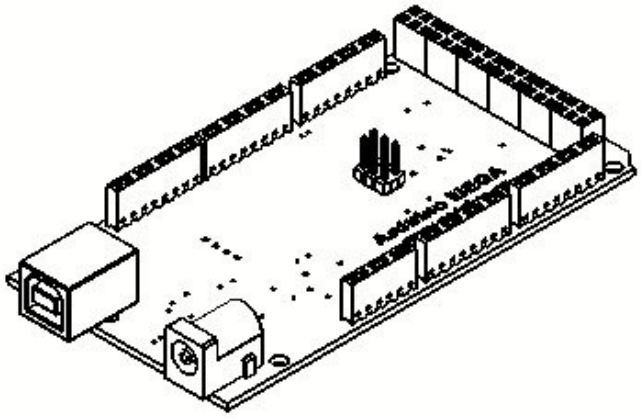
Once you have your skecth you'll see something very close to the screenshot on the right.

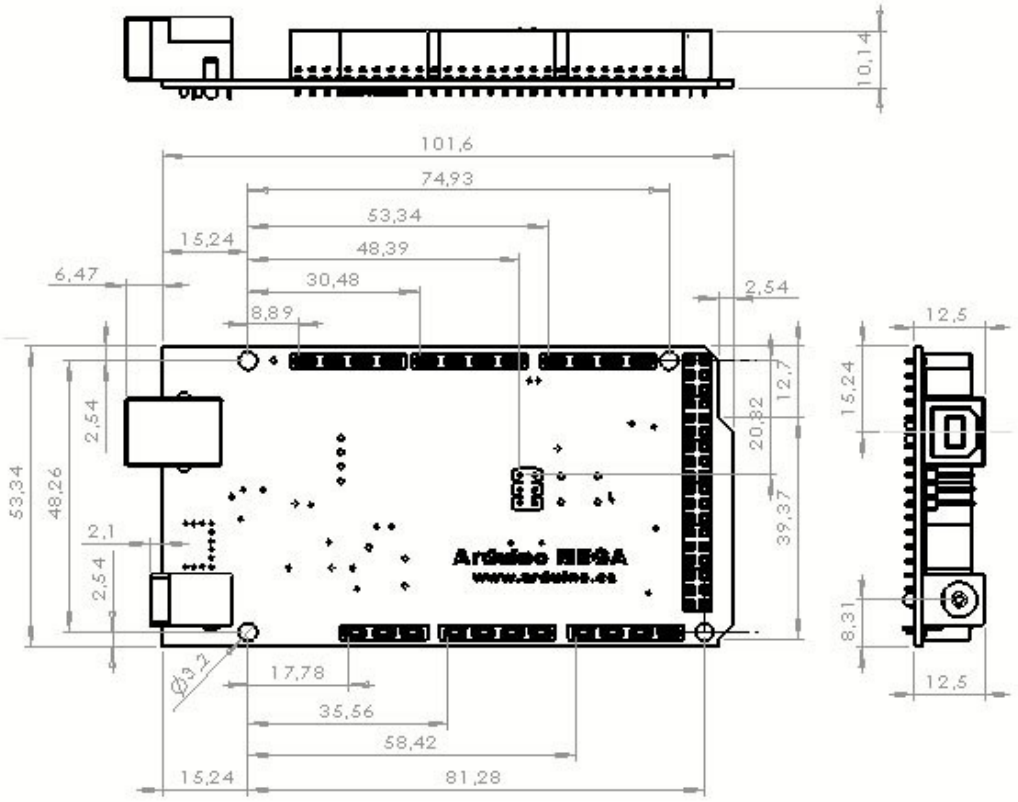
In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.



Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

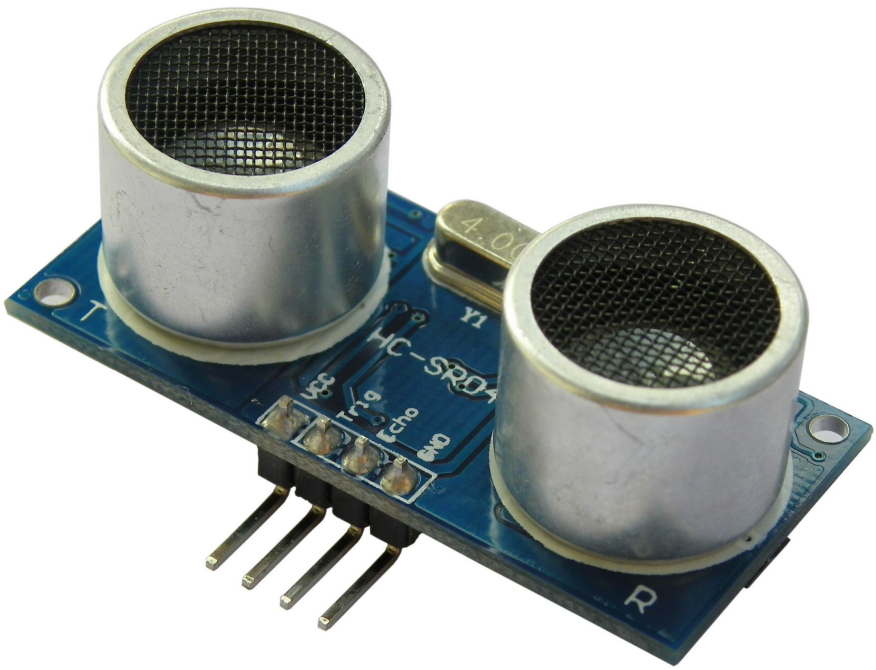
In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.



User's Manual

V1.0

May 2013

Index

1.0 INTRODUCTION

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats or dolphins do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. From 2cm to 400 cm or 1" to 13 feet. It operation is not affected by sunlight or black material like Sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with ultrasonic transmitter and receiver module.

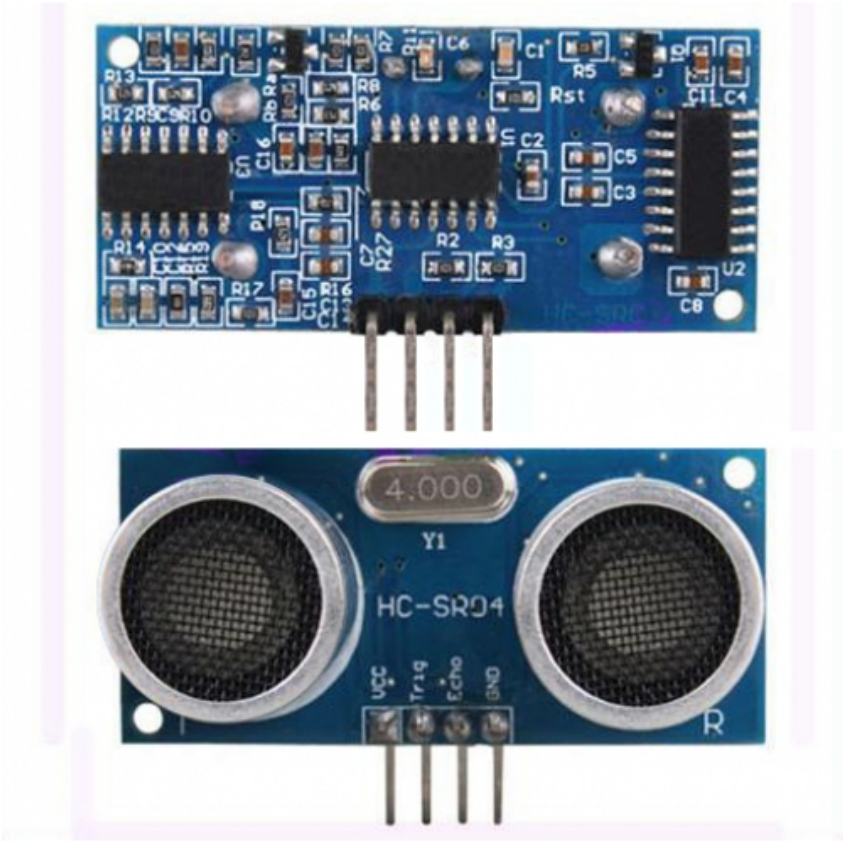
Features:

- Power Supply :+5V DC
- Quiescent Current : <2mA
- Working Currnt: 15mA
- Effectual Angle: <15°
- Ranging Distance : 2cm – 400 cm/1" - 13ft
- Resolution : 0.3 cm
- Measuring Angle: 30 degree
- Trigger Input Pulse width: 10uS
- Dimension: 45mm x 20mm x 15mm

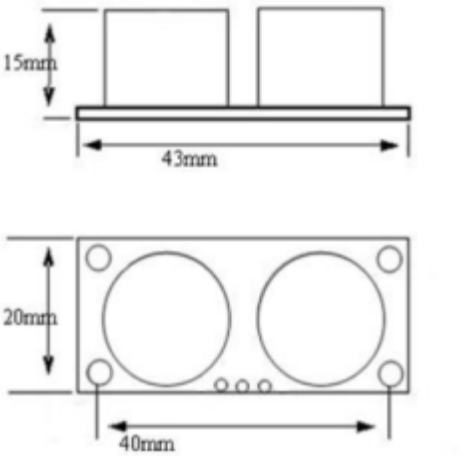
2.0 PACKING LIST

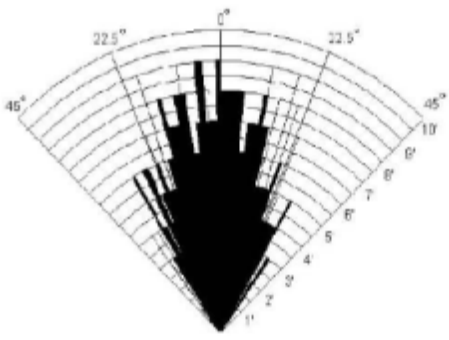
1. 1 x HC-SR04 module

3.0 PRODUCT LAYOUT



VCC = +5VDC
Trig = Trigger input of Sensor
Echo = Echo output of Sensor
GND = GND





Practical test of performance,
Best in 30 degree angle

4.0 PRODUCT SPECIFICATION AND LIMITATIONS

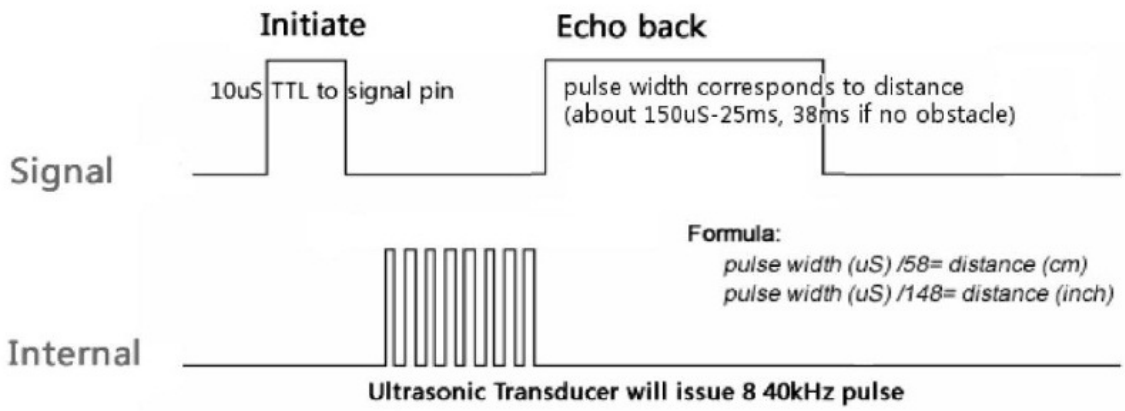
Parameter	Min	Typ.	Max	Unit
Operating Voltage	4.50	5.0	5.5	V
Quiescent Current	1.5	2	2.5	mA
Working Current	10	15	20	mA
Ultrasonic Frequency	-	40	-	kHz

5.0 OPERATION

The timing diagram of HC-SR04 is shown. To start measurement, Trig of SR04 must receive a pulse of high (5V) for at least 10us, this will initiate the sensor will transmit out 8 cycle of ultrasonic burst at 40kHz and wait for the reflected ultrasonic burst. When the sensor detected ultrasonic from receiver, it will set the Echo pin to high (5V) and delay for a period (width) which proportion to distance. To obtain the distance, measure the width (Ton) of Echo pin.

Time = Width of Echo pulse, in uS (micro second)

- Distance in centimeters = Time / 58
- Distance in inches = Time / 148
- Or you can utilize the speed of sound, which is 340m/s

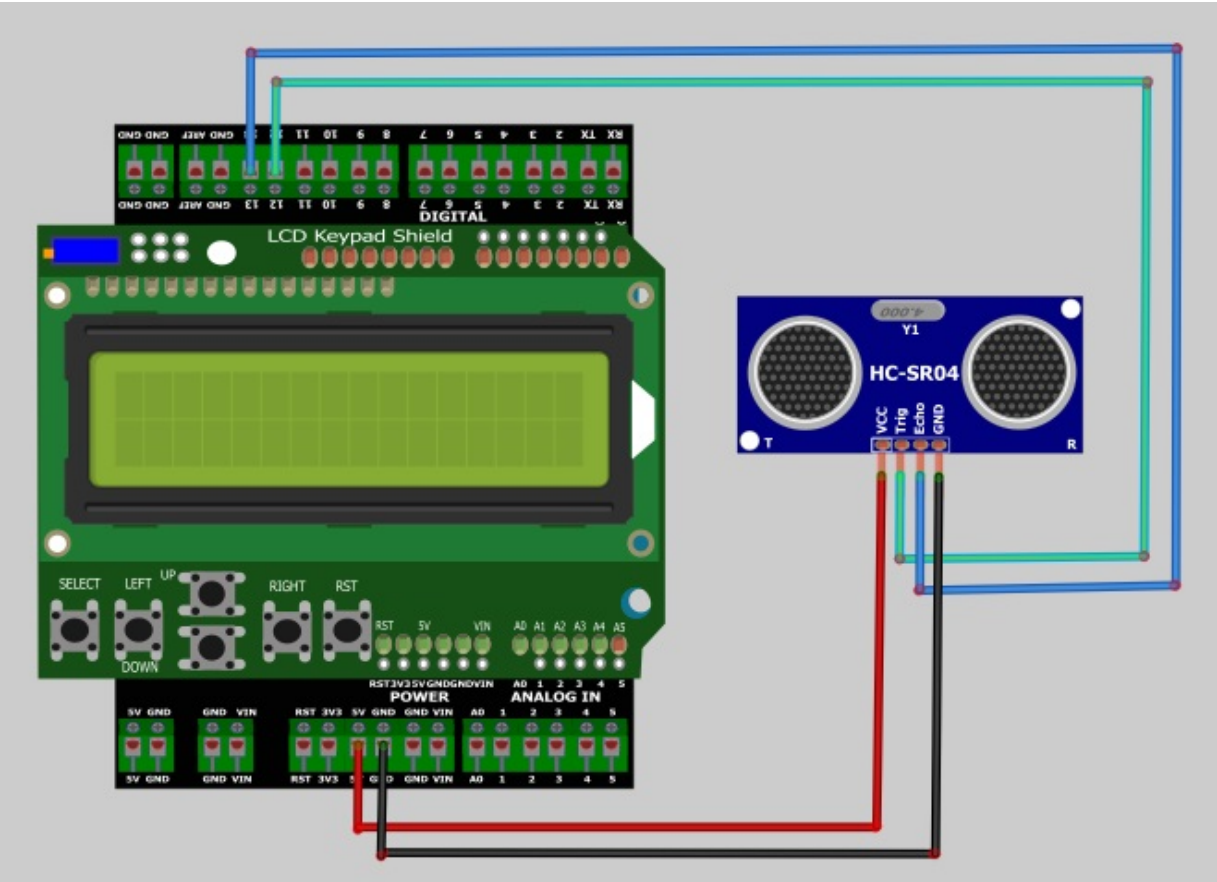


Note:

- Please connect the GND pin first before supplying power to VCC.
- Please make sure the surface of object to be detect should have at least 0.5 meter^2 better performance.

6.0 HARDWARE INTERFACE

Here is example connection for Ultrasonic Ranging module to Arduino UNO board. It can be interface with any microcontroller with digital input such as PIC, SK40C, SK28A, SKds40A, Arduino series.



7.0 EXAMPLE CODE

This is example code Ultrasonic Ranging module. Please download the complete code at the product page.

```
#include "Ultrasonic.h"
#include <LiquidCrystal.h>
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
Ultrasonic ultrasonic(12,13);

void setup() {
lcd.begin(16, 2);
lcd.setCursor(0, 0);
lcd.print("HC-SR4 testing..");
delay(1000);

}

void loop()
{
  //lcd.clear();
  lcd.setCursor(0, 1);
  lcd.print(ultrasonic.Ranging(CM));
  lcd.print("cm  ");

  delay(100);
}
```

8.0 WARRANTY

- Product warranty is valid for 6 months.
- Warranty only applies to manufacturing defect.
- Damaged caused by miss-use is not covered under warranty
- Warranty does not cover freight cost for both ways.

Prepared by
Cytron Technologies Sdn. Bhd.
19, Jalan Kebudayaan 1A,
Taman Universiti,
81300 Skudai,
Johor, Malaysia.

Tel: *+607-521 3178*
Fax: *+607-521 1861*

URL: www.cytron.com.my
Email: support@cytron.com.my
sales@cytron.com.my

nRF24L01+

Single Chip 2.4GHz Transceiver

Product Specification v1.0

Key Features

- Worldwide 2.4GHz ISM band operation
- 250kbps, 1Mbps and 2Mbps on air data rates
- Ultra low power operation
- 11.3mA TX at 0dBm output power
- 13.5mA RX at 2Mbps air data rate
- 900nA in power down
- 26µA in standby-I
- On chip voltage regulator
- 1.9 to 3.6V supply range
- Enhanced ShockBurst™
- Automatic packet handling
- Auto packet transaction handling
- 6 data pipe MultiCeiver™
- Drop-in compatibility with nRF24L01
- On-air compatible in 250kbps and 1Mbps with nRF2401A, nRF2402, nRF24E1 and nRF24E2
- Low cost BOM
- ±60ppm 16MHz crystal
- 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

Applications

- Wireless PC Peripherals
- Mouse, keyboards and remotes
- 3-in-1 desktop bundles
- Advanced Media center remote controls
- VoIP headsets
- Game controllers
- Sports watches and sensors
- RF remote controls for consumer electronics
- Home and commercial automation
- Ultra low power sensor networks
- Active RFID
- Asset tracking systems
- Toys

September 2008

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

All application information is advisory and does not form part of the specification.

Limiting values

Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the specifications are not implied. Exposure to limiting values for extended periods may affect device reliability.

Life support applications

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Data sheet status	
Objective product specification	This product specification contains target specifications for product development.
Preliminary product specification	This product specification contains preliminary data; supplementary data may be published from Nordic Semiconductor ASA later.
Product specification	This product specification contains final product specifications. Nordic Semiconductor ASA reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

Contact details

Visit www.nordicsemi.no for Nordic Semiconductor sales offices and distributors worldwide

Main office:

Otto Nielsens vei 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.no

Writing Conventions

This product specification follows a set of typographic rules that makes the document consistent and easy to read. The following writing conventions are used:

- Commands, bit state conditions, and register names are written in `Courier`.
- Pin names and pin signal conditions are written in **`Courier bold`**.
- Cross references are underlined and highlighted in blue.

Revision History

Date	Version	Description
September 2008	1.0	

Attention!

Observe precaution for handling
Electrostatic Sensitive Device.

HBM (Human Body Model) ≥ 1Kv
MM (Machine Model) ≥ 200V

Contents

1 Introduction

The nRF24L01+ is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), suitable for ultra low power wireless applications. The nRF24L01+ is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz.

To design a radio system with the nRF24L01+, you simply need an MCU (microcontroller) and a few external passive components.

You can operate and configure the nRF24L01+ through a Serial Peripheral Interface (SPI). The register map, which is accessible through the SPI, contains all configuration registers in the nRF24L01+ and is accessible in all operation modes of the chip.

The embedded baseband protocol engine (Enhanced ShockBurst™) is based on packet communication and supports various modes from manual operation to advanced autonomous protocol operation. Internal FIFOs ensure a smooth data flow between the radio front end and the system's MCU. Enhanced ShockBurst™ reduces system cost by handling all the high speed link layer operations.

The radio front end uses GFSK modulation. It has user configurable parameters like frequency channel, output power and air data rate. nRF24L01+ supports an air data rate of 250 kbps, 1 Mbps and 2Mbps. The high air data rate combined with two power saving modes make the nRF24L01+ very suitable for ultra low power designs.

nRF24L01+ is drop-in compatible with nRF24L01 and on-air compatible with nRF2401A, nRF2402, nRF24E1 and nRF24E2. Intermodulation and wideband blocking values in nRF24L01+ are much improved in comparison to the nRF24L01 and the addition of internal filtering to nRF24L01+ has improved the margins for meeting RF regulatory standards.

Internal voltage regulators ensure a high Power Supply Rejection Ratio (PSRR) and a wide power supply range.

1.1 Features

Features of the nRF24L01+ include:

- Radio
  - Worldwide 2.4GHz ISM band operation
  - 126 RF channels
  - Common RX and TX interface
  - GFSK modulation
  - 250kbps, 1 and 2Mbps air data rate
  - 1MHz non-overlapping channel spacing at 1Mbps
  - 2MHz non-overlapping channel spacing at 2Mbps
- Transmitter
  - Programmable output power: 0, -6, -12 or -18dBm
  - 11.3mA at 0dBm output power
- Receiver
  - Fast AGC for improved dynamic range
  - Integrated channel filters
  - 13.5mA at 2Mbps
  - -82dBm sensitivity at 2Mbps
  - -85dBm sensitivity at 1Mbps
  - -94dBm sensitivity at 250kbps
- RF Synthesizer
  - Fully integrated synthesizer
  - No external loop filer, VCO varactor diode or resonator
  - Accepts low cost ±60ppm 16MHz crystal
- Enhanced ShockBurst™
  - 1 to 32 bytes dynamic payload length
  - Automatic packet handling
  - Auto packet transaction handling
  - 6 data pipe MultiCeiver™ for 1:6 star networks
- Power Management
  - Integrated voltage regulator
  - 1.9 to 3.6V supply range
  - Idle modes with fast start-up times for advanced power management
  - 26µA Standby-I mode, 900nA power down mode
  - Max 1.5ms start-up from power down mode
  - Max 130us start-up from standby-I mode
- Host Interface
  - 4-pin hardware SPI
  - Max 10Mbps
  - 3 separate 32 bytes TX and RX FIFOs
  - 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

1.2 Block diagram

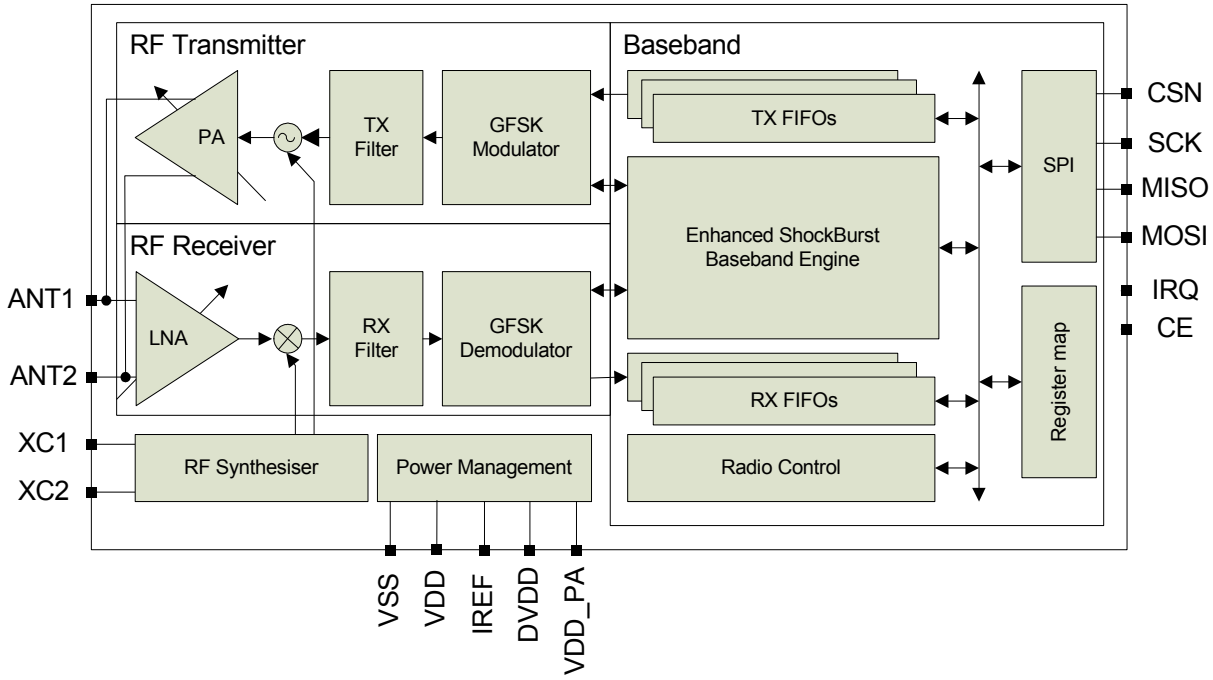


Figure 1. nRF24L01+ block diagram

2 Pin Information

2.1 Pin assignment

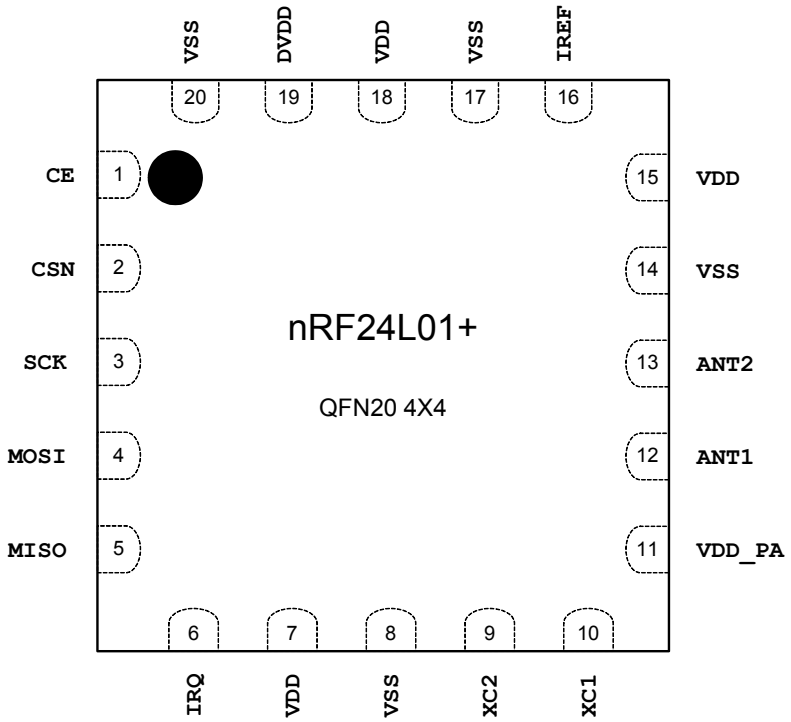


Figure 2. nRF24L01+ pin assignment (top view) for the QFN20 4x4 package

2.2 Pin functions

Pin	Name	Pin function	Description
1	CE	Digital Input	Chip Enable Activates RX or TX mode
2	CSN	Digital Input	SPI Chip Select
3	SCK	Digital Input	SPI Clock
4	MOSI	Digital Input	SPI Slave Data Input
5	MISO	Digital Output	SPI Slave Data Output, with tri-state option
6	IRQ	Digital Output	Maskable interrupt pin. Active low
7	VDD	Power	Power Supply (+1.9V - +3.6V DC)
8	VSS	Power	Ground (0V)
9	XC2	Analog Output	Crystal Pin 2
10	XC1	Analog Input	Crystal Pin 1
11	VDD_PA	Power Output	Power Supply Output (+1.8V) for the internal nRF24L01+ Power Amplifier. Must be connected to ANT1 and ANT2 as shown in Figure 32.
12	ANT1	RF	Antenna interface 1
13	ANT2	RF	Antenna interface 2
14	VSS	Power	Ground (0V)
15	VDD	Power	Power Supply (+1.9V - +3.6V DC)
16	IREF	Analog Input	Reference current. Connect a 22kΩ resistor to ground. See Figure 32.
17	VSS	Power	Ground (0V)
18	VDD	Power	Power Supply (+1.9V - +3.6V DC)
19	DVDD	Power Output	Internal digital supply output for de-coupling purposes. See Figure 32.
20	VSS	Power	Ground (0V)

Table 1. nRF24L01+ pin function

3 Absolute maximum ratings

Note: Exceeding one or more of the limiting values may cause permanent damage to nRF24L01+.

Operating conditions	Minimum	Maximum	Units
Supply voltages			
VDD	-0.3	3.6	V
VSS		0	V
Input voltage			
V_I	-0.3	5.25	V
Output voltage			
V_O	VSS to VDD	VSS to VDD	
Total Power Dissipation			
P_D (T_A=85°C)		60	mW
Temperatures			
Operating Temperature	-40	+85	°C
Storage Temperature	-40	+125	°C

Table 2. Absolute maximum ratings

4 Operating conditions

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
VDD	Supply voltage		1.9	3.0	3.6	V
VDD	Supply voltage if input signals >3.6V		2.7	3.0	3.3	V
TEMP	Operating Temperature		-40	+27	+85	ºC

Table 3. Operating conditions

5 Electrical specifications

Conditions: VDD = +3V, VSS = 0V, T_A = - 40ºC to + 85ºC

5.1 Power consumption

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	Idle modes					
I_{VDD_PD}	Supply current in power down			900		nA
I_{VDD_ST1}	Supply current in standby-I mode	a		26		µA
I_{VDD_ST2}	Supply current in standby-II mode			320		µA
I_{VDD_SU}	Average current during 1.5ms crystal oscillator startup			400		µA
	Transmit					
I_{VDD_TX0}	Supply current @ 0dBm output power	b		11.3		mA
I_{VDD_TX6}	Supply current @ -6dBm output power	b		9.0		mA
I_{VDD_TX12}	Supply current @ -12dBm output power	b		7.5		mA
I_{VDD_TX18}	Supply current @ -18dBm output power	b		7.0		mA
I_{VDD_AVG}	Average Supply current @ -6dBm output power, ShockBurst™	c		0.12		mA
I_{VDD_TXS}	Average current during TX settling	d		8.0		mA
	Receive					
I_{VDD_2M}	Supply current 2Mbps			13.5		mA
I_{VDD_1M}	Supply current 1Mbps			13.1		mA
I_{VDD_250}	Supply current 250kbps			12.6		mA
I_{VDD_RXS}	Average current during RX settling	e		8.9		mA

a. This current is for a 12pF crystal. Current when using external clock is dependent on signal swing.
b. Antenna load impedance = 15Ω+j88Ω..
c. Antenna load impedance = 15Ω+j88Ω. Average data rate 10kbps and max. payload length packets.
d. Average current consumption during TX startup (130µs) and when changing mode from RX to TX (130µs).
e. Average current consumption during RX startup (130µs) and when changing mode from TX to RX (130µs).

Table 4. Power consumption

5.2 General RF conditions

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
f_{OP}	Operating frequency	a	2400		2525	MHz
PLL_{res}	PLL Programming resolution			1		MHz
f_{XTAL}	Crystal frequency			16		MHz
Δf_{250}	Frequency deviation @ 250kbps			±160		kHz
Δf_{1M}	Frequency deviation @ 1Mbps			±160		kHz
Δf_{2M}	Frequency deviation @ 2Mbps			±320		kHz
R_{GFSK}	Air Data rate	b	250		2000	kbps
$F_{CHANNEL\ 1M}$	Non-overlapping channel spacing @ 250kbps/ 1Mbps	c		1		MHz
$F_{CHANNEL\ 2M}$	Non-overlapping channel spacing @ 2Mbps	c		2		MHz

a. Regulatory standards determine the band range you can use.
b. Data rate in each burst on-air
c. The minimum channel spacing is 1MHz

Table 5. General RF conditions

5.3 Transmitter operation

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
P_{RF}	Maximum Output Power	a		0	+4	dBm
P_{RFC}	RF Power Control Range		16	18	20	dB
P_{RFCR}	RF Power Accuracy				±4	dB
P_{BW2}	20dB Bandwidth for Modulated Carrier (2Mbps)			1800	2000	kHz
P_{BW1}	20dB Bandwidth for Modulated Carrier (1Mbps)			900	1000	kHz
P_{BW250}	20dB Bandwidth for Modulated Carrier (250kbps)			700	800	kHz
$P_{RF1.2}$	1^{st} Adjacent Channel Transmit Power 2MHz (2Mbps)				-20	dBc
$P_{RF2.2}$	2^{nd} Adjacent Channel Transmit Power 4MHz (2Mbps)				-50	dBc
$P_{RF1.1}$	1^{st} Adjacent Channel Transmit Power 1MHz (1Mbps)				-20	dBc
$P_{RF2.1}$	2^{nd} Adjacent Channel Transmit Power 2MHz (1Mbps)				-45	dBc
$P_{RF1.250}$	1^{st} Adjacent Channel Transmit Power 1MHz (250kbps)				-30	dBc
$P_{RF2.250}$	2^{nd} Adjacent Channel Transmit Power 2MHz (250kbps)				-45	dBc

a. Antenna load impedance = 15Ω+j88Ω

Table 6. Transmitter operation

5.4 Receiver operation

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	RX_{max}	Maximum received signal at <0.1% BER			0		dBm
2Mbps	RX_{SENS}	Sensitivity (0.1%BER) @2Mbps			-82		dBm
1Mbps	RX_{SENS}	Sensitivity (0.1%BER) @1Mbps			-85		dBm
250kbps	RX_{SENS}	Sensitivity (0.1%BER) @250kbps			-94		dBm

Table 7. RX Sensitivity

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	C/I_{CO}	C/I Co-channel			7		dBc
	C/I_{1ST}	1^{st} ACS (Adjacent Channel Selectivity) C/I 2MHz			3		dBc
	C/I_{2ND}	2^{nd} ACS C/I 4MHz			-17		dBc
	C/I_{3RD}	3^{rd} ACS C/I 6MHz			-21		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 12MHz			-40		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 36MHz	a		-48		dBc
1Mbps	C/I_{CO}	C/I Co-channel			9		dBc
	C/I_{1ST}	1^{st} ACS C/I 1MHz			8		dBc
	C/I_{2ND}	2^{nd} ACS C/I 2MHz			-20		dBc
	C/I_{3RD}	3^{rd} ACS C/I 3MHz			-30		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 6MHz			-40		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 25MHz	a		-47		dBc
250kbps	C/I_{CO}	C/I Co-channel			12		dBc
	C/I_{1ST}	1^{st} ACS C/I 1MHz			-12		dBc
	C/I_{2ND}	2^{nd} ACS C/I 2MHz			-33		dBc
	C/I_{3RD}	3^{rd} ACS C/I 3MHz			-38		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 6MHz			-50		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 25MHz	a		-60		dBc

a. **Narrow Band (In Band) Blocking measurements:**
0 to ±40MHz; 1MHz step size
For Interferer frequency offsets n*2*fxtal, blocking performance is degraded by approximately 5dB compared to adjacent figures.

Table 8. RX selectivity according to ETSI EN 300 440-1 V1.3.1 (2001-09) page 27

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	C/I_{CO}	C/I Co-channel (Modulated carrier)			11		dBc
	C/I_{1ST}	1^{st} ACS C/I 2MHz			4		dBc
	C/I_{2ND}	2^{nd} ACS C/I 4MHz			-18		dBc
	C/I_{3RD}	3^{rd} ACS C/I 6MHz			-24		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 12MHz			-40		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 36MHz	a		-48		dBc
1Mbps	C/I_{CO}	C/I Co-channel			12		dBc
	C/I_{1ST}	1^{st} ACS C/I 1MHz			8		dBc
	C/I_{2ND}	2^{nd} ACS C/I 2MHz			-21		dBc
	C/I_{3RD}	3^{rd} ACS C/I 3MHz			-30		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 6MHz			-40		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i > 25MHz	a		-50		dBc
250kbps	C/I_{CO}	C/I Co-channel			7		dBc
	C/I_{1ST}	1^{st} ACS C/I 1MHz			-12		dBc
	C/I_{2ND}	2^{nd} ACS C/I 2MHz			-34		dBc
	C/I_{3RD}	3^{rd} ACS C/I 3MHz			-39		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i >6MHz			-50		dBc
	C/I_{Nth}	N^{th} ACS C/I, f_i >25MHz	a		-60		dBc

a. **Narrow Band (In Band) Blocking measurements:**
0 to ±40MHz; 1MHz step size
Wide Band Blocking measurements:
30MHz to 2000MHz; 10MHz step size
2000MHz to 2399MHz; 3MHz step size
2484MHz to 3000MHz; 3MHz step size
3GHz to 12.75GHz; 25MHz step size
Wanted signal for wideband blocking measurements:
-67dBm in 1Mbps and 2Mbps mode
-77dBm in 250kbps mode
For Interferer frequency offsets n*2*fxtal, blocking performance are degraded by approximately 5dB compared to adjacent figures.
If the wanted signal is 3dB or more above the sensitivity level then, the carrier/interferer ratio is independent of the wanted signal level for a given frequency offset.

Table 9. RX selectivity with nRF24L01+ equal modulation on interfering signal. Measured using Pin = -67dBm for wanted signal.

Datarate	Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
2Mbps	P_IM(6	Input power of IM interferers at 6 and 12MHz offset from wanted signal			-42		dBm
	P_IM(8)	Input power of IM interferers at 8 and 16MHz offset from wanted signal			-38		dBm
	P_IM(10)	Input power of IM interferers at 10 and 20MHz offset from wanted signal			-37		dBm
1Mbps	P_IM(3)	Input power of IM interferers at 3 and 6MHz offset from wanted signal			-36		dBm
	P_IM(4)	Input power of IM interferers at 4 and 8MHz offset from wanted signal			-36		dBm
	P_IM(5)	Input power of IM interferers at 5 and 10MHz offset from wanted signal			-36		dBm
250kbps	P_IM(3)	Input power of IM interferers at 3 and 6MHz offset from wanted signal			-36		dBm
	P_IM(4)	Input power of IM interferers at 4 and 8MHz offset from wanted signal			-36		dBm
	P_IM(5)	Input power of IM interferers at 5 and 10MHz offset from wanted signal			-36		dBm

Note: Wanted signal level at Pin = -64 dBm. Two interferers with equal input power are used. The interferer closest in frequency is unmodulated, the other interferer is modulated equal with the wanted signal. The input power of interferers where the sensitivity equals BER = 0.1% is presented.

Table 10. RX intermodulation test performed according to Bluetooth Specification version 2.0

5.5 Crystal specifications

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
Fxo	Crystal Frequency			16		MHz
ΔF	Tolerance	a b			±60	ppm
C_0	Equivalent parallel capacitance			1.5	7.0	pF
Ls	Equivalent serial inductance	c		30		mH
C_L	Load capacitance		8	12	16	pF
ESR	Equivalent Series Resistance				100	Ω

a. Frequency accuracy including; tolerance at 25°C, temperature drift, aging and crystal loading.
b. Frequency regulations in certain regions set tighter requirements for frequency tolerance (For example, Japan and South Korea specify max. +/- 50ppm).
c. Startup time from power down to standby mode is dependant on the Ls parameter. See Table 16. on page 24 for details.

Table 11. Crystal specifications

The crystal oscillator startup time is proportional to the crystal equivalent inductance. The trend in crystal design is to reduce the physical outline. An effect of a small outline is an increase in equivalent serial inductance Ls, which gives a longer startup time. The maximum crystal oscillator startup time, Tpd2stby = 1.5 ms, is set using a crystal with equivalent serial inductance of maximum 30mH.

An application specific worst case startup time can be calculated as :

Tpd2stby= Ls/30mH *1.5ms if Ls exceeds 30mH.

Note: In some crystal datasheets Ls is called L1 or Lm and Cs is called C1 or Cm.

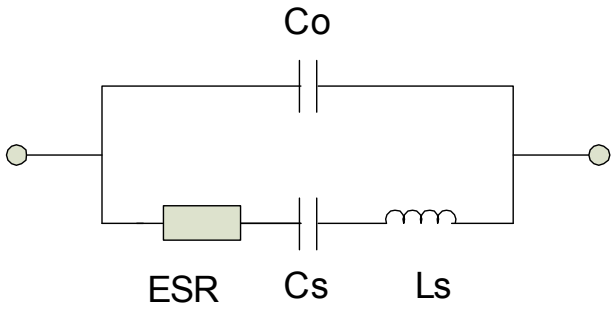


Figure 3. Equivalent crystal components

5.6 DC characteristics

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
V_{IH}	HIGH level input voltage		0.7VDD		5.25[a]	V
V_{IL}	LOW level input voltage		VSS		0.3VDD	V

a. If the input signal >3.6V, the VDD of the nRF24L01+ must be between 2.7V and 3.3V (3.0V±10%)

Table 12. Digital input pin

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
V_{OH}	HIGH level output voltage (I_{OH}=-0.25mA)		VDD -0.3		VDD	V
V_{OL}	LOW level output voltage (I_{OL}=0.25mA)				0.3	V

Table 13. Digital output pin

5.7 Power on reset

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
T_{PUP}	Power ramp up time	a			100	ms
T_{POR}	Power on reset	b	1		100	ms

a. From 0V to 1.9V.
b. Measured from when the VDD reaches 1.9V to when the reset finishes.

Table 14. Power on reset

6 Radio Control

This chapter describes the nRF24L01+ radio transceiver's operating modes and the parameters used to control the radio.

The nRF24L01+ has a built-in state machine that controls the transitions between the chip's operating modes. The state machine takes input from user defined register values and internal signals.

6.1 Operational Modes

You can configure the nRF24L01+ in power down, standby, RX or TX mode. This section describes these modes in detail.

6.1.1 State diagram

The state diagram in Figure 4. shows the operating modes and how they function. There are three types of distinct states highlighted in the state diagram:

- **Recommended operating mode:** is a recommended state used during normal operation.
- **Possible operating mode:** is a possible operating state, but is not used during normal operation.
- **Transition state:** is a time limited state used during start up of the oscillator and settling of the PLL.

When the VDD reaches 1.9V or higher nRF24L01+ enters the Power on reset state where it remains in reset until entering the Power Down mode.

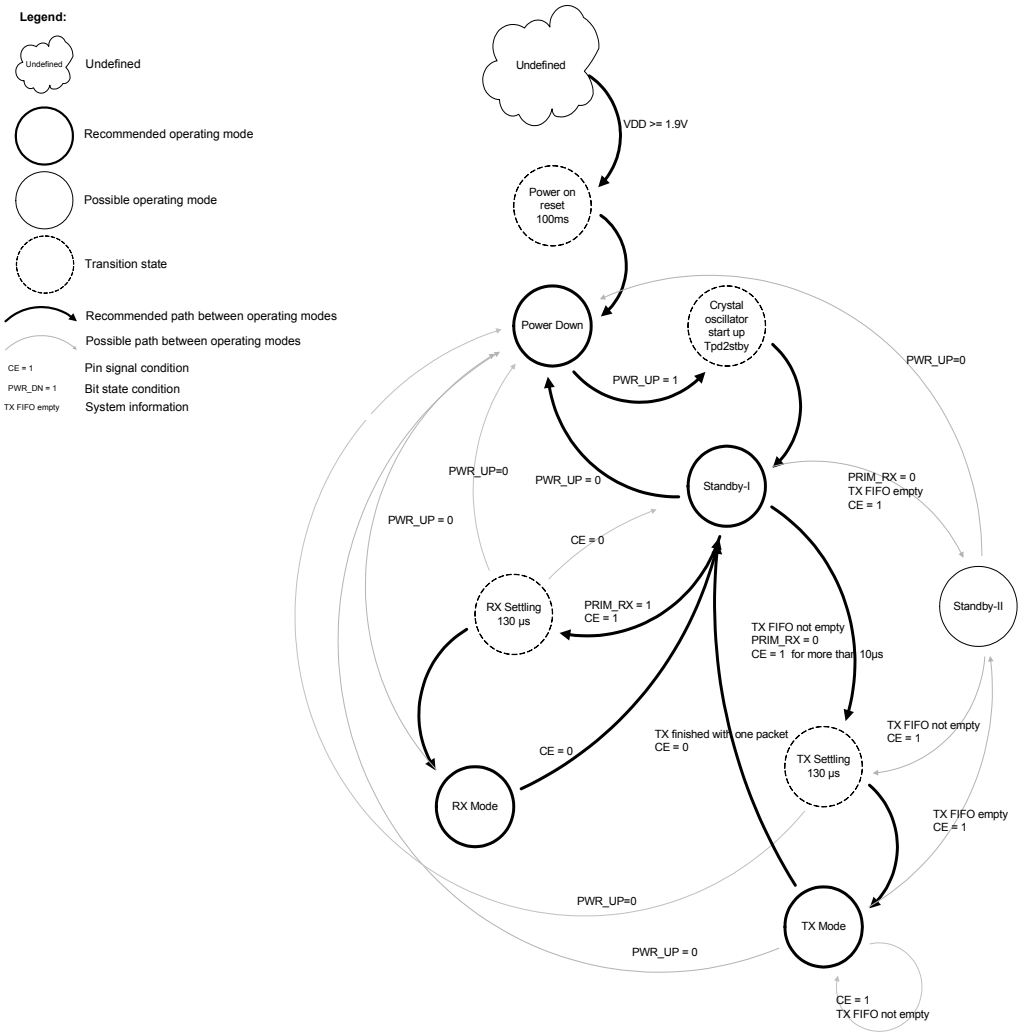


Figure 4. Radio control state diagram

6.1.2 Power Down Mode

In power down mode nRF24L01+ is disabled using minimal current consumption. All register values available are maintained and the SPI is kept active, enabling change of configuration and the uploading/downloading of data registers. For start up times see Table 16. on page 24. Power down mode is entered by setting the PWR_UP bit in the CONFIG register low.

6.1.3 Standby Modes

6.1.3.1 Standby-I mode

By setting the PWR_UP bit in the CONFIG register to 1, the device enters standby-I mode. Standby-I mode is used to minimize average current consumption while maintaining short start up times. In this mode only part of the crystal oscillator is active. Change to active modes only happens if **CE** is set high and when **CE** is set low, the nRF24L01 returns to standby-I mode from both the TX and RX modes.

6.1.3.2 Standby-II mode

In standby-II mode extra clock buffers are active and more current is used compared to standby-I mode. nRF24L01+ enters standby-II mode if **CE** is held high on a PTX device with an empty TX FIFO. If a new packet is uploaded to the TX FIFO, the PLL immediately starts and the packet is transmitted after the normal PLL settling delay (130µs).

Register values are maintained and the SPI can be activated during both standby modes. For start up times see Table 16. on page 24.

6.1.4 RX mode

The RX mode is an active mode where the nRF24L01+ radio is used as a receiver. To enter this mode, the nRF24L01+ must have the `PWR_UP` bit, `PRIM_RX` bit and the **CE** pin set high.

In RX mode the receiver demodulates the signals from the RF channel, constantly presenting the demodulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFOs. If the RX FIFOs are full, the received packet is discarded.

The nRF24L01+ remains in RX mode until the MCU configures it to standby-I mode or power down mode. However, if the automatic protocol features (Enhanced ShockBurst™) in the baseband protocol engine are enabled, the nRF24L01+ can enter other modes in order to execute the protocol.

In RX mode a Received Power Detector (RPD) signal is available. The RPD is a signal that is set high when a RF signal higher than -64 dBm is detected inside the receiving frequency channel. The internal `RPD` signal is filtered before presented to the `RPD` register. The RF signal must be present for at least 40µs before the `RPD` is set high. How to use the RPD is described in Section 6.4 on page 25.

6.1.5 TX mode

The TX mode is an active mode for transmitting packets. To enter this mode, the nRF24L01+ must have the `PWR_UP` bit set high, `PRIM_RX` bit set low, a payload in the TX FIFO and a high pulse on the **CE** for more than 10µs.

The nRF24L01+ stays in TX mode until it finishes transmitting a packet. If **CE** = 0, nRF24L01+ returns to standby-I mode. If **CE** = 1, the status of the TX FIFO determines the next action. If the TX FIFO is not empty the nRF24L01+ remains in TX mode and transmits the next packet. If the TX FIFO is empty the nRF24L01+ goes into standby-II mode. The nRF24L01+ transmitter PLL operates in open loop when in TX mode. It is important never to keep the nRF24L01+ in TX mode for more than 4ms at a time. If the Enhanced ShockBurst™ features are enabled, nRF24L01+ is never in TX mode longer than 4ms.

6.1.6 Operational modes configuration

The following table (Table 15.) describes how to configure the operational modes.

Mode	PWR_UP register	PRIM_RX register	CE input pin	FIFO state
RX mode	1	1	1	-
TX mode	1	0	1	Data in TX FIFOs. Will empty all levels in TX FIFOs[a].
TX mode	1	0	Minimum 10µs high pulse	Data in TX FIFOs.Will empty one level in TX FIFOs[b].
Standby-II	1	0	1	TX FIFO empty.
Standby-I	1	-	0	No ongoing packet transmission.
Power Down	0	-	-	-

a. If **CE** is held high all TX FIFOs are emptied and all necessary ACK and possible retransmits are carried out. The transmission continues as long as the TX FIFO is refilled. If the TX FIFO is empty when the **CE** is still high, nRF24L01+ enters standby-II mode. In this mode the transmission of a packet is started as soon as the **CSN** is set high after an upload (UL) of a packet to TX FIFO.

b. This operating mode pulses the **CE** high for at least 10µs. This allows one packet to be transmitted. This is the normal operating mode. After the packet is transmitted, the nRF24L01+ enters standby-I mode.

Table 15. nRF24L01+ main modes

6.1.7 Timing Information

The timing information in this section relates to the transitions between modes and the timing for the **CE** pin. The transition from TX mode to RX mode or vice versa is the same as the transition from the standby modes to TX mode or RX mode (max. 130µs), as described in Table 16.

Name	nRF24L01+	Notes	Max.	Min.	Comments
Tpd2stby	Power Down ➔ Standby mode		150µs		With external clock
		a	1.5ms		External crystal, Ls < 30mH
			3ms		External crystal, Ls = 60mH
			4.5ms		External crystal, Ls = 90mH
Tstby2a	Standby modes ➔ TX/RX mode		130µs		
Thce	Minimum **CE** high			10µs	
Tpece2csn	Delay from **CE** positive edge to **CSN** low			4µs	

a. See Table 11. on page 19 for crystal specifications.

Table 16. Operational timing of nRF24L01+

For nRF24L01+ to go from power down mode to TX or RX mode it must first pass through stand-by mode. There must be a delay of Tpd2stby (see Table 16.) after the nRF24L01+ leaves power down mode before the **CE** is set high.

Note: If **VDD** is turned off the register value is lost and you must configure nRF24L01+ before entering the TX or RX modes.

6.2 Air data rate

The air data rate is the modulated signaling rate the nRF24L01+ uses when transmitting and receiving data. It can be 250kbps, 1Mbps or 2Mbps. Using lower air data rate gives better receiver sensitivity than higher air data rate. But, high air data rate gives lower average current consumption and reduced probability of on-air collisions.

The air data rate is set by the `RF_DR` bit in the `RF_SETUP` register. A transmitter and a receiver must be programmed with the same air data rate to communicate with each other.

nRF24L01+ is fully compatible with nRF24L01. For compatibility with nRF2401A, nRF2402, nRF24E1, and nRF24E2 the air data rate must be set to 250kbps or 1Mbps.

6.3 RF channel frequency

The RF channel frequency determines the center of the channel used by the nRF24L01+. The channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps and a bandwidth of less than 2MHz at 2Mbps. nRF24L01+ can operate on frequencies from 2.400GHz to 2.525GHz. The programming resolution of the RF channel frequency setting is 1MHz.

At 2Mbps the channel occupies a bandwidth wider than the resolution of the RF channel frequency setting. To ensure non-overlapping channels in 2Mbps mode, the channel spacing must be 2MHz or more. At 1Mbps and 250kbps the channel bandwidth is the same or lower than the resolution of the RF frequency.

The RF channel frequency is set by the `RF_CH` register according to the following formula:

$F_0 = 2400 + RF\_CH$ *[MHz]*

You must program a transmitter and a receiver with the same RF channel frequency to communicate with each other.

6.4 Received Power Detector measurements

Received Power Detector (RPD), located in register 09, bit 0, triggers at received power levels above -64 dBm that are present in the RF channel you receive on. If the received power is less than -64 dBm, RDP = 0.

The RPD can be read out at any time while nRF24L01+ is in receive mode. This offers a snapshot of the current received power level in the channel. The RPD status is latched when a valid packet is received which then indicates signal strength from your own transmitter. If no packets are received the RPD is latched at the end of a receive period as a result of host MCU setting CE low or RX time out controlled by Enhanced ShockBurst™.

The status of RPD is correct when RX mode is enabled and after a wait time of Tstby2a +Tdelay_AGC= 130us + 40us. The RX gain varies over temperature which means that the RPD threshold also varies over temperature. The RPD threshold value is reduced by - 5dB at T = -40°C and increased by + 5dB at 85°C.

6.5 PA control

The PA (Power Amplifier) control is used to set the output power from the nRF24L01+ power amplifier. In TX mode PA control has four programmable steps, see Table 17.

The PA control is set by the RF_PWR bits in the RF_SETUP register.

SPI RF-SETUP (RF_PWR)	RF output power	DC current consumption
11	0dBm	11.3mA
10	-6dBm	9.0mA
01	-12dBm	7.5mA
00	-18dBm	7.0mA

Conditions: VDD = 3.0V, VSS = 0V, T_A = 27ºC, Load impedance = 15Ω+j88Ω.

Table 17. RF output power setting for the nRF24L01+

6.6 RX/TX control

The RX/TX control is set by PRIM_RX bit in the CONFIG register and sets the nRF24L01+ in transmit/ receive mode.

7 Enhanced ShockBurst™

Enhanced ShockBurst™ is a packet based data link layer that features automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets. Enhanced ShockBurst™ enables the implementation of ultra low power and high performance communication with low cost host microcontrollers. The Enhanced ShockBurst™ features enable significant improvements of power efficiency for bi-directional and uni-directional systems, without adding complexity on the host controller side.

7.1 Features

The main features of Enhanced ShockBurst™ are:

- 1 to 32 bytes dynamic payload length
- Automatic packet handling
- Automatic packet transaction handling
  - Auto Acknowledgement with payload
  - Auto retransmit
- 6 data pipe MultiCeiver™ for 1:6 star networks

7.2 Enhanced ShockBurst™ overview

Enhanced ShockBurst™ uses ShockBurst™ for automatic packet handling and timing. During transmit, ShockBurst™ assembles the packet and clocks the bits in the data packet for transmission. During receive, ShockBurst™ constantly searches for a valid address in the demodulated signal. When ShockBurst™ finds a valid address, it processes the rest of the packet and validates it by CRC. If the packet is valid the payload is moved into a vacant slot in the RX FIFOs. All high speed bit handling and timing is controlled by ShockBurst™.

Enhanced ShockBurst™ features automatic packet transaction handling for the easy implementation of a reliable bi-directional data link. An Enhanced ShockBurst™ packet transaction is a packet exchange between two transceivers, with one transceiver acting as the Primary Receiver (PRX) and the other transceiver acting as the Primary Transmitter (PTX). An Enhanced ShockBurst™ packet transaction is always initiated by a packet transmission from the PTX, the transaction is complete when the PTX has received an acknowledgment packet (ACK packet) from the PRX. The PRX can attach user data to the ACK packet enabling a bi-directional data link.

The automatic packet transaction handling works as follows:

1. You begin the transaction by transmitting a data packet from the PTX to the PRX. Enhanced ShockBurst™ automatically sets the PTX in receive mode to wait for the ACK packet.
2. If the packet is received by the PRX, Enhanced ShockBurst™ automatically assembles and transmits an acknowledgment packet (ACK packet) to the PTX before returning to receive mode.
3. If the PTX does not receive the ACK packet immediately, Enhanced ShockBurst™ automatically retransmits the original data packet after a programmable delay and sets the PTX in receive mode to wait for the ACK packet.

In Enhanced ShockBurst™ it is possible to configure parameters such as the maximum number of retransmits and the delay from one transmission to the next retransmission. All automatic handling is done without the involvement of the MCU.

7.3 Enhanced Shockburst™ packet format

The format of the Enhanced ShockBurst™ packet is described in this section. The Enhanced Shock-Burst™ packet contains a preamble, address, packet control, payload and CRC field. Figure 5. shows the packet format with MSB to the left.

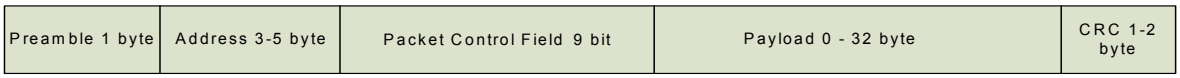


Figure 5. An Enhanced ShockBurst™ packet with payload (0-32 bytes)

7.3.1 Preamble

The preamble is a bit sequence used to synchronize the receivers demodulator to the incoming bit stream. The preamble is one byte long and is either 01010101 or 10101010. If the first bit in the address is 1 the preamble is automatically set to 10101010 and if the first bit is 0 the preamble is automatically set to 01010101. This is done to ensure there are enough transitions in the preamble to stabilize the receiver.

7.3.2 Address

This is the address for the receiver. An address ensures that the packet is detected and received by the correct receiver, preventing accidental cross talk between multiple nRF24L01+ systems. You can configure the address field width in the `AW` register to be 3, 4 or 5 bytes, see Table 28. on page 63.

Note: Addresses where the level shifts only one time (that is, 000FFFFFFF) can often be detected in noise and can give a false detection, which may give a raised Packet Error Rate. Addresses as a continuation of the preamble (hi-low toggling) also raises the Packet Error Rate.

7.3.3 Packet control field

Figure 6. shows the format of the 9 bit packet control field, MSB to the left.

Figure 6. Packet control field

The packet control field contains a 6 bit payload length field, a 2 bit PID (Packet Identity) field and a 1 bit `NO_ACK` flag.

7.3.3.1 Payload length

This 6 bit field specifies the length of the payload in bytes. The length of the payload can be from 0 to 32 bytes.

Coding: 000000 = 0 byte (only used in empty ACK packets.) 100000 = 32 byte, 100001 = Don't care.

This field is only used if the Dynamic Payload Length function is enabled.

7.3.3.2 PID (Packet identification)

The 2 bit PID field is used to detect if the received packet is new or retransmitted. PID prevents the PRX device from presenting the same payload more than once to the receiving host MCU. The PID field is incremented at the TX side for each new packet received through the SPI. The PID and CRC fields (see section 7.3.5 on page 30) are used by the PRX device to determine if a packet is retransmitted or new. When several data packets are lost on the link, the PID fields may become equal to the last received PID. If a packet has the same PID as the previous packet, nRF24L01+ compares the CRC sums from both packets. If the CRC sums are also equal, the last received packet is considered a copy of the previously received packet and discarded.

7.3.3.3 No Acknowledgment flag (NO_ACK)

The Selective Auto Acknowledgement feature controls the NO_ACK flag.

This flag is only used when the auto acknowledgement feature is used. Setting the flag high tells the receiver that the packet is not to be auto acknowledged.

On the PTX you can set the NO_ACK flag bit in the Packet Control Field with this command:

W_TX_PAYLOAD_NOACK

However, the function must first be enabled in the FEATURE register by setting the EN_DYN_ACK bit. When you use this option the PTX goes directly to standby-I mode after transmitting the packet. The PRX does not transmit an ACK packet when it receives the packet.

7.3.4 Payload

The payload is the user defined content of the packet. It can be 0 to 32 bytes wide and is transmitted on-air when it is uploaded to nRF24L01+.

Enhanced ShockBurst™ provides two alternatives for handling payload lengths; static and dynamic.

The default is static payload length. With static payload length all packets between a transmitter and a receiver have the same length. Static payload length is set by the RX_PW_Px registers on the receiver side. The payload length on the transmitter side is set by the number of bytes clocked into the TX_FIFO and must equal the value in the RX_PW_Px register on the receiver side.

Dynamic Payload Length (DPL) is an alternative to static payload length. DPL enables the transmitter to send packets with variable payload length to the receiver. This means that for a system with different payload lengths it is not necessary to scale the packet length to the longest payload.

With the DPL feature the nRF24L01+ can decode the payload length of the received packet automatically instead of using the RX_PW_Px registers. The MCU can read the length of the received payload by using the R_RX_PL_WID command.

Note: Always check if the packet width reported is 32 bytes or shorter when using the R_RX_PL_WID command. If its width is longer than 32 bytes then the packet contains errors and must be discarded. Discard the packet by using the Flush_RX command.

In order to enable DPL the EN_DPL bit in the FEATURE register must be enabled. In RX mode the DYNPD register must be set. A PTX that transmits to a PRX with DPL enabled must have the DPL_P0 bit in DYNPD set.

7.3.5 CRC (Cyclic Redundancy Check)

The CRC is the mandatory error detection mechanism in the packet. It is either 1 or 2 bytes and is calculated over the address, Packet Control Field and Payload.

The polynomial for 1 byte CRC is $X^8 + X^2 + X + 1$. Initial value 0xFF.

The polynomial for 2 byte CRC is $X^{16} + X^{12} + X^5 + 1$. Initial value 0xFFFF.

The number of bytes in the CRC is set by the CRCO bit in the CONFIG register. No packet is accepted by Enhanced ShockBurst™ if the CRC fails.

7.3.6 Automatic packet assembly

The automatic packet assembly assembles the preamble, address, packet control field, payload and CRC to make a complete packet before it is transmitted.

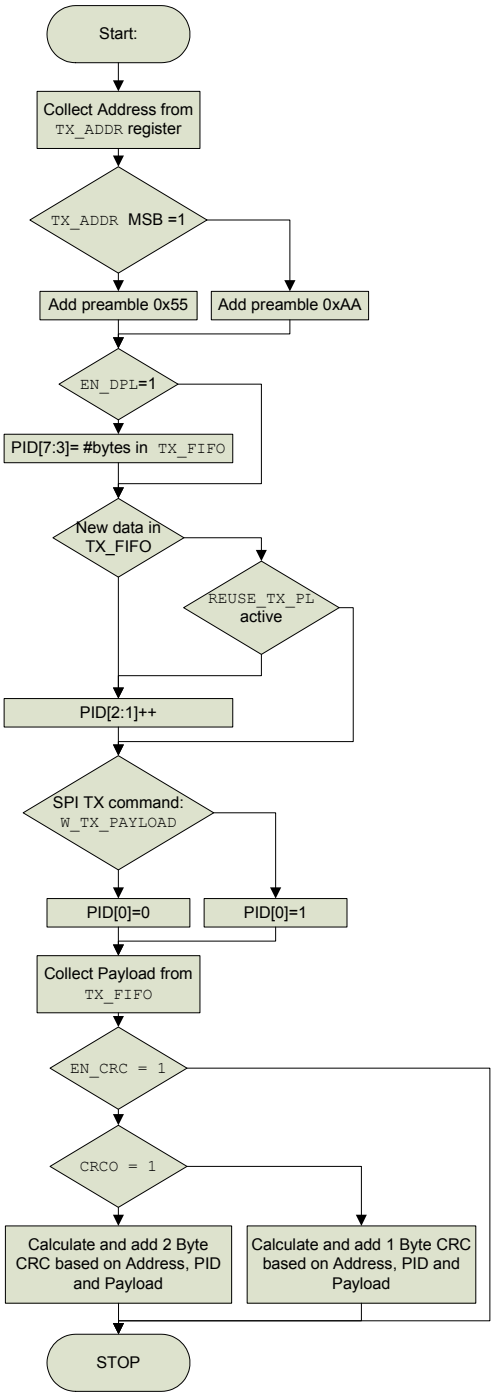


Figure 7. Automatic packet assembly

7.3.7 Automatic packet disassembly

After the packet is validated, Enhanced ShockBurst™ disassembles the packet and loads the payload into the RX FIFO, and asserts the RX_DR IRQ.

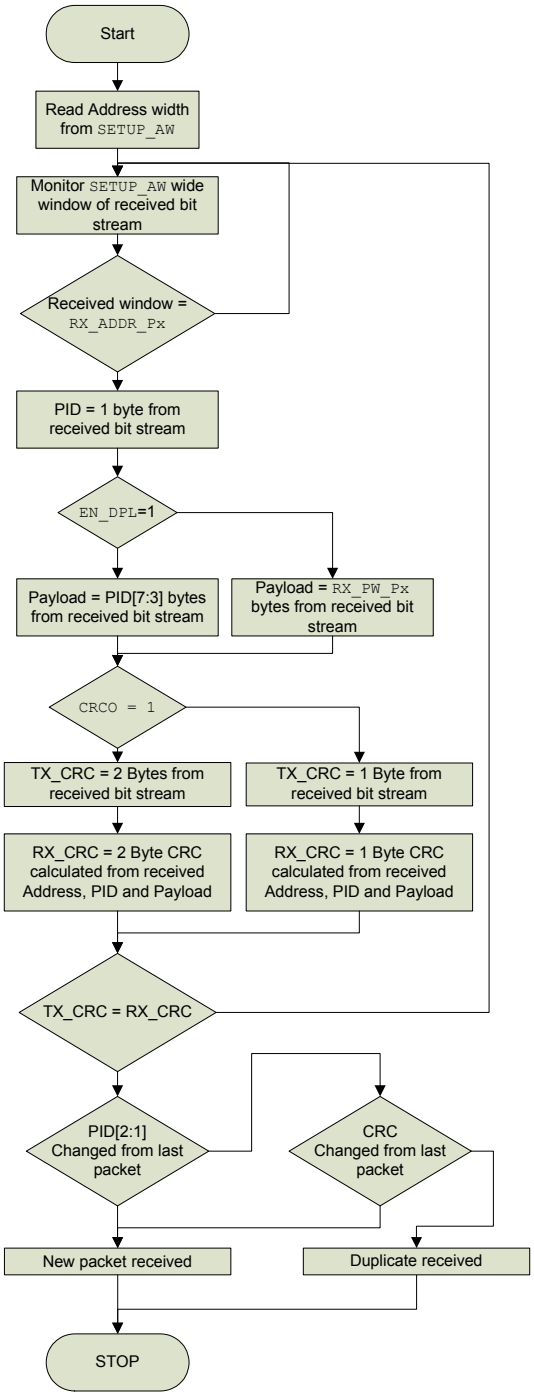


Figure 8. Automatic packet disassembly

7.4 Automatic packet transaction handling

Enhanced ShockBurst™ has two functions for automatic packet transaction handling; auto acknowledgement and auto re-transmit.

7.4.1 Auto acknowledgement

Auto acknowledgment is a function that automatically transmits an ACK packet to the PTX after it has received and validated a packet. The auto acknowledgement function reduces the load of the system MCU and can remove the need for dedicated SPI hardware. This also reduces cost and average current consumption. The Auto Acknowledgement feature is enabled by setting the `EN_AA` register.

Note: If the received packet has the `NO_ACK` flag set, auto acknowledgement is not executed.

An ACK packet can contain an optional payload from PRX to PTX. In order to use this feature, the Dynamic Payload Length (DPL) feature must be enabled. The MCU on the PRX side has to upload the payload by clocking it into the TX FIFO by using the `W_ACK_PAYLOAD` command. The payload is pending in the TX FIFO (PRX) until a new packet is received from the PTX. nRF24L01+ can have three ACK packet payloads pending in the TX FIFO (PRX) at the same time.

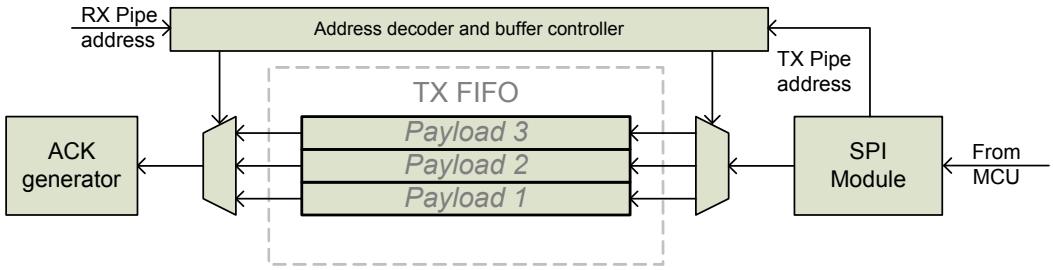


Figure 9. TX FIFO (PRX) with pending payloads

Figure 9. shows how the TX FIFO (PRX) is operated when handling pending ACK packet payloads. From the MCU the payload is clocked in with the `W_ACK_PAYLOAD` command. The address decoder and buffer controller ensure that the payload is stored in a vacant slot in the TX FIFO (PRX). When a packet is received, the address decoder and buffer controller are notified with the PTX address. This ensures that the right payload is presented to the ACK generator.

If the TX FIFO (PRX) contains more than one payload to a PTX, payloads are handled using the first in – first out principle. The TX FIFO (PRX) is blocked if all pending payloads are addressed to a PTX where the link is lost. In this case, the MCU can flush the TX FIFO (PRX) by using the `FLUSH_TX` command.

In order to enable Auto Acknowledgement with payload the `EN_ACK_PAY` bit in the `FEATURE` register must be set.

7.4.2 Auto Retransmission (ART)

The auto retransmission is a function that retransmits a packet if an ACK packet is not received. It is used in an auto acknowledgement system on the PTX. When a packet is not acknowledged, you can set the number of times it is allowed to retransmit by setting the ARC bits in the `SETUP_RETR` register. PTX enters RX mode and waits a short period for an ACK packet each time a packet is transmitted. The time period the PTX is in RX mode is based on the following conditions:

- Auto Retransmit Delay (ARD) has elapsed.
- No address match within 250µs (or 500µs in 250kbps mode).
- After received packet (CRC correct or not).

nRF24L01+ asserts the TX_DS IRQ when the ACK packet is received.

nRF24L01+ enters standby-I mode if there is no more untransmitted data in the TX FIFO and the CE pin is low. If the ACK packet is not received, nRF24L01+ goes back to TX mode after a delay defined by ARD and retransmits the data. This continues until acknowledgment is received, or the maximum number of retransmits is reached.

Two packet loss counters are incremented each time a packet is lost, ARC_CNT and PLOS_CNT in the OBSERVE_TX register. The ARC_CNT counts the number of retransmissions for the current transaction. You reset ARC_CNT by initiating a new transaction. The PLOS_CNT counts the total number of retransmissions since the last channel change. You reset PLOS_CNT by writing to the RF_CH register. It is possible to use the information in the OBSERVE_TX register to make an overall assessment of the channel quality.

The ARD defines the time from the end of a transmitted packet to when a retransmit starts on the PTX. ARD is set in SETUP_RETR register in steps of 250µs. A retransmit is made if no ACK packet is received by the PTX.

There is a restriction on the length of ARD when using ACK packets with payload. The ARD time must never be shorter than the sum of the startup time and the time on-air for the ACK packet:

- For 2Mbps data rate and 5 byte address; 15 byte is maximum ACK packet payload length for ARD=250µs (reset value).
- For 1Mbps data rate and 5 byte address; 5 byte is maximum ACK packet payload length for ARD=250µs (reset value).

ARD=500µs is long enough for any ACK payload length in 1 or 2Mbps mode.

- For 250kbps data rate and 5byte address the following values apply:

ARD	ACK packet size (in bytes)
1500µs	All ACK payload sizes
1250µs	≤ 24
1000µs	≤ 16
750µs	≤ 8
500µs	Empty ACK with no payload

Table 18. Maximum ACK payload length for different retransmit delays at 250kbps

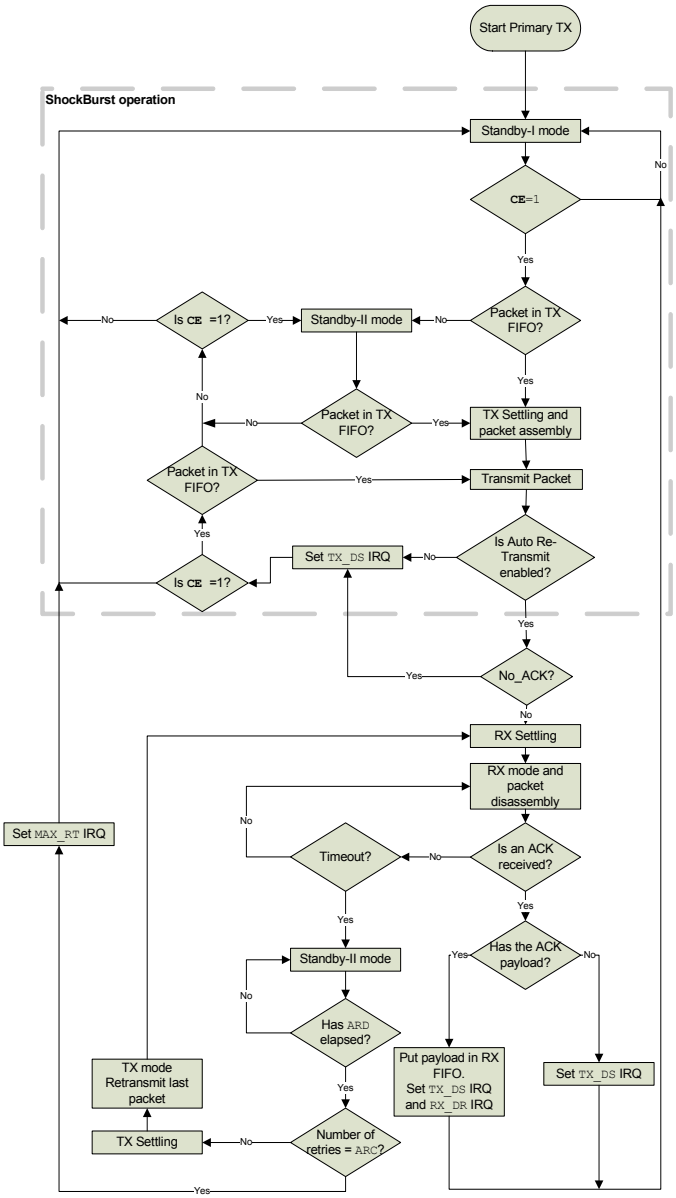
As an alternative to Auto Retransmit it is possible to manually set the nRF24L01+ to retransmit a packet a number of times. This is done by the REUSE_TX_PL command. The MCU must initiate each transmission of the packet with a pulse on the CE pin when this command is used.

7.5 Enhanced ShockBurst flowcharts

This section contains flowcharts outlining PTX and PRX operation in Enhanced ShockBurst™.

7.5.1 PTX operation

The flowchart in Figure 10. outlines how a nRF24L01+ configured as a PTX behaves after entering standby-I mode.



Note: ShockBurst™ operation is outlined with a dashed square.

Figure 10. PTX operations in Enhanced ShockBurst™

Activate PTX mode by setting the **CE** pin high. If there is a packet present in the TX FIFO the nRF24L01+ enters TX mode and transmits the packet. If Auto Retransmit is enabled, the state machine checks if the NO_ACK flag is set. If it is not set, the nRF24L01+ enters RX mode to receive an ACK packet. If the received ACK packet is empty, only the TX_DS IRQ is asserted. If the ACK packet contains a payload, both TX_DS IRQ and RX_DR IRQ are asserted simultaneously before nRF24L01+ returns to standby-I mode.

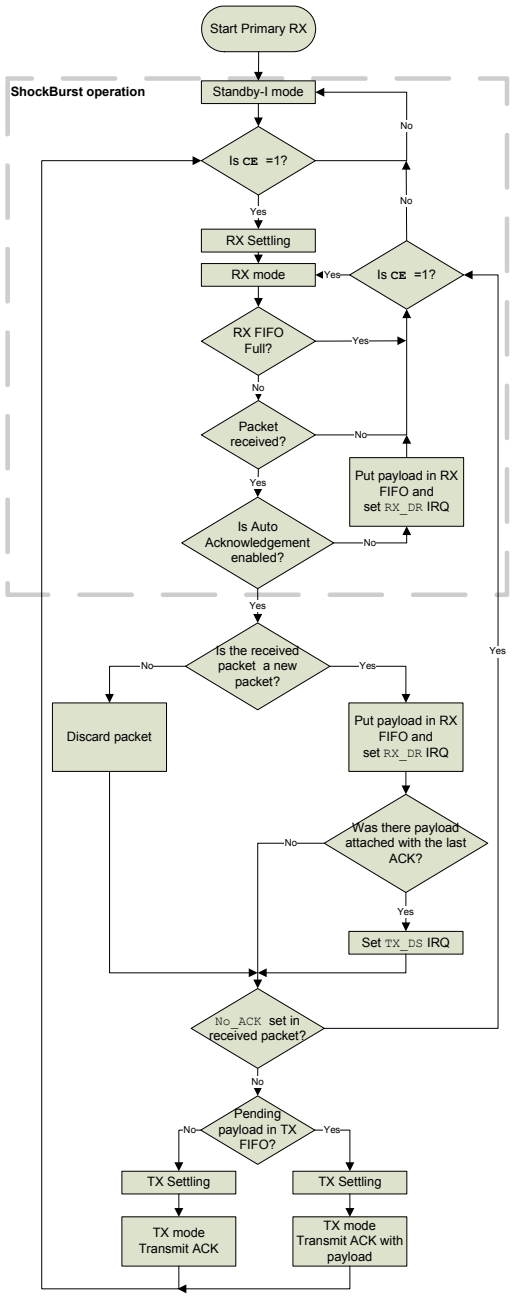
If the ACK packet is not received before timeout occurs, the nRF24L01+ returns to standby-II mode. It stays in standby-II mode until the ARD has elapsed. If the number of retransmits has not reached the ARC, the nRF24L01+ enters TX mode and transmits the last packet once more.

While executing the Auto Retransmit feature, the number of retransmits can reach the maximum number defined in ARC. If this happens, the nRF24L01+ asserts the MAX_RT IRQ and returns to standby-I mode.

If the **CE** is high and the TX FIFO is empty, the nRF24L01+ enters Standby-II mode.

7.5.2 PRX operation

The flowchart in Figure 11. outlines how a nRF24L01+ configured as a PRX behaves after entering standby-I mode.



Note: ShockBurst™ operation is outlined with a dashed square.

Figure 11. PRX operations in Enhanced ShockBurst™

Activate PRX mode by setting the **CE** pin high. The nRF24L01+ enters RX mode and starts searching for packets. If a packet is received and Auto Acknowledgement is enabled, nRF24L01+ decides if the packet is new or a copy of a previously received packet. If the packet is new the payload is made available in the

RX FIFO and the RX_DR IRQ is asserted. If the last received packet from the transmitter is acknowledged with an ACK packet with payload, the TX_DS IRQ indicates that the PTX received the ACK packet with payload. If the No_ACK flag is not set in the received packet, the PRX enters TX mode. If there is a pending payload in the TX FIFO it is attached to the ACK packet. After the ACK packet is transmitted, the nRF24L01+ returns to RX mode.

A copy of a previously received packet might be received if the ACK packet is lost. In this case, the PRX discards the received packet and transmits an ACK packet before it returns to RX mode.

7.6 MultiCeiver™

MultiCeiver™ is a feature used in RX mode that contains a set of six parallel data pipes with unique addresses. A data pipe is a logical channel in the physical RF channel. Each data pipe has its own physical address (data pipe address) decoding in the nRF24L01+.

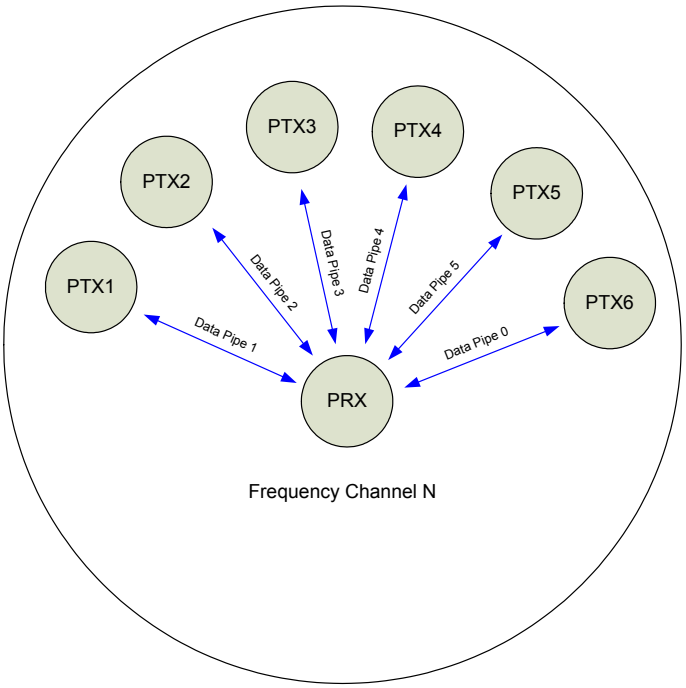


Figure 12. PRX using MultiCeiver™

nRF24L01+ configured as PRX (primary receiver) can receive data addressed to six different data pipes in one frequency channel as shown in Figure 12. Each data pipe has its own unique address and can be configured for individual behavior.

Up to six nRF24L01+s configured as PTX can communicate with one nRF24L01+ configured as a PRX. All data pipe addresses are searched for simultaneously. Only one data pipe can receive a packet at a time. All data pipes can perform Enhanced ShockBurst™ functionality.

The following settings are common to all data pipes:

- CRC enabled/disabled (CRC always enabled when Enhanced ShockBurst™ is enabled)
- CRC encoding scheme
- RX address width
- Frequency channel
- Air data rate
- LNA gain

The data pipes are enabled with the bits in the `EN_RXADDR` register. By default only data pipe 0 and 1 are enabled. Each data pipe address is configured in the `RX_ADDR_PX` registers.

Note: Always ensure that none of the data pipes have the same address.

Each pipe can have up to a 5 byte configurable address. Data pipe 0 has a unique 5 byte address. Data pipes 1-5 share the four most significant address bytes. The LSByte must be unique for all six pipes. Figure 13. is an example of how data pipes 0-5 are addressed.

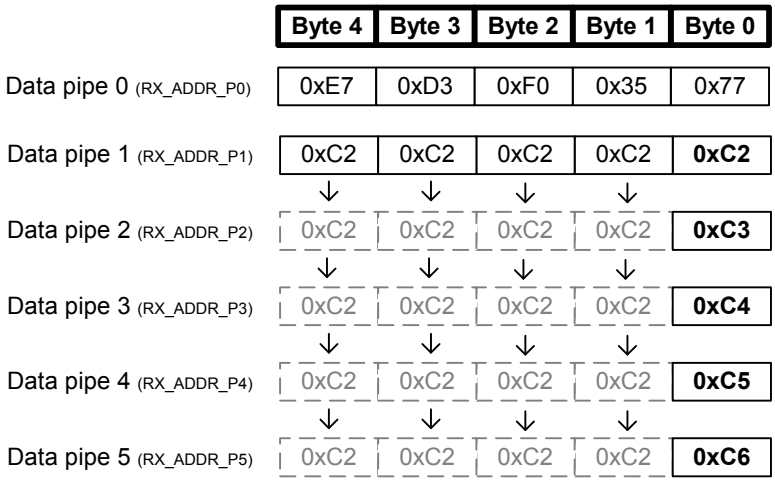


Figure 13. Addressing data pipes 0-5

The PRX, using MultiCeiver™ and Enhanced ShockBurst™, receives packets from more than one PTX. To ensure that the ACK packet from the PRX is transmitted to the correct PTX, the PRX takes the data pipe address where it received the packet and uses it as the TX address when transmitting the ACK packet. Figure 14. is an example of an address configuration for the PRX and PTX. On the PRX the `RX_ADDR_Pn`, defined as the pipe address, must be unique. On the PTX the `TX_ADDR` must be the same as the `RX_ADDR_P0` and as the pipe address for the designated pipe.

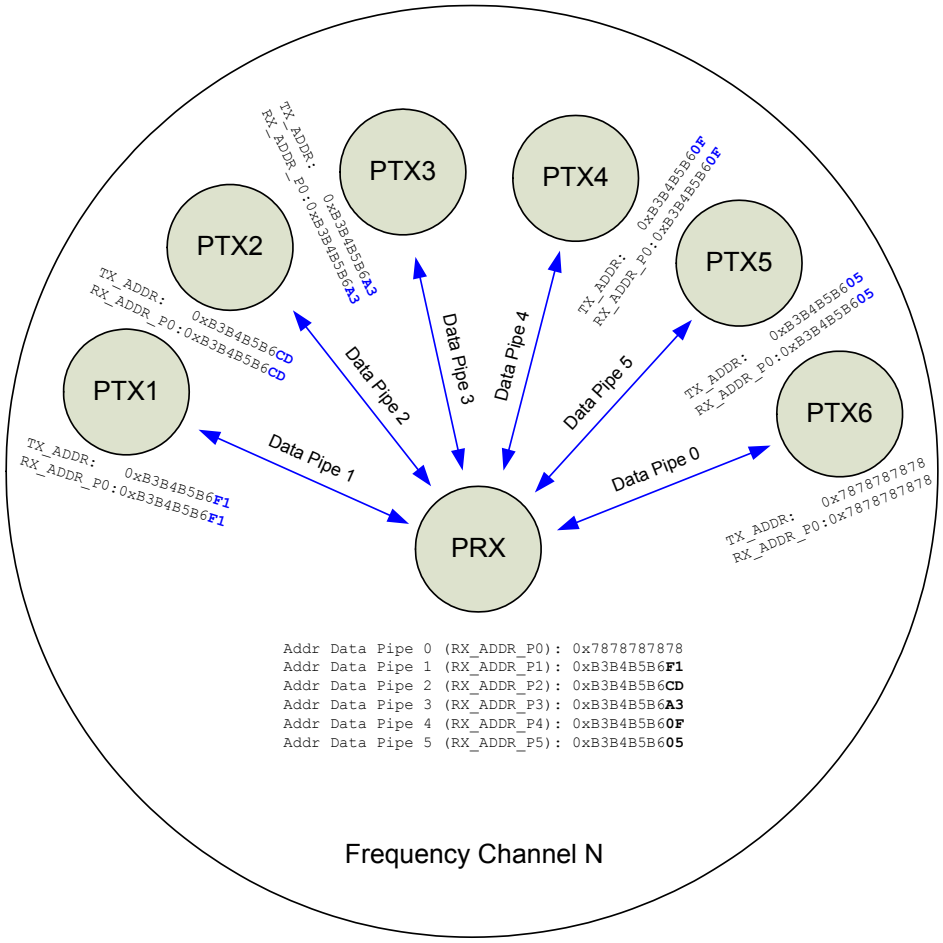
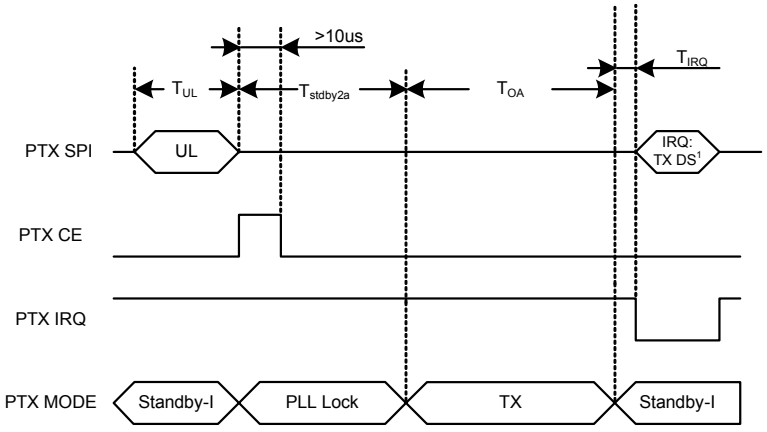


Figure 14. Example of data pipe addressing in MultiCeiver™

Only when a data pipe receives a complete packet can other data pipes begin to receive data. When multiple PTXs are transmitting to a PRX, the ARD can be used to skew the auto retransmission so that they only block each other once.

7.7 Enhanced ShockBurst™ timing

This section describes the timing sequence of Enhanced ShockBurst™ and how all modes are initiated and operated. The Enhanced ShockBurst™ timing is controlled through the Data and Control interface. The nRF24L01+ can be set to static modes or autonomous modes where the internal state machine controls the events. Each autonomous mode/sequence ends with an interrupt at the **IRQ** pin. All the interrupts are indicated as IRQ events in the timing diagrams.



1 IRQ if No Ack is on.

T_{IRQ} = 8.2µs @ 1Mbps, T_{IRQ} = 6.0µs @ 2Mbps, $T_{stdby2a}$ = 130us

Figure 15. Transmitting one packet with NO_ACK on

The following equations calculate various timing measurements:

Symbol	Description	Equation
T_{OA}	Time on-air	$T_{OA} = \frac{packet\ length}{air\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot\left(1[byte]_{preamble} + 3,4\ or\ 5[bytes]_{address} + N[bytes]_{payload} + 1\ or\ 2[bytes]_{CRC}\right) + 9[bit]_{packet\ control\ field}}{air\ data\ rate\left[\frac{bit}{s}\right]}$
T_{ACK}	Time on-air Ack	$T_{ACK} = \frac{packet\ length}{air\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot\left(1[byte]_{preamble} + 3,4\ or\ 5[bytes]_{address} + N[bytes]_{payload} + 1\ or\ 2[bytes]_{CRC}\right) + 9[bit]_{packet\ control\ field}}{air\ data\ rate\left[\frac{bit}{s}\right]}$
T_{UL}	Time Upload	$T_{UL} = \frac{payload\ length}{SPI\ data\ rate} = \frac{8\left[\frac{bit}{byte}\right]\cdot N[bytes]_{payload}}{SPI\ data\ rate\left[\frac{bit}{s}\right]}$
T_{ESB}	Time Enhanced ShockBurst™ cycle	$T_{ESB} = T_{UL} + 2\ .\ T_{stby2a} + T_{OA} + T_{ACK} + T_{IRQ}$

Table 19. Timing equations

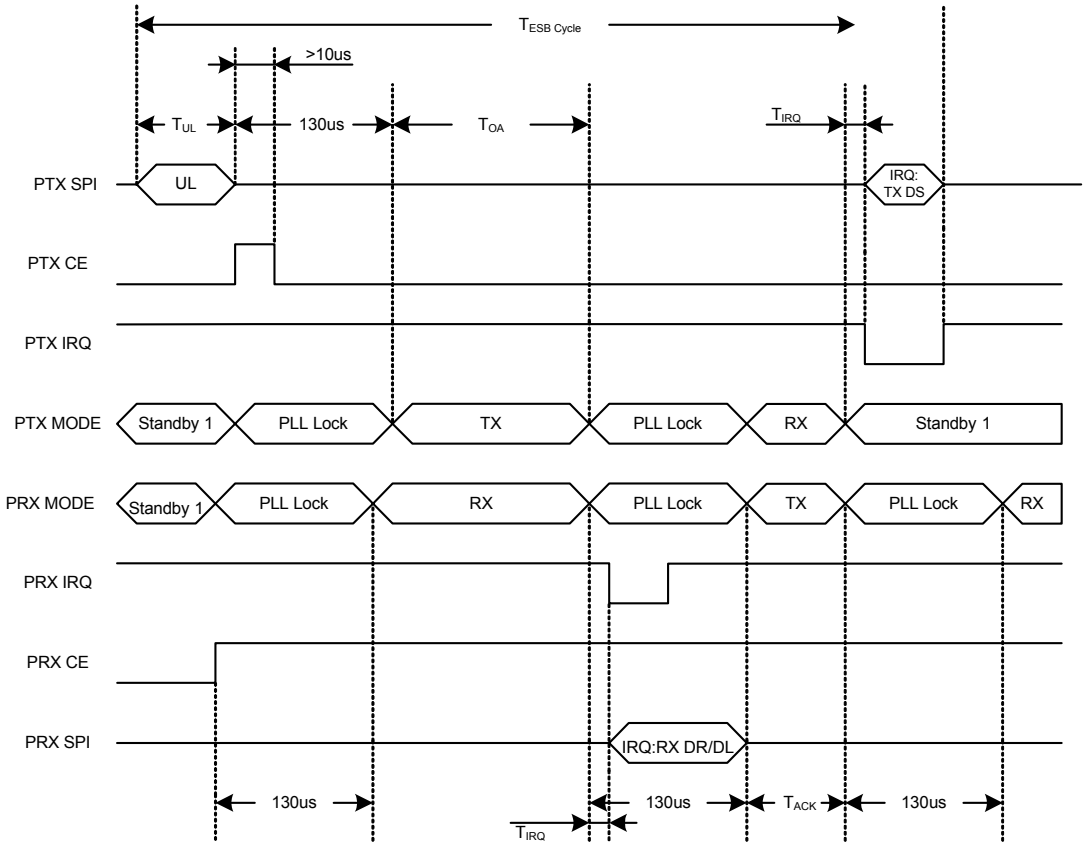


Figure 16. Timing of Enhanced ShockBurst™ for one packet upload (2Mbps)

In Figure 16. the transmission and acknowledgement of a packet is shown. The PRX device activates RX mode (**CE**=1), and the PTX device is activated in TX mode (**CE**=1 for minimum 10µs). After 130µs the transmission starts and finishes after the elapse of T_{OA}.

When the transmission ends the PTX device automatically switches to RX mode to wait for the ACK packet from the PRX device. When the PRX device receives the packet it sets the interrupt for the host MCU and switches to TX mode to send an ACK. After the PTX device receives the ACK packet it sets the interrupt to the MCU and clears the packet from the TX FIFO.

In Figure 17. the PTX timing of a packet transmission is shown when the first ACK packet is lost. To see the complete transmission when the ACK packet fails see Figure 20. on page 46.

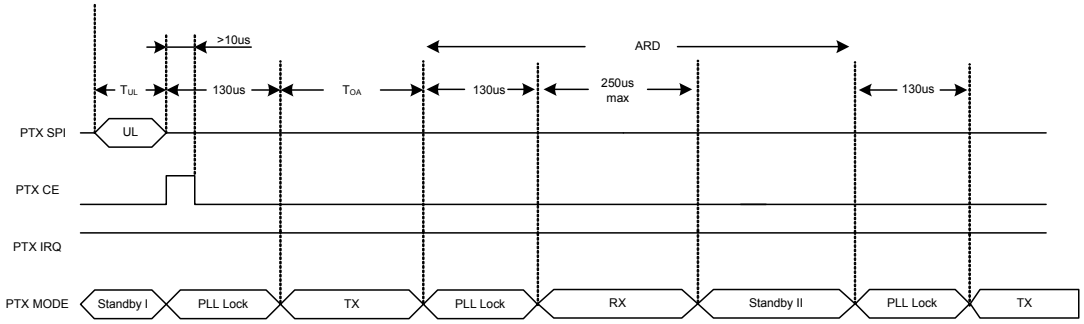


Figure 17. Timing of Enhanced ShockBurst™ when the first ACK packet is lost (2Mbps)

7.8 Enhanced ShockBurst™ transaction diagram

This section describes several scenarios for the Enhanced ShockBurst™ automatic transaction handling. The call outs in this section's figures indicate the IRQs and other events. For MCU activity the event may be placed at a different timeframe.

Note: The figures in this section indicate the earliest possible download (DL) of the packet to the MCU and the latest possible upload (UL) of payload to the transmitter.

7.8.1 Single transaction with ACK packet and interrupts

In Figure 18. the basic auto acknowledgement is shown. After the packet is transmitted by the PTX and received by the PRX the ACK packet is transmitted from the PRX to the PTX. The `RX_DR` IRQ is asserted after the packet is received by the PRX, whereas the `TX_DS` IRQ is asserted when the packet is acknowledged and the ACK packet is received by the PTX.

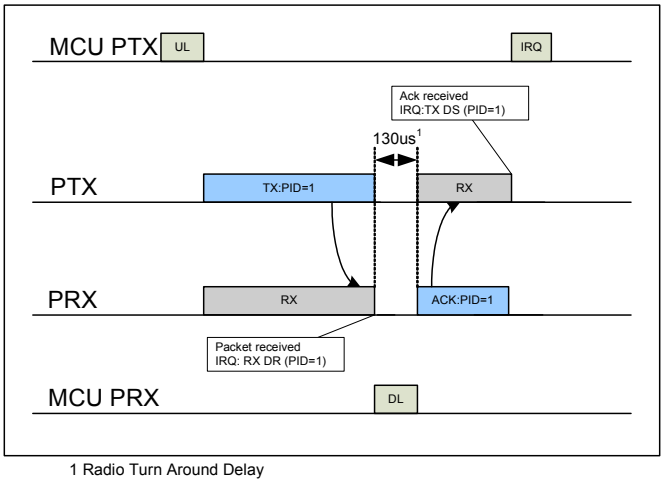


Figure 18. TX/RX cycles with ACK and the according interrupts

7.8.2 Single transaction with a lost packet

Figure 19. is a scenario where a retransmission is needed due to loss of the first packet transmit. After the packet is transmitted, the PTX enters RX mode to receive the ACK packet. After the first transmission, the PTX waits a specified time for the ACK packet, if it is not in the specific time slot the PTX retransmits the packet as shown in Figure 19.

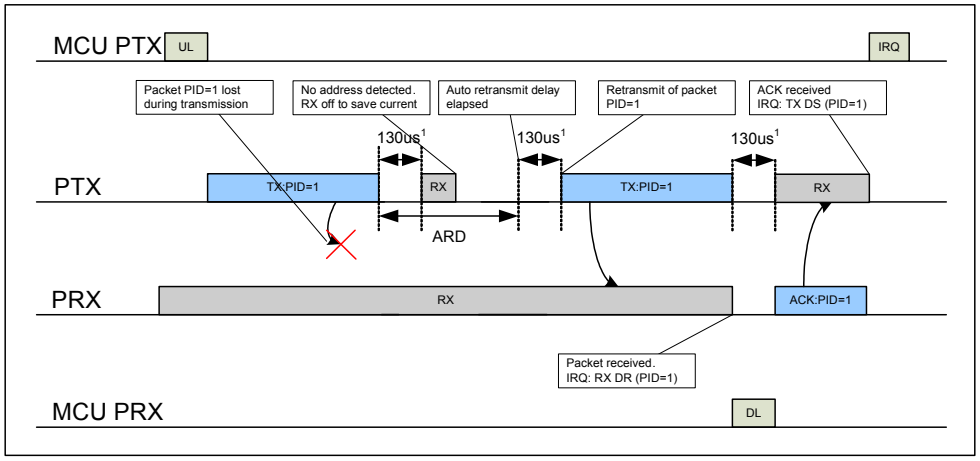


Figure 19. TX/RX cycles with ACK and the according interrupts when the first packet transmit fails

When an address is detected the PTX stays in RX mode until the packet is received. When the retransmitted packet is received by the PRX (see Figure 19.), the `RX_DR` IRQ is asserted and an ACK is transmitted back to the PTX. When the ACK is received by the PTX, the `TX_DS` IRQ is asserted.

7.8.3 Single transaction with a lost ACK packet

Figure 20. is a scenario where a retransmission is needed after a loss of the ACK packet. The corresponding interrupts are also indicated.

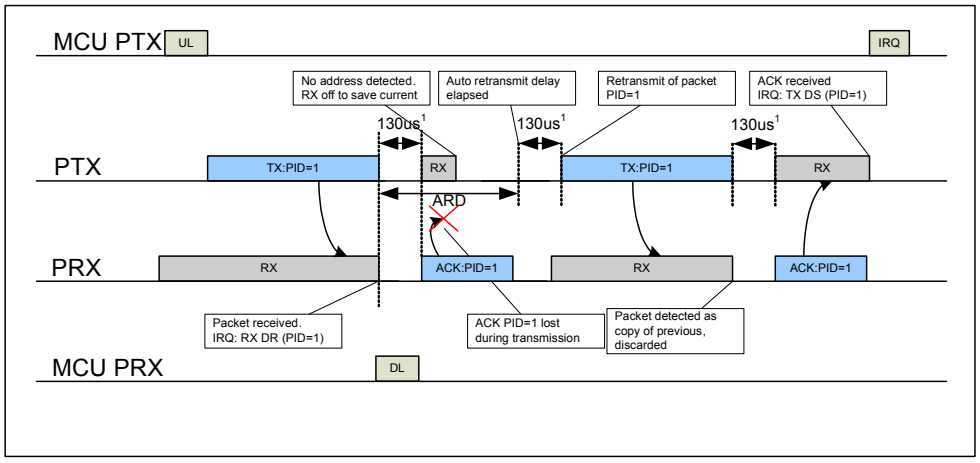
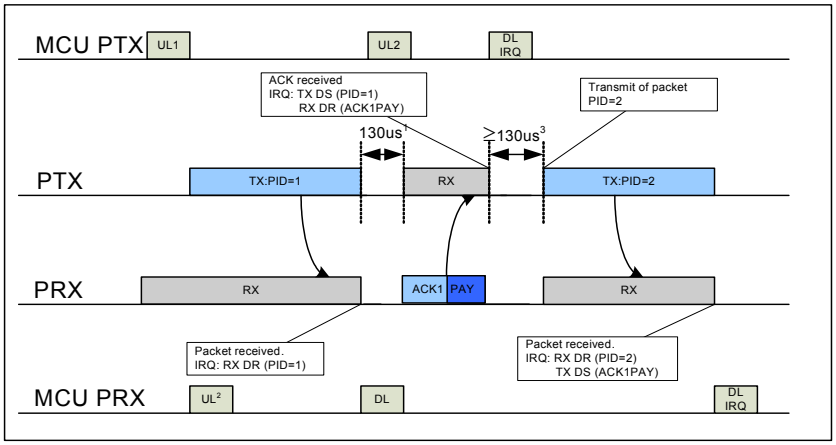


Figure 20. TX/RX cycles with ACK and the according interrupts when the ACK packet fails

7.8.4 Single transaction with ACK payload packet

Figure 21. is a scenario of the basic auto acknowledgement with payload. After the packet is transmitted by the PTX and received by the PRX the ACK packet with payload is transmitted from the PRX to the PTX. The RX_DR IRQ is asserted after the packet is received by the PRX, whereas on the PTX side the TX_DS IRQ is asserted when the ACK packet is received by the PTX. On the PRX side, the TX_DS IRQ for the ACK packet payload is asserted after a new packet from PTX is received. The position of the IRQ in Figure 21. shows where the MCU can respond to the interrupt.

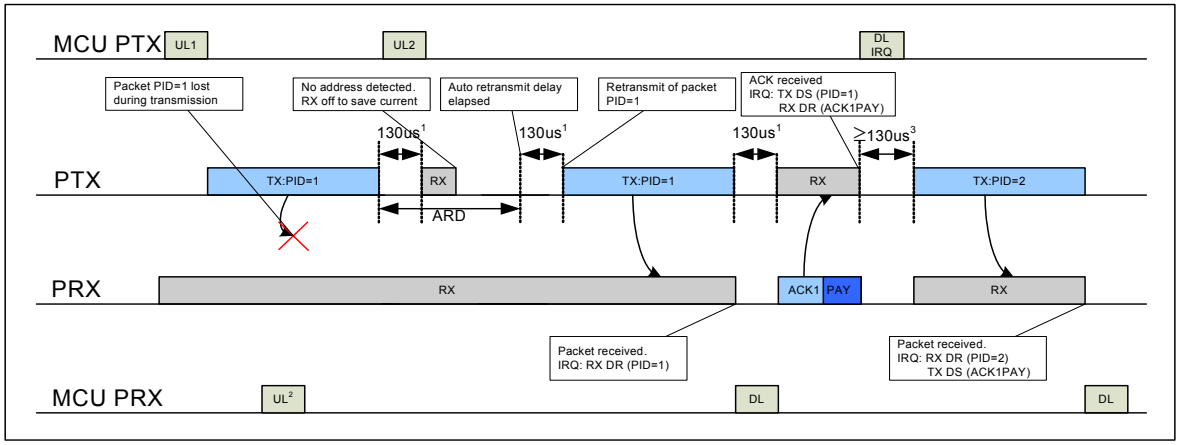


1 Radio Turn Around Delay
2 Uploading Payload for Ack Packet
3 Delay defined by MCU on PTX side, ≥ 130us

Figure 21. TX/RX cycles with ACK Payload and the according interrupts

7.8.5 Single transaction with ACK payload packet and lost packet

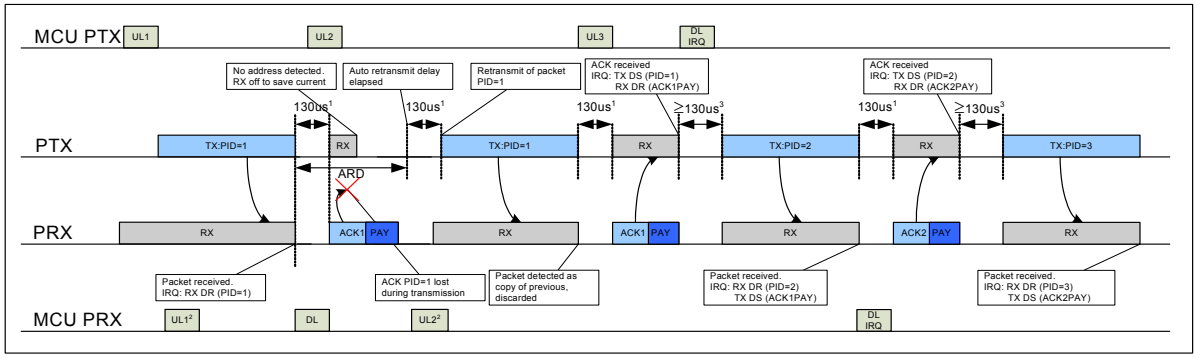
Figure 22. is a scenario where the first packet is lost and a retransmission is needed before the RX_DR IRQ on the PRX side is asserted. For the PTX both the TX_DS and RX_DR IRQ are asserted after the ACK packet is received. After the second packet (PID=2) is received on the PRX side both the RX_DR (PID=2) and TX_DS (ACK packet payload) IRQ are asserted.



1 Radio Turn Around Delay
2 Uploading Paylod for Ack Packet
3 Delay defined by MCU on PTX side, ≥ 130us

Figure 22. TX/RX cycles and the according interrupts when the packet transmission fails

7.8.6 Two transactions with ACK payload packet and the first ACK packet lost

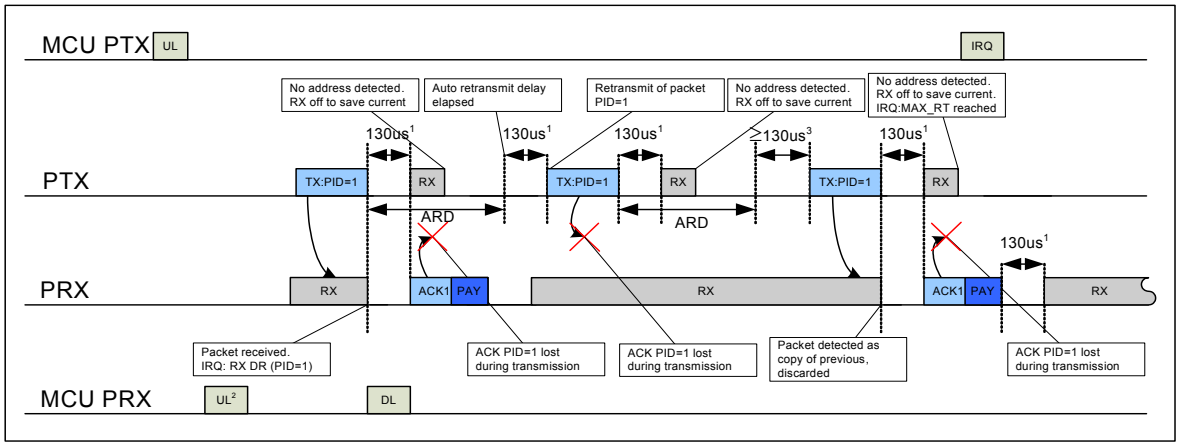


1 Radio Turn Around Delay
2 Uploading Payload for Ack Packet
3 Delay defined by MCU on PTX side, ≥ 130us

Figure 23. TX/RX cycles with ACK Payload and the according interrupts when the ACK packet fails

In Figure 23. the ACK packet is lost and a retransmission is needed before the TX_DS IRQ is asserted, but the RX_DR IRQ is asserted immediately. The retransmission of the packet (PID=1) results in a discarded packet. For the PTX both the TX_DS and RX_DR IRQ are asserted after the second transmission of ACK, which is received. After the second packet (PID=2) is received on the PRX both the RX_DR (PID=2) and TX_DS (ACK1PAY) IRQ is asserted. The callouts explains the different events and interrupts.

7.8.7 Two transactions where max retransmissions is reached



1 Radio Turn Around Delay
2 Uploading Paylod for Ack Packet
3 Delay defined by MCU on PTX side, ≥ 130us

Figure 24. TX/RX cycles with ACK Payload and the according interrupts when the transmission fails. ARC is set to 2.

MAX_RT IRQ is asserted if the auto retransmit counter (ARC_CNT) exceeds the programmed maximum limit (ARC). In Figure 24. the packet transmission ends with a MAX_RT IRQ. The payload in TX FIFO is NOT removed and the MCU decides the next step in the protocol. A toggle of the **CE** starts a new transmitting sequence of the same packet. The payload can be removed from the TX FIFO using the FLUSH_TX command.

7.9 Compatibility with ShockBurst™

You must disable Enhanced ShockBurst™ for backward compatibility with the nRF2401A, nRF2402, nRF24E1 and nRF24E2. Set the register `EN_AA` = 0x00 and `ARC` = 0 to disable Enhanced ShockBurst™. In addition, the nRF24L01+ air data rate must be set to 1Mbps or 250kbps.

7.9.1 ShockBurst™ packet format

Figure 25. shows the packet format with MSB to the left.

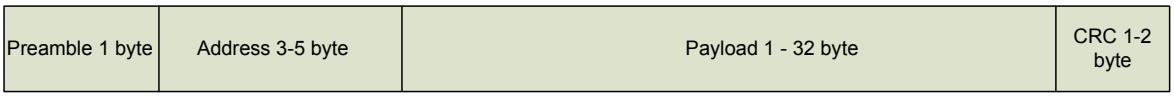


Figure 25. A ShockBurst™ packet compatible with nRF2401/nRF2402/nRF24E1/nRF24E2 devices.

The ShockBurst™ packet format has a preamble, address, payload and CRC field that are the same as the Enhanced ShockBurst™ packet format described in section 7.3 on page 28.

The differences between the ShockBurst™ packet and the Enhanced ShockBurst™ packet are:

- The 9 bit Packet Control Field is not present in the ShockBurst™ packet format.
- The CRC is optional in the ShockBurst™ packet format and is controlled by the `EN_CRC` bit in the `CONFIG` register.

8 Data and Control Interface

The data and control interface gives you access to all the features in the nRF24L01+. The data and control interface consists of the following six 5Volt tolerant digital signals:

- **IRQ** (this signal is active low and controlled by three maskable interrupt sources)
- **CE** (this signal is active high and used to activate the chip in RX or TX mode)
- **CSN** (SPI signal)
- **SCK** (SPI signal)
- **MOSI** (SPI signal)
- **MISO** (SPI signal)

Using 1 byte SPI commands, you can activate the nRF24L01+ data FIFOs or the register map during all modes of operation.

8.1 Features

- Special SPI commands for quick access to the most frequently used features
- 0-10Mbps 4-wire SPI
- 8 bit command set
- Easily configurable register map
- Full three level FIFO for both TX and RX direction

8.2 Functional description

The SPI is a standard SPI with a maximum data rate of 10Mbps.

8.3 SPI operation

This section describes the SPI commands and timing.

8.3.1 SPI commands

The SPI commands are shown in Table 20. Every new command must be started by a high to low transition on **CSN**.

The STATUS register is serially shifted out on the **MISO** pin simultaneously to the SPI command word shifting to the **MOSI** pin.

The serial shifting SPI commands is in the following format:

<**Command word**: MSBit to LSBit (one byte)>

<**Data bytes**: LSByte to MSByte, MSBit in each byte first>

See Figure 26. on page 52 and Figure 27. on page 52 for timing information.

Command name	Command word (binary)	# Data bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and status registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and status registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last transmitted payload. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission.
R_RX_PL_WID[a]	0110 0000	1	Read RX payload width for the top R_RX_PAYLOAD in the RX FIFO. **Note:** Flush RX FIFO if the read value is larger than 32 bytes.
W_ACK_PAYLOAD[a]	1010 1PPP	1 to 32 LSByte first	Used in RX mode. Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0.
W_TX_PAYLOAD_NOACK[a]	1011 0000	1 to 32 LSByte first	Used in TX mode. Disables AUTOACK on this specific packet.
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

a. The bits in the FEATURE register shown in Table 28. on page 63 have to be set.

Table 20. Command set for the nRF24L01+ SPI

The W_REGISTER and R_REGISTER commands operate on single or multi-byte registers. When accessing multi-byte registers read or write to the MSBit of LSByte first. You can terminate the writing before all bytes in a multi-byte register are written, leaving the unwritten MSByte(s) unchanged. For example, the LSByte of RX_ADDR_P0 can be modified by writing only one byte to the RX_ADDR_P0 register. The content of the status register is always read to **MISO** after a high to low transition on **CSN**.

Note: The 3 bit pipe information in the STATUS register is updated during the **IRQ** pin high to low transition. The pipe information is unreliable if the STATUS register is read during an **IRQ** pin high to low transition.

8.3.2 SPI timing

SPI operation and timing is shown in Figure 26. to Figure 28. and in Table 22. to Table 27.. nRF24L01+ must be in a standby or power down mode before writing to the configuration registers.

In Figure 26. to Figure 28. the following abbreviations are used:

Abbreviation	Description
Cn	SPI command bit
Sn	STATUS register bit
Dn	Data Bit (**Note:** LSByte to MSByte, MSBit in each byte first)

Table 21. Abbreviations used in Figure 26. to Figure 28.

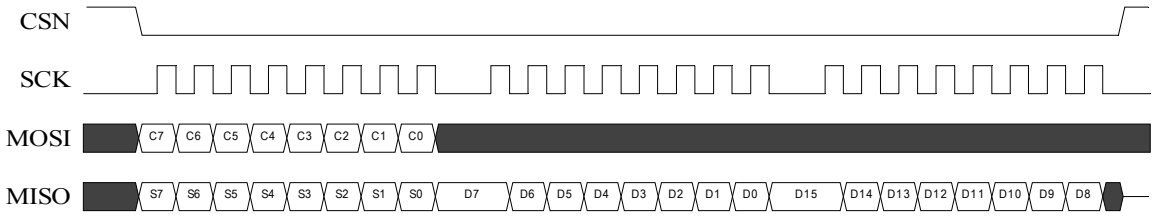


Figure 26. SPI read operation

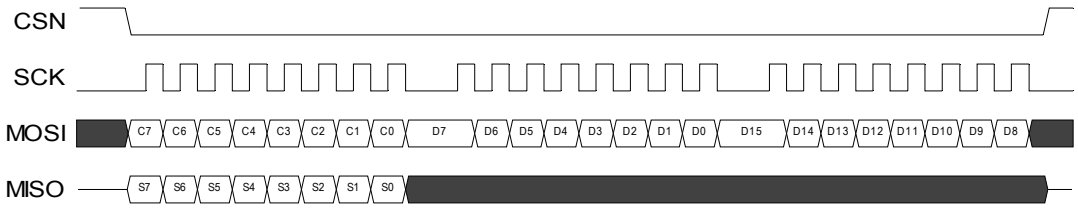


Figure 27. SPI write operation

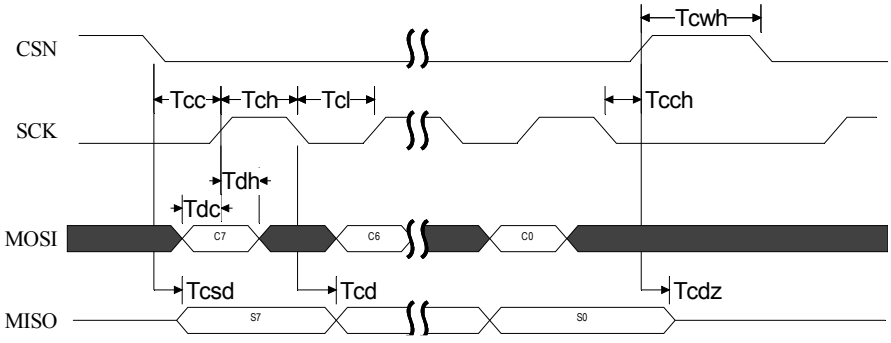


Figure 28. SPI NOP timing diagram

Figure 29. shows the R_{pull} and C_{load} that are referenced in Table 22. to Table 27.

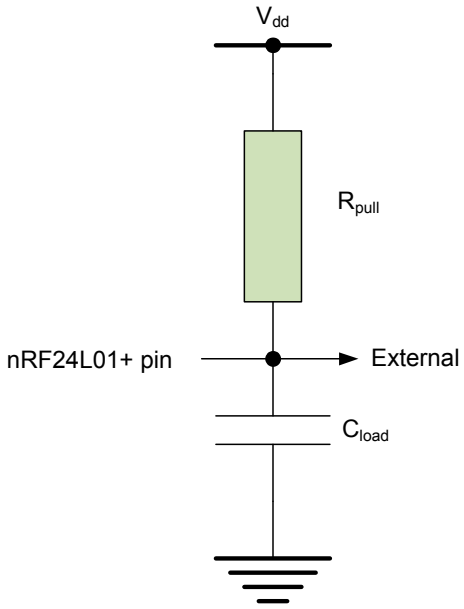


Figure 29. R_{pull} and C_{load}

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		38	ns
Tcd	SCK to Data Valid		55	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	10	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		38	ns

Table 22. SPI timing parameters (C_{load} = 5pF)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		42	ns
Tcd	SCK to Data Valid		58	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	8	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns

Symbol	Parameters	Min.	Max	Units
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		42	ns

Table 23. SPI timing parameters (C_{load} = 10pF)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		75	ns
Tcd	SCK to Data Valid		86	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	5	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		75	ns

Table 24. SPI timing parameters (R_{pull} = 10kΩ, C_{load} = 50pF)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		116	ns
Tcd	SCK to Data Valid		123	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	4	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		116	ns

Table 25. SPI timing parameters (R_{pull} = 10kΩ, C_{load} = 100pF)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		75	ns
Tcd	SCK to Data Valid		85	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	5	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		75	ns

Table 26. SPI timing parameters (R_{pull} = 50kΩ, C_{load} = 50pF)

Symbol	Parameters	Min.	Max	Units
Tdc	Data to SCK Setup	2		ns
Tdh	SCK to Data Hold	2		ns
Tcsd	CSN to Data Valid		116	ns
Tcd	SCK to Data Valid		121	ns
Tcl	SCK Low Time	40		ns
Tch	SCK High Time	40		ns
Fsck	SCK Frequency	0	4	MHz
Tr,Tf	SCK Rise and Fall		100	ns
Tcc	CSN to SCK Setup	2		ns
Tcch	SCK to CSN Hold	2		ns
Tcwh	CSN Inactive time	50		ns
Tcdz	CSN to Output High Z		116	ns

Table 27. SPI timing parameters (R_{pull} = 50kΩ, C_{load} = 100pF)

8.4 Data FIFO

The data FIFOs store transmitted payloads (TX FIFO) or received payloads that are ready to be clocked out (RX FIFO). The FIFOs are accessible in both PTX mode and PRX mode.

The following FIFOs are present in nRF24L01+:

- TX three level, 32 byte FIFO
- RX three level, 32 byte FIFO

Both FIFOs have a controller and are accessible through the SPI by using dedicated SPI commands. A TX FIFO in PRX can store payloads for ACK packets to three different PTX devices. If the TX FIFO contains more than one payload to a pipe, payloads are handled using the first in - first out principle. The TX FIFO in a PRX is blocked if all pending payloads are addressed to pipes where the link to the PTX is lost. In this case, the MCU can flush the TX FIFO using the `FLUSH_TX` command.

The RX FIFO in PRX can contain payloads from up to three different PTX devices and a TX FIFO in PTX can have up to three payloads stored.

You can write to the TX FIFO using these three commands; W_TX_PAYLOAD and W_TX_PAYLOAD_NO_ACK in PTX mode and W_ACK_PAYLOAD in PRX mode. All three commands provide access to the TX_PLD register (see Table 28. on page 63. for details of this register).

The RX FIFO can be read by the command R_RX_PAYLOAD in PTX and PRX mode. This command provides access to the RX_PLD register.

The payload in TX FIFO in a PTX is not removed if the MAX_RT IRQ is asserted.

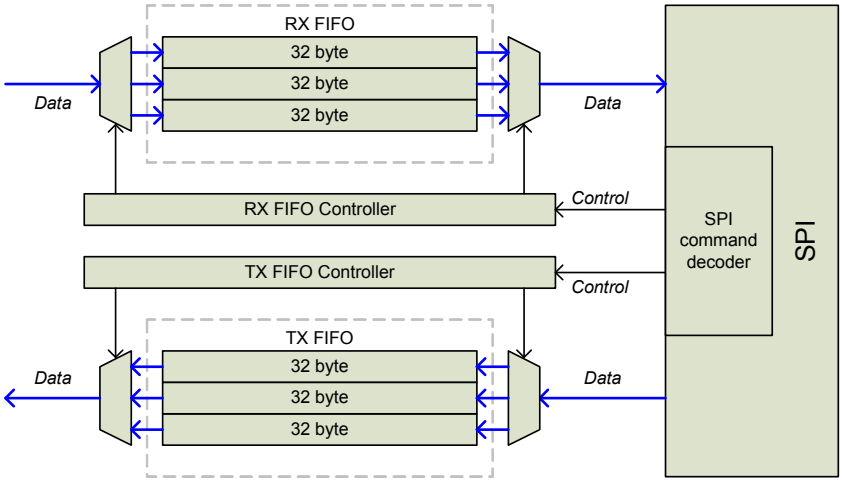


Figure 30. FIFO (RX and TX) block diagram

You can read if the TX and RX FIFO are full or empty in the FIFO_STATUS register.

8.5 Interrupt

The nRF24L01+ has an active low interrupt (**IRQ**) pin. The **IRQ** pin is activated when TX_DS IRQ, RX_DR IRQ or MAX_RT IRQ are set high by the state machine in the STATUS register. The **IRQ** pin resets when MCU writes '1' to the IRQ source bit in the STATUS register. The IRQ mask in the CONFIG register is used to select the IRQ sources that are allowed to assert the **IRQ** pin. By setting one of the MASK bits high, the corresponding IRQ source is disabled. By default all IRQ sources are enabled.

Note: The 3 bit pipe information in the STATUS register is updated during the **IRQ** pin high to low transition. The pipe information is unreliable if the STATUS register is read during an **IRQ** pin high to low transition.

9 Register Map

You can configure and control the radio by accessing the register map through the SPI.

9.1 Register map table

All undefined bits in the table below are redundant. They are read out as '0'.

Note: Addresses 18 to 1B are reserved for test purposes, altering them makes the chip malfunction.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the **IRQ** pin 0: Reflect RX_DR as active low interrupt on the **IRQ** pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the **IRQ** pin 0: Reflect TX_DS as active low interrupt on the **IRQ** pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the **IRQ** pin 0: Reflect MAX_RT as active low interrupt on the **IRQ** pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX
01	EN_AA Enhanced ShockBurst™				Enable 'Auto Acknowledgment' Function Disable this functionality to be compatible with nRF2401, see page 75
	Reserved	7:6	00	R/W	Only '00' allowed
	ENAA_P5	5	1	R/W	Enable auto acknowledgement data pipe 5
	ENAA_P4	4	1	R/W	Enable auto acknowledgement data pipe 4
	ENAA_P3	3	1	R/W	Enable auto acknowledgement data pipe 3
	ENAA_P2	2	1	R/W	Enable auto acknowledgement data pipe 2
	ENAA_P1	1	1	R/W	Enable auto acknowledgement data pipe 1
	ENAA_P0	0	1	R/W	Enable auto acknowledgement data pipe 0
02	EN_RXADDR				Enabled RX Addresses
	Reserved	7:6	00	R/W	Only '00' allowed
	ERX_P5	5	0	R/W	Enable data pipe 5.
	ERX_P4	4	0	R/W	Enable data pipe 4.
	ERX_P3	3	0	R/W	Enable data pipe 3.
	ERX_P2	2	0	R/W	Enable data pipe 2.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	ERX_P1	1	1	R/W	Enable data pipe 1.
	ERX_P0	0	1	R/W	Enable data pipe 0.
03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' – 5 bytes LSByte is used if address width is below 5 bytes
04	SETUP_RETR				Setup of Automatic Retransmission
	ARD[a]	7:4	0000	R/W	Auto Retransmit Delay ‘0000’ – Wait 250µS ‘0001’ – Wait 500µS ‘0010’ – Wait 750µS …….. ‘1111’ – Wait 4000µS (Delay defined from end of transmission to start of next transmission)[b]
	ARC	3:0	0011	R/W	Auto Retransmit Count ‘0000’ –Re-Transmit disabled ‘0001’ – Up to 1 Re-Transmit on fail of AA …… ‘1111’ – Up to 15 Re-Transmit on fail of AA
05	RF_CH				RF Channel
	Reserved	7	0	R/W	Only '0' allowed
	RF_CH	6:0	0000010	R/W	Sets the frequency channel nRF24L01+ operates on
06	RF_SETUP				RF Setup Register
	CONT_WAVE	7	0	R/W	Enables continuous carrier transmit when high.
	Reserved	6	0	R/W	Only '0' allowed
	RF_DR_LOW	5	0	R/W	Set RF Data Rate to 250kbps. See RF_DR_HIGH for encoding.
	PLL_LOCK	4	0	R/W	Force PLL lock signal. Only used in test
	RF_DR_HIGH	3	1	R/W	Select between the high speed data rates. This bit is don’t care if RF_DR_LOW is set. Encoding: [RF_DR_LOW, RF_DR_HIGH]: ‘00’ – 1Mbps ‘01’ – 2Mbps ‘10’ – 250kbps ‘11’ – Reserved

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RF_PWR	2:1	11	R/W	Set RF output power in TX mode '00' – -18dBm '01' – -12dBm '10' – -6dBm '11' – 0dBm
	Obsolete	0			Don't care
07	STATUS				Status Register (In parallel to the SPI command word applied on the **MOSI** pin, the STATUS register is shifted serially out on the **MISO** pin)
	Reserved	7	0	R/W	Only '0' allowed
	RX_DR	6	0	R/W	Data Ready RX FIFO interrupt. Asserted when new data arrives RX FIFO[c]. Write 1 to clear bit.
	TX_DS	5	0	R/W	Data Sent TX FIFO interrupt. Asserted when packet transmitted on TX. If AUTO_ACK is activated, this bit is set high only when ACK is received. Write 1 to clear bit.
	MAX_RT	4	0	R/W	Maximum number of TX retransmits interrupt Write 1 to clear bit. If MAX_RT is asserted it must be cleared to enable further communication.
	RX_P_NO	3:1	111	R	Data pipe number for the payload available for reading from RX_FIFO 000-101: Data Pipe Number 110: Not Used 111: RX FIFO Empty
	TX_FULL	0	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
08	OBSERVE_TX				Transmit observe register
	PLOS_CNT	7:4	0	R	Count lost packets. The counter is overflow protected to 15, and discontinues at max until reset. The counter is reset by writing to RF_CH. See page 75.
	ARC_CNT	3:0	0	R	Count retransmitted packets. The counter is reset when transmission of a new packet starts. See page 75.
09	RPD				
	Reserved	7:1	000000	R	
	RPD	0	0	R	Received Power Detector. This register is called CD (Carrier Detect) in the nRF24L01. The name is different in nRF24L01+ due to the different input power level threshold for this bit. See section 6.4 on page 25.
0A	RX_ADDR_P0	39:0	0xE7E7E7E7E7	R/W	Receive address data pipe 0. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
0B	RX_ADDR_P1	39:0	0xC2C2C2C2C2	R/W	Receive address data pipe 1. 5 Bytes maximum length. (LSByte is written first. Write the number of bytes defined by SETUP_AW)
0C	RX_ADDR_P2	7:0	0xC3	R/W	Receive address data pipe 2. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0D	RX_ADDR_P3	7:0	0xC4	R/W	Receive address data pipe 3. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0E	RX_ADDR_P4	7:0	0xC5	R/W	Receive address data pipe 4. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
0F	RX_ADDR_P5	7:0	0xC6	R/W	Receive address data pipe 5. Only LSB. MSBytes are equal to RX_ADDR_P1[39:8]
10	TX_ADDR	39:0	0xE7E7E7E7E7	R/W	Transmit address. Used for a PTX device only. (LSByte is written first) Set RX_ADDR_P0 equal to this address to handle automatic acknowledge if this is a PTX device with Enhanced ShockBurst™ enabled. See page 75.
11	RX_PW_P0				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P0	5:0	0	R/W	Number of bytes in RX payload in data pipe 0 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
12	RX_PW_P1				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P1	5:0	0	R/W	Number of bytes in RX payload in data pipe 1 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
13	RX_PW_P2				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P2	5:0	0	R/W	Number of bytes in RX payload in data pipe 2 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
14	RX_PW_P3				
	Reserved	7:6	00	R/W	Only '00' allowed

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	RX_PW_P3	5:0	0	R/W	Number of bytes in RX payload in data pipe 3 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
15	RX_PW_P4				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P4	5:0	0	R/W	Number of bytes in RX payload in data pipe 4 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
16	RX_PW_P5				
	Reserved	7:6	00	R/W	Only '00' allowed
	RX_PW_P5	5:0	0	R/W	Number of bytes in RX payload in data pipe 5 (1 to 32 bytes). 0 Pipe not used 1 = 1 byte … 32 = 32 bytes
17	FIFO_STATUS				FIFO Status Register
	Reserved	7	0	R/W	Only '0' allowed
	TX_REUSE	6	0	R	Used for a PTX device Pulse the rfce high for at least 10µs to Reuse last transmitted payload. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX_REUSE is set by the SPI command REUSE_TX_PL, and is reset by the SPI commands W_TX_PAYLOAD or FLUSH TX
	TX_FULL	5	0	R	TX FIFO full flag. 1: TX FIFO full. 0: Available locations in TX FIFO.
	TX_EMPTY	4	1	R	TX FIFO empty flag. 1: TX FIFO empty. 0: Data in TX FIFO.
	Reserved	3:2	00	R/W	Only '00' allowed
	RX_FULL	1	0	R	RX FIFO full flag. 1: RX FIFO full. 0: Available locations in RX FIFO.
	RX_EMPTY	0	1	R	RX FIFO empty flag. 1: RX FIFO empty. 0: Data in RX FIFO.

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
N/A	ACK_PLD	255:0	X	W	Written by separate SPI command ACK packet payload to data pipe number PPP given in SPI command. Used in RX mode only. Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled first in first out.
N/A	TX_PLD	255:0	X	W	Written by separate SPI command TX data payload register 1 - 32 bytes. This register is implemented as a FIFO with three levels. Used in TX mode only.
N/A	RX_PLD	255:0	X	R	Read by separate SPI command. RX data payload register. 1 - 32 bytes. This register is implemented as a FIFO with three levels. All RX channels share the same FIFO.
1C	DYNPD				Enable dynamic payload length
	Reserved	7:6	0	R/W	Only '00' allowed
	DPL_P5	5	0	R/W	Enable dynamic payload length data pipe 5. (Requires EN_DPL and ENAA_P5)
	DPL_P4	4	0	R/W	Enable dynamic payload length data pipe 4. (Requires EN_DPL and ENAA_P4)
	DPL_P3	3	0	R/W	Enable dynamic payload length data pipe 3. (Requires EN_DPL and ENAA_P3)

Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
	DPL_P2	2	0	R/W	Enable dynamic payload length data pipe 2. (Requires EN_DPL and ENAA_P2)
	DPL_P1	1	0	R/W	Enable dynamic payload length data pipe 1. (Requires EN_DPL and ENAA_P1)
	DPL_P0	0	0	R/W	Enable dynamic payload length data pipe 0. (Requires EN_DPL and ENAA_P0)
1D	FEATURE			R/W	Feature Register
	Reserved	7:3	0	R/W	Only '00000' allowed
	EN_DPL	2	0	R/W	Enables Dynamic Payload Length
	EN_ACK_PAY[d]	1	0	R/W	Enables Payload with ACK
	EN_DYN_ACK	0	0	R/W	Enables the W_TX_PAYLOAD_NOACK command

a. Please take care when setting this parameter. If the ACK payload is more than 15 byte in 2Mbps mode the ARD must be 500µS or more, if the ACK payload is more than 5byte in 1Mbps mode the ARD must be 500µS or more. In 250kbps mode (even when the payload is not in ACK) the ARD must be 500µS or more. Please see section 7.4.2 on page 33 for more information.
b. This is the time the PTX is waiting for an ACK packet before a retransmit is made. The PTX is in RX mode for 250µS (500µS in 250kbps mode) to wait for address match. If the address match is detected, it stays in RX mode to the end of the packet, unless ARD elapses. Then it goes to standby-II mode for the rest of the specified ARD. After the ARD it goes to TX mode and then retransmits the packet.
c. The RX_DR IRQ is asserted by a new packet arrival event. The procedure for handling this interrupt should be: 1) read payload through SPI, 2) clear RX_DR IRQ, 3) read FIFO_STATUS to check if there are more payloads available in RX FIFO, 4) if there are more data in RX FIFO, repeat from step 1).
d. If ACK packet payload is activated, ACK packets have dynamic payload lengths and the Dynamic Payload Length feature should be enabled for pipe 0 on the PTX and PRX. This is to ensure that they receive the ACK packets with payloads. If the ACK payload is more than 15 byte in 2Mbps mode the ARD must be 500µS or more, and if the ACK payload is more than 5 byte in 1Mbps mode the ARD must be 500µS or more. In 250kbps mode (even when the payload is not in ACK) the ARD must be 500µS or more.

Table 28. Register map of nRF24L01+

10 Peripheral RF Information

This chapter describes peripheral circuitry and PCB layout requirements that are important for achieving optimum RF performance from the nRF24L01+.

10.1 Antenna output

The **ANT1** and **ANT2** output pins provide a balanced RF output to the antenna. The pins must have a DC path to **VDD_PA**, either through a RF choke or through the center point in a balanced dipole antenna. A load of 15Ω+j88Ω is recommended for maximum output power (0dBm). Lower load impedance (for instance, 50Ω) can be obtained by fitting a simple matching network between the load and **ANT1** and **ANT2**. A recommended matching network for 50Ω load impedance is described in chapter 11 on page 66.

10.2 Crystal oscillator

A crystal used with the nRF24L01+ must fulfil the specifications in Table 11. on page 19.

To achieve a crystal oscillator solution with low power consumption and fast start up time use a crystal with a low load capacitance specification. A lower C_0 also gives lower current consumption and faster start up time, but can increase the cost of the crystal. Typically C_0=1.5pF at a crystal specified for C_{0max}=7.0pF.

The crystal load capacitance, C_L, is given by:

$C_L = \frac{C_1' \cdot C_2'}{C_1' + C_2'}$, where $C_1' = C_1 + C_{PCB1} + C_{I1}$ and $C_2' = C_2 + C_{PCB2} + C_{I2}$

C_1 and C_2 are SMD capacitors, see the application schematics in Figure 32. on page 66. C_{PCB1} and C_{PCB2} are the layout parasitic on the circuit board. C_{I1} and C_{I2} are the internal capacitance load of the **XC1** and **XC2** pins respectively; the value is typically 1pF for both these pins.

10.3 nRF24L01+ crystal sharing with an MCU

Follow the rules described in sections 10.3.1 and 10.3.2 when using an MCU to drive the crystal reference input **XC1** of the nRF24L01+ transceiver.

10.3.1 Crystal parameters

The MCU sets the requirement of load capacitance C_L when it is driving the nRF24L01+ clock input. A frequency accuracy of ±60ppm is required to get a functional radio link. The nRF24L01+ loads the crystal by 1pF in addition to the PCB routing.

10.3.2 Input crystal amplitude and current consumption

The input signal should not have amplitudes exceeding any rail voltage. Exceeding rail voltage excites the ESD structure and consequently, the radio performance degrades below specification. You must use an external DC block if you are testing the nRF24L01+ with a reference source that has no DC offset (which is usual with a RF source).

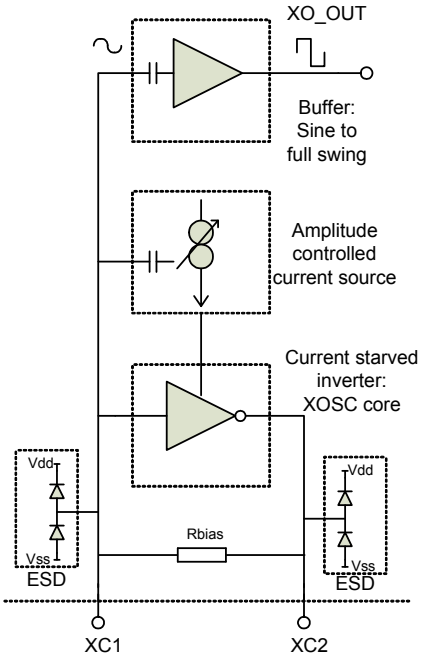


Figure 31. Principle of crystal oscillator

The nRF24L01+ crystal oscillator is amplitude regulated. It is recommended to use an input signal larger than 0.4V-peak to achieve low current consumption and good signal-to-noise ratio when using an external clock. **XC2** is not used and can be left as an open pin when clocked externally.

10.4 PCB layout and decoupling guidelines

A well designed PCB is necessary to achieve good RF performance. A poor layout can lead to loss of performance or functionality. You can download a fully qualified RF layout for the nRF24L01+ and its surrounding components, including matching networks, from www.nordicsemi.no.

A PCB with a minimum of two layers including a ground plane is recommended for optimum performance. The nRF24L01+ DC supply voltage should be decoupled as close as possible to the **VDD** pins with high performance RF capacitors, see Table 29. on page 67. Mounting a large surface mount capacitor (for example, 4.7µF ceramic) in parallel with the smaller value capacitors is recommended. The nRF24L01+ supply voltage should be filtered and routed separately from the supply voltages of any digital circuitry.

Avoid long power supply lines on the PCB. All device grounds, **VDD** connections and **VDD** bypass capacitors must be connected as close as possible to the nRF24L01+ IC. The **VSS** pins should be connected directly to the ground plane for a PCB with a topside RF ground plane. We recommend having via holes as close as possible to the **VSS** pads for a PCB with a bottom ground plane. A minimum of one via hole should be used for each **VSS** pin.

Full swing digital data or control signals should not be routed close to the crystal or the power supply lines. The exposed die attach pad is a ground pad connected to the IC substrate die ground and is intentionally not used in our layouts. We recommend to keep it unconnected.

11 Application example

nRF24L01+ with single ended matching network crystal, bias resistor, and decoupling capacitors.

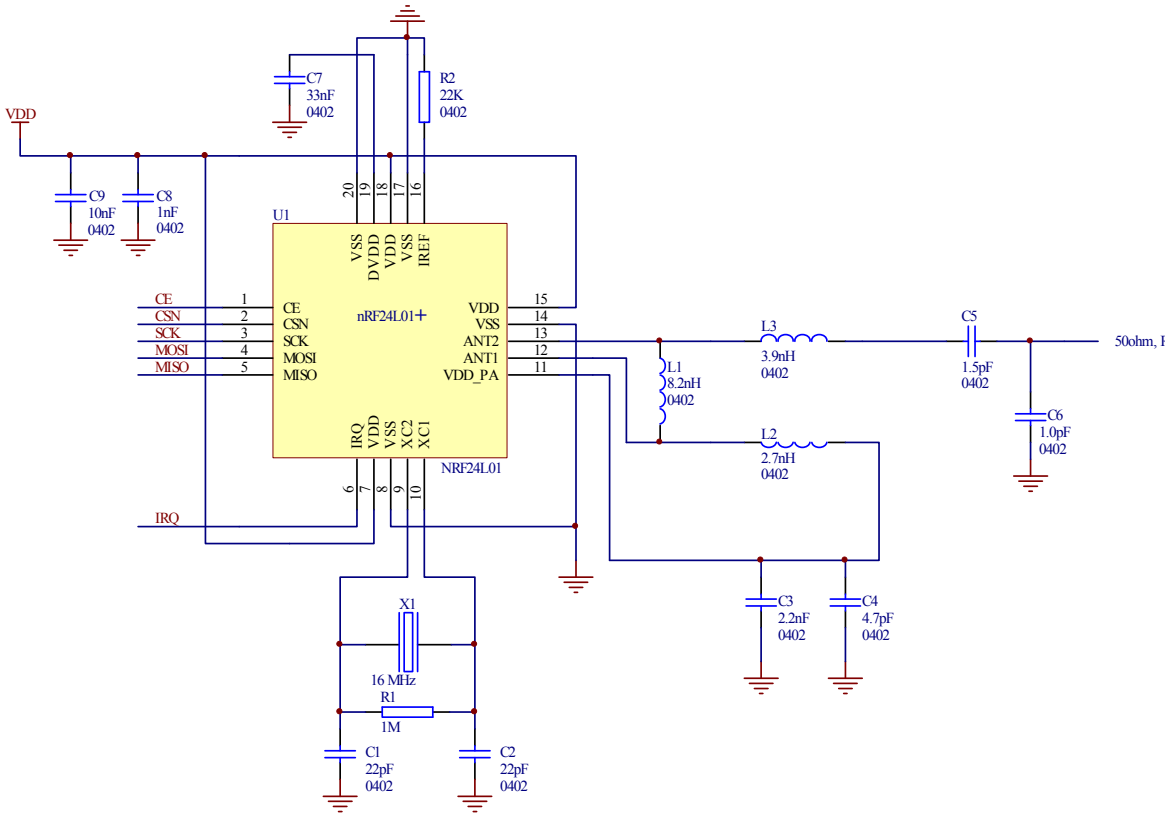


Figure 32. nRF24L01+ schematic for RF layouts with single ended 50Ω RF output

Part	Designator	Footprint	Description
22pF[a]	C1	0402	NPO, +/- 2%
22pF[a]	C2	0402	NPO, +/- 2%
2.2nF	C3	0402	X7R, +/- 10%
4.7pF	C4	0402	NPO, +/- 0.25pF
1.5pF	C5	0402	NPO, +/- 0.1pF
1,0pF	C6	0402	NPO, +/- 0.1pF
33nF	C7	0402	X7R, +/- 10%
1nF	C8	0402	X7R, +/- 10%
10nF	C9	0402	X7R, +/- 10%
8,2nH	L1	0402	chip inductor +/- 5%
2.7nH	L2	0402	chip inductor +/- 5%
3,9nH	L3	0402	chip inductor +/- 5%
Not mounted[b]	R1	0402	
22kΩ	R2	0402	+/-1%
nRF24L01+	U1	QFN20 4x4	
16MHz	X1		+/-60ppm, C_L=12pF

a. C1 and C2 must have values that match the crystals load capacitance, C_L.
b. The nRF24L01+ and nRF24L01 application example and BOM are the same with the exception of R1. R1 can be mounted for backward compatibility with nRF24L01. The use of a 1Mohm resistor externally does not have any impact on crystal performance.

Table 29. Recommended components (BOM) in nRF24L01+ with antenna matching network

11.1 PCB layout examples

Figure 33., Figure 34. and Figure 35. show a PCB layout example for the application schematic in Figure 32..

A double-sided FR-4 board of 1.6mm thickness is used. This PCB has a ground plane on the bottom layer. Additionally, there are ground areas on the component side of the board to ensure sufficient grounding of critical components. A large number of via holes connect the top layer ground areas to the bottom layer ground plane.

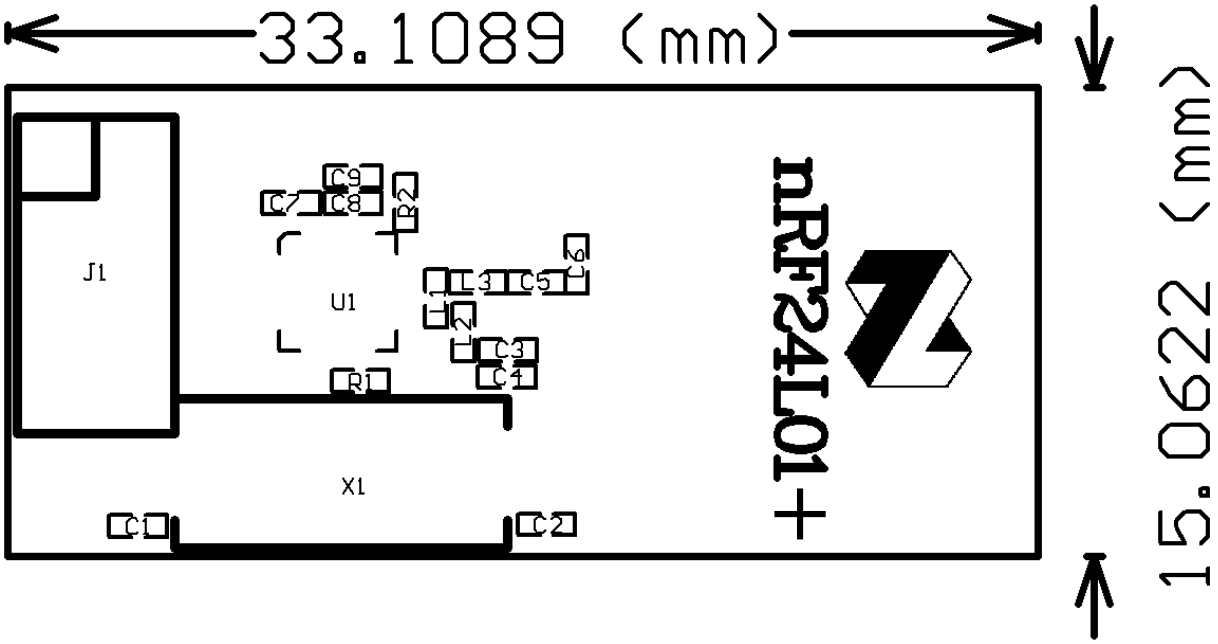


Figure 33. Top overlay (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components)

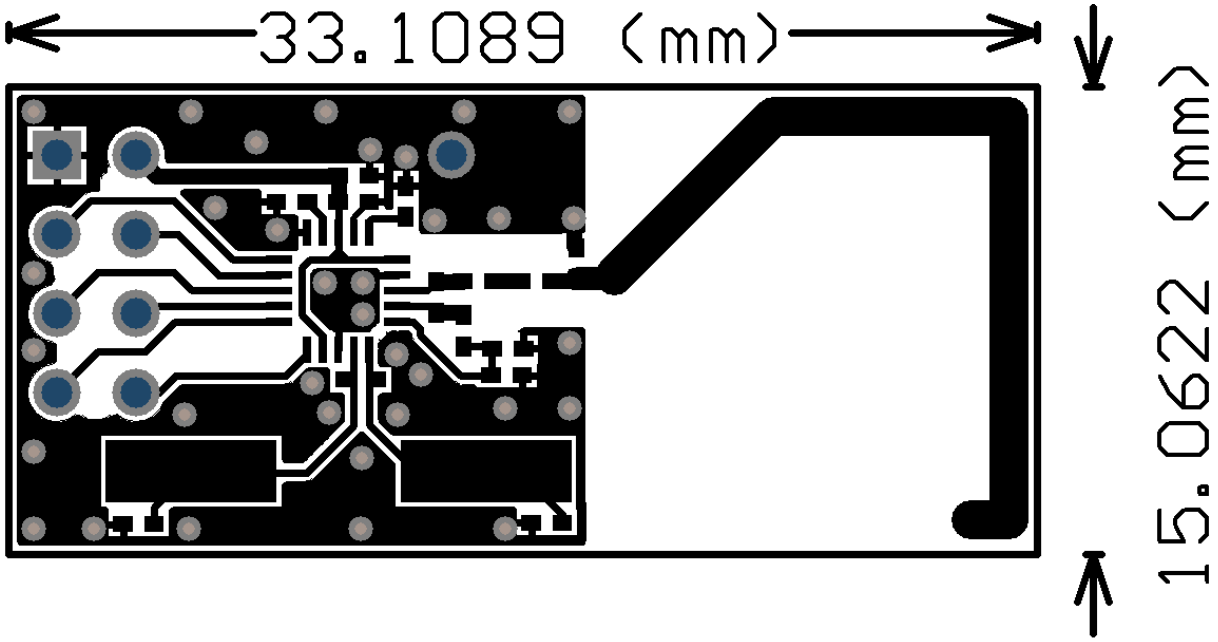


Figure 34. Top layer (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components)

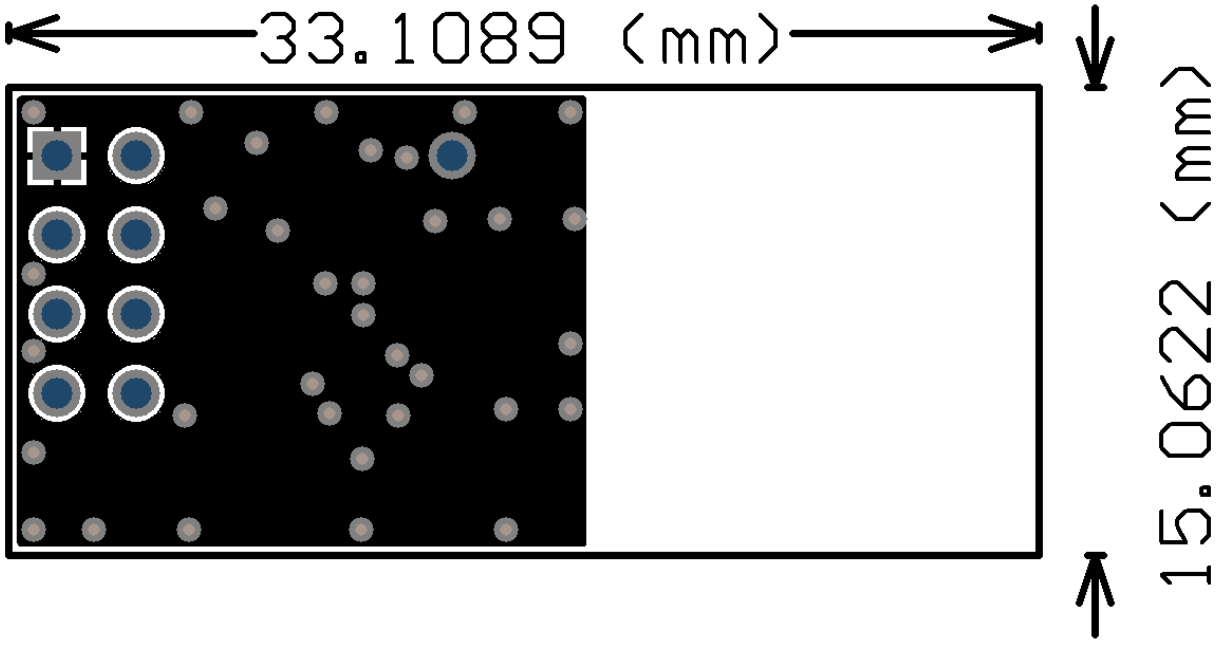


Figure 35. Bottom layer (nRF24L01+ RF layout with single ended connection to PCB antenna and 0402 size passive components

The nest figure (Figure 36., Figure 37. and Figure 38.) is for the SMA output to have a board for direct measurements at a 50Ω SMA connector.

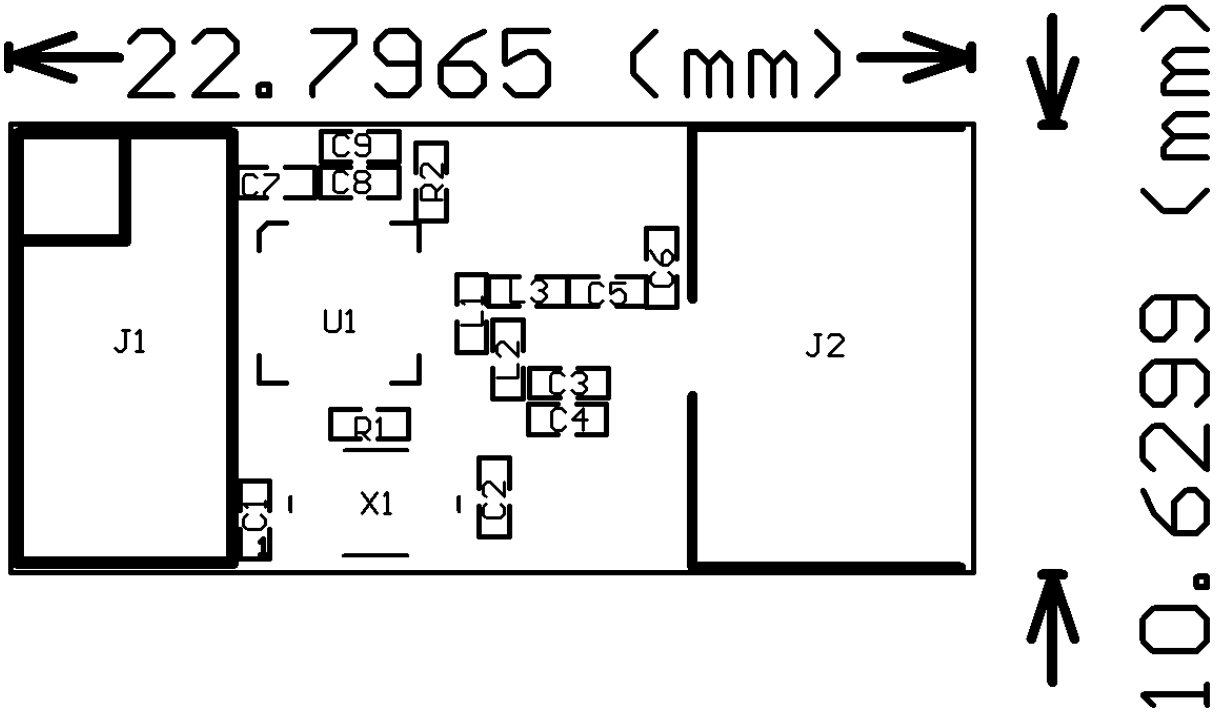
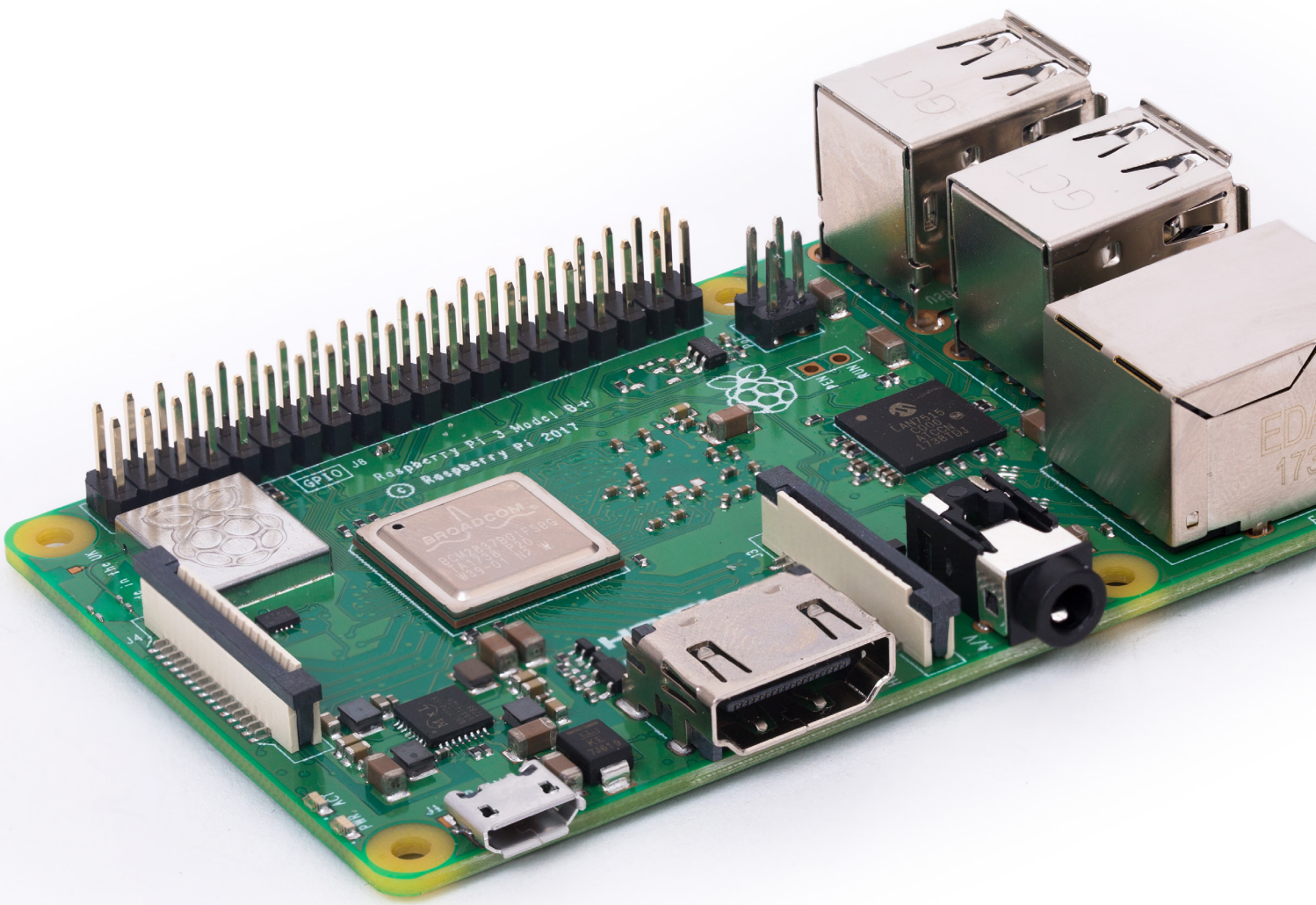
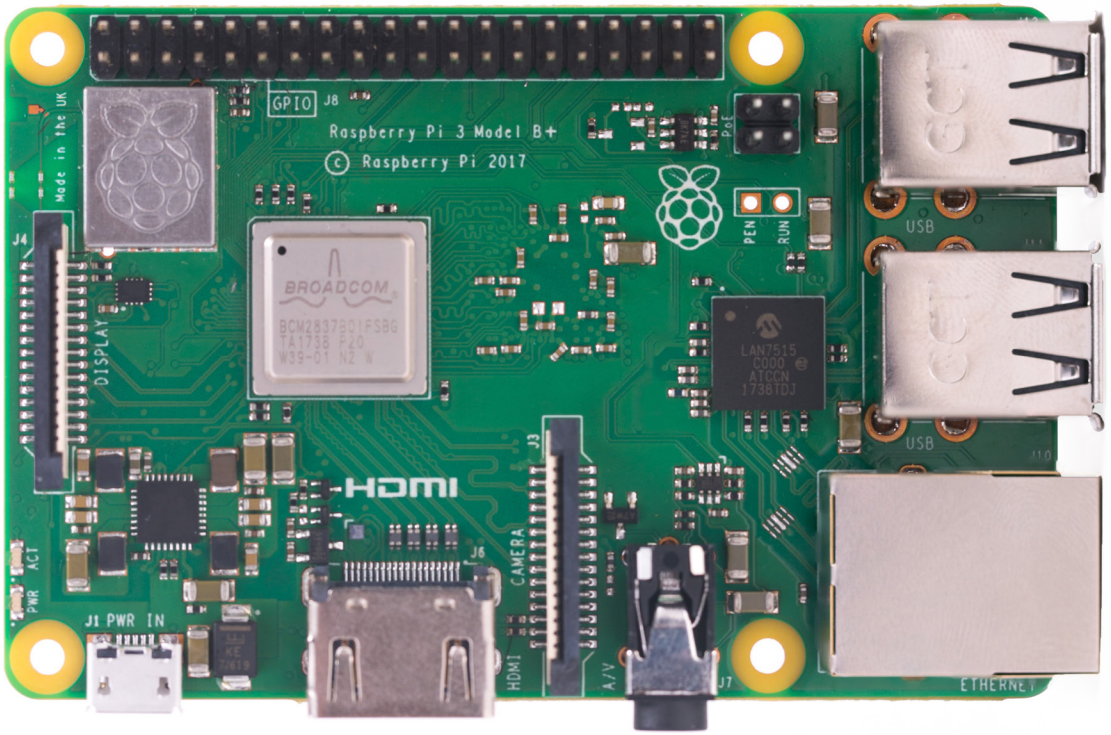


Figure 36. Top Overlay (Module with OFM crystal and SMA connector)

Raspberry Pi 3 Model B+



Overview



The Raspberry Pi 3 Model B+ is the latest product in the Raspberry Pi 3 range, boasting a 64-bit quad core processor running at 1.4 GHz, dual-band 2.4 GHz and 5 GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and PoE capability via a separate PoE HAT

The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market.

The Raspberry Pi 3 Model B+ maintains the same mechanical footprint as both the Raspberry Pi 2 Model B and the Raspberry Pi 3 Model B.

Specifications

Processor:
Broadcom BCM2837B0, Cortex-A53
64-bit SoC @ 1.4 GHz

Memory:
1GB LPDDR2 SDRAM

Connectivity:
- 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- 4 × USB 2.0 ports

Access:
Extended 40-pin GPIO header

Video & sound:
- 1 × full size HDMI
- MIPI DSI display port
- MIPI CSI camera port
- 4 pole stereo output and composite video port

Multimedia:
H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics

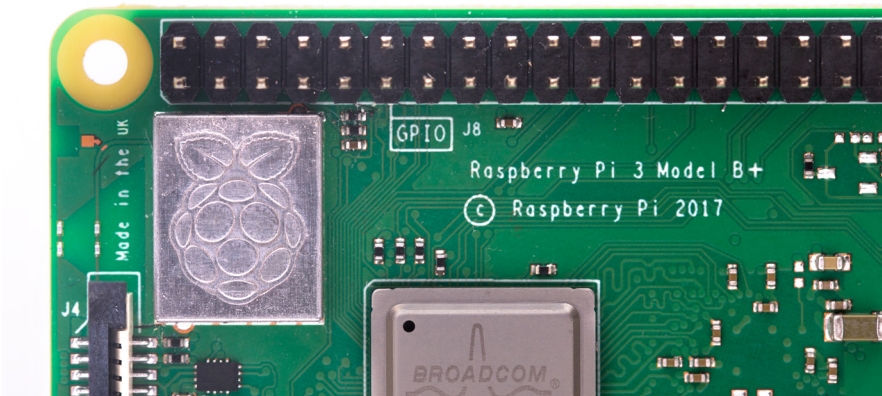
SD card support:
Micro SD format for loading operating system and data storage

Input power:
- 5 V/2.5 A DC via micro USB connector
- 5 V DC via GPIO header
- Power over Ethernet (PoE)–enabled (requires separate PoE HAT)

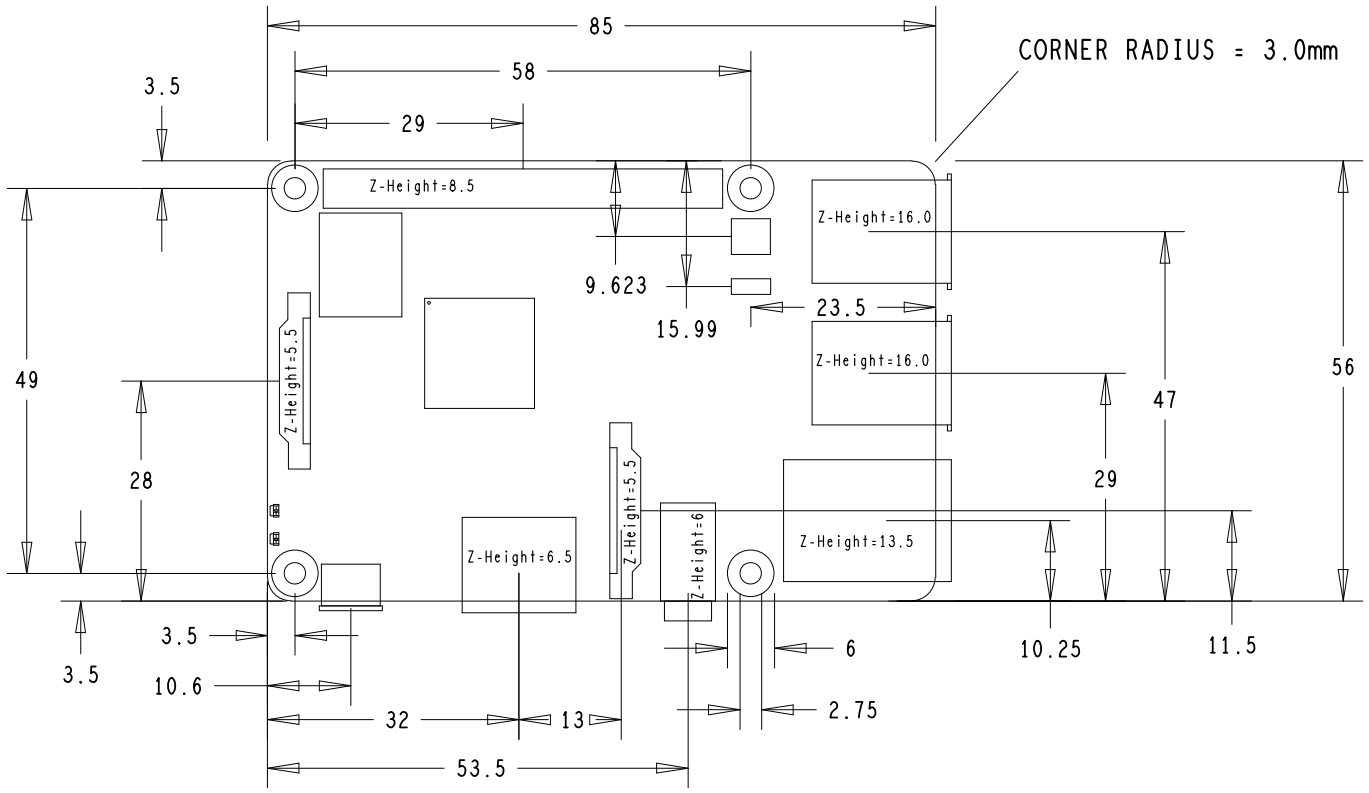
Environment:
Operating temperature, 0–50 °C

Compliance:
For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+

Production lifetime:
The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.



Physical specifications



Warnings

- This product should only be connected to an external power supply rated at 5 V/2.5 A DC. Any external power supply used with the Raspberry Pi 3 Model B+ shall comply with relevant regulations and standards applicable in the country of intended use.
- This product should be operated in a well-ventilated environment and, if used inside a case, the case should not be covered.
- Whilst in use, this product should be placed on a stable, flat, non-conductive surface and should not be contacted by conductive items.
- The connection of incompatible devices to the GPIO connection may affect compliance, result in damage to the unit, and invalidate the warranty.
- All peripherals used with this product should comply with relevant standards for the country of use and be marked accordingly to ensure that safety and performance requirements are met. These articles include but are not limited to keyboards, monitors, and mice when used in conjunction with the Raspberry Pi.
- The cables and connectors of all peripherals used with this product must have adequate insulation so that relevant safety requirements are met.

Safety instructions

To avoid malfunction of or damage to this product, please observe the following:

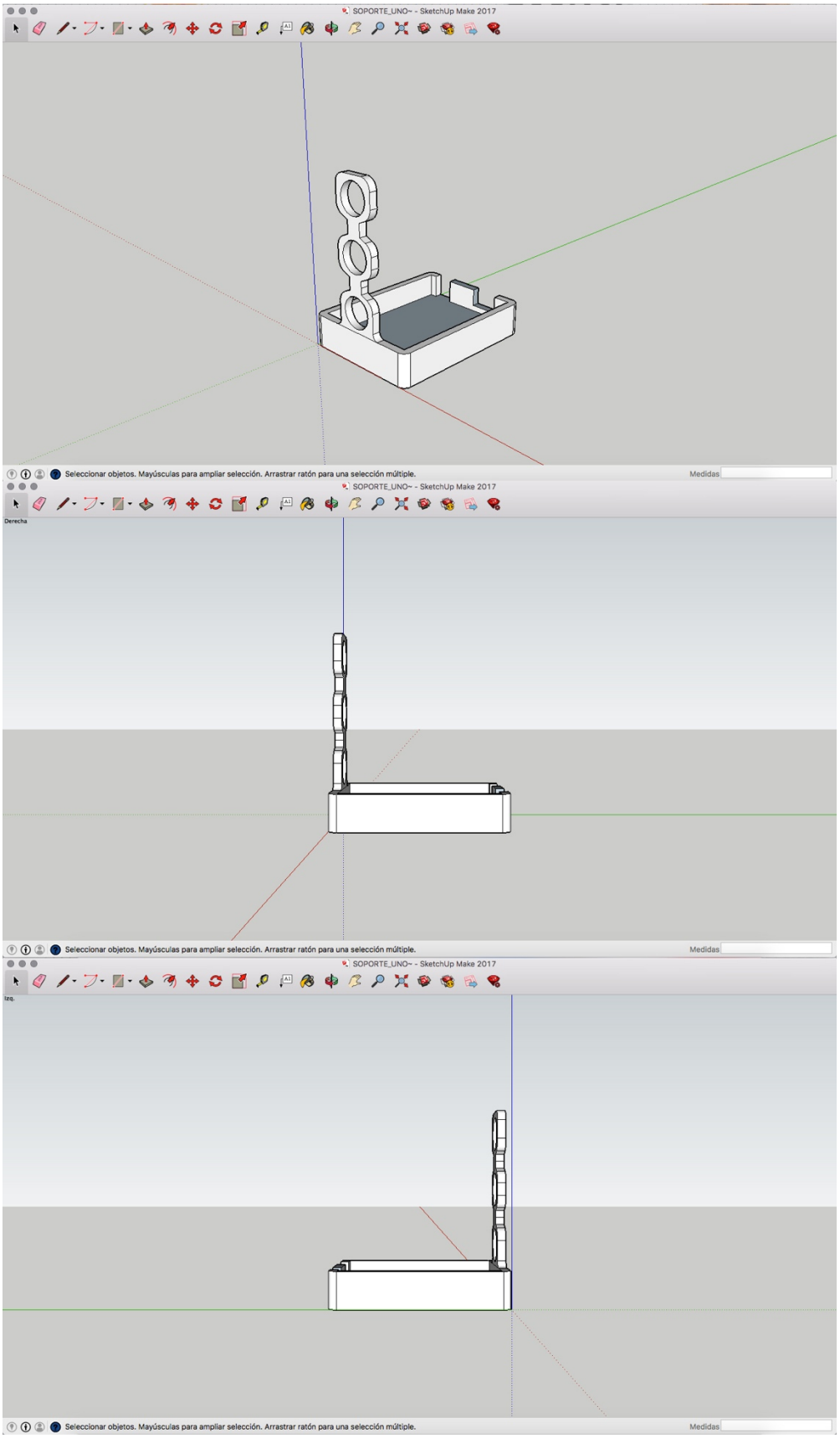
- Do not expose to water or moisture, or place on a conductive surface whilst in operation.
- Do not expose to heat from any source; the Raspberry Pi 3 Model B+ is designed for reliable operation at normal ambient temperatures.
- Take care whilst handling to avoid mechanical or electrical damage to the printed circuit board and connectors.
- Whilst it is powered, avoid handling the printed circuit board, or only handle it by the edges to minimise the risk of electrostatic discharge damage.

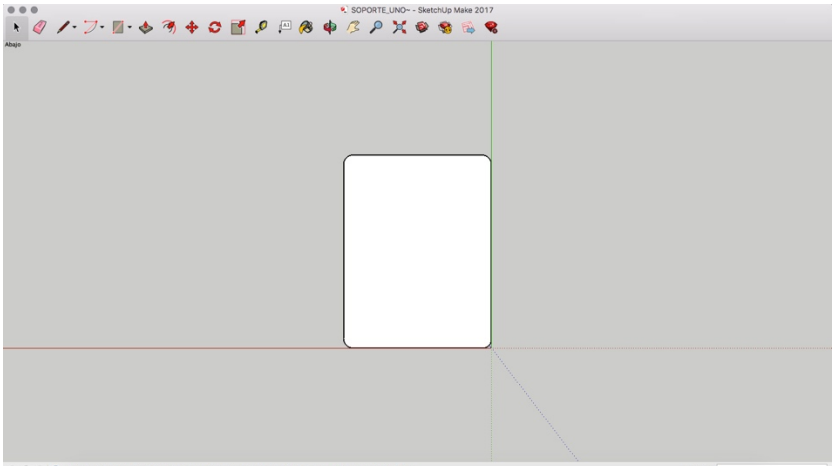
Anexo C.

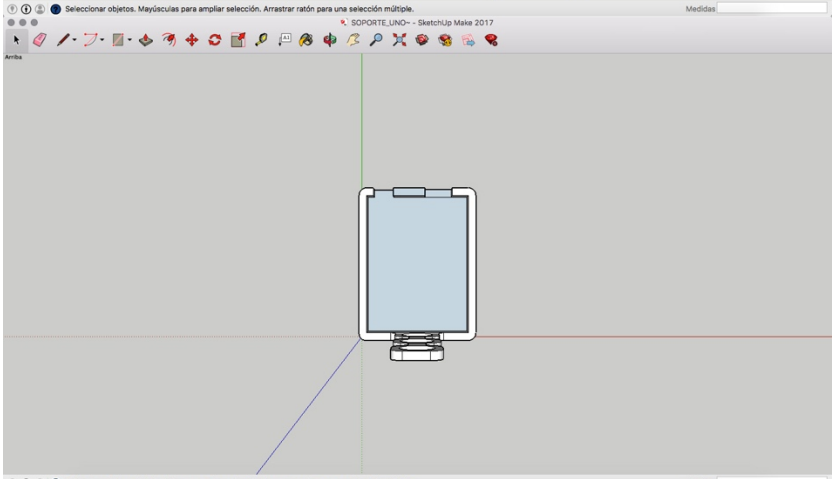
Diseños 3D

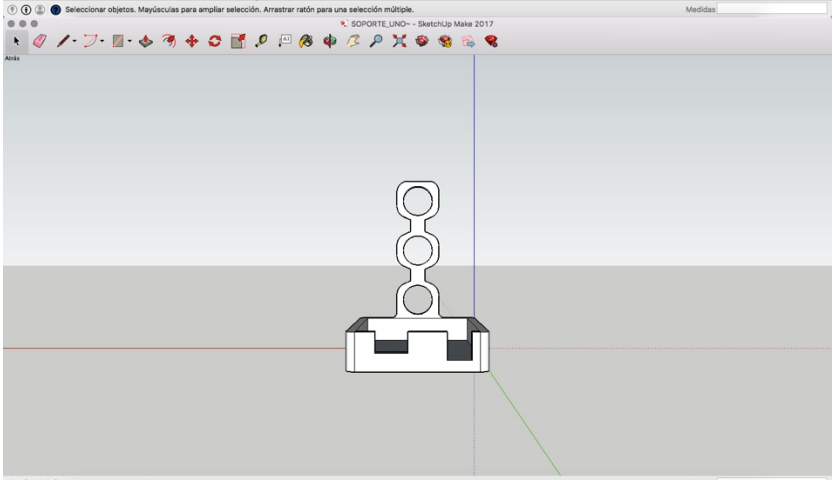
En este anexo se recogen los diseños realizados en el programa SketchUp para la implementación en ciertas partes del proyecto. Los diseños han sido impresos en PLA gracias a una impresora 3D como se ha indicado a lo largo de la memoria.

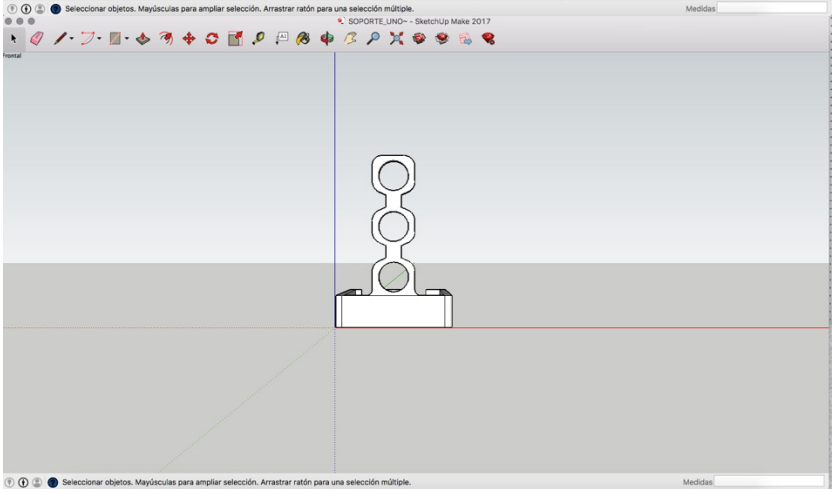
C.1 Diseño para el emplazamiento de los objetos emisores



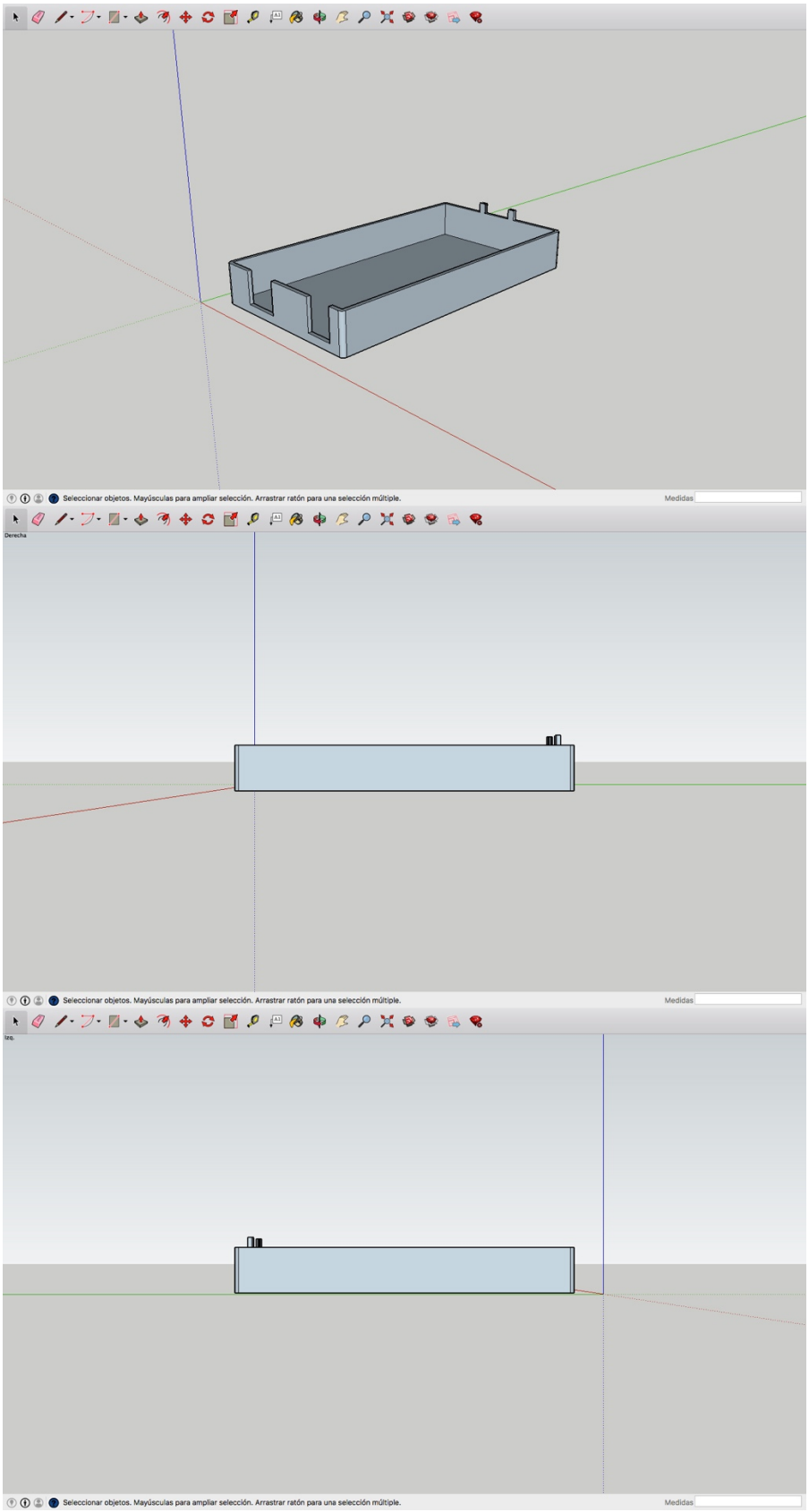

SOPORTE_UNO~ - SketchUp Make 2017
Abajo
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

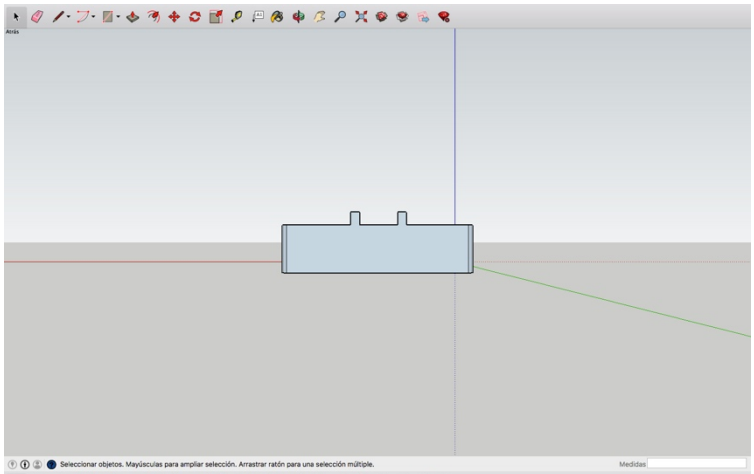

SOPORTE_UNO~ - SketchUp Make 2017
Arriba
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

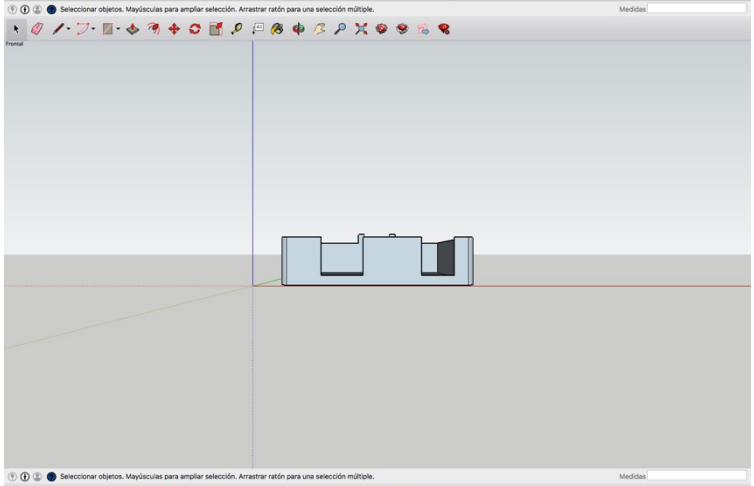

SOPORTE_UNO~ - SketchUp Make 2017
Atrás
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

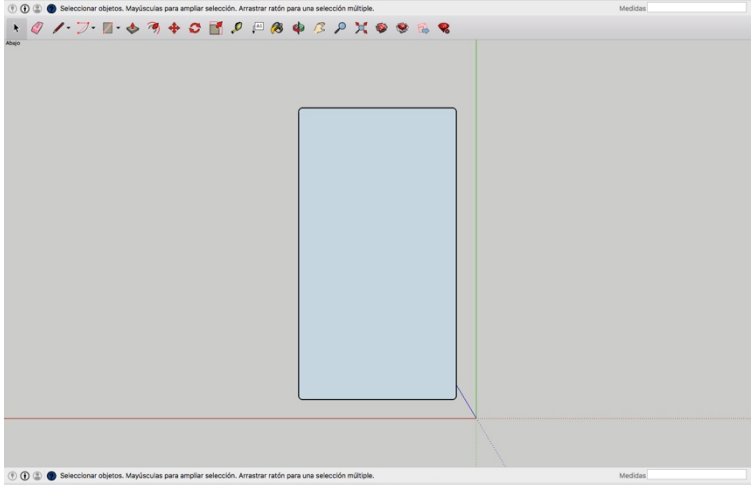

SOPORTE_UNO~ - SketchUp Make 2017
Frontal
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

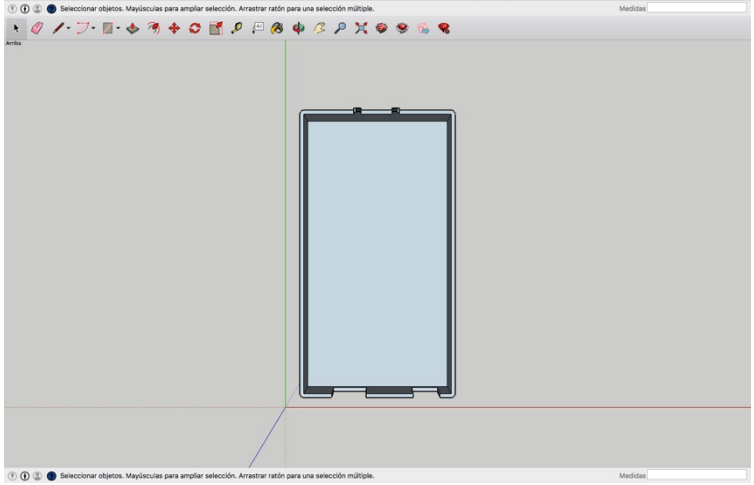
C.2 Diseño para el emplazamiento de Arduino MEGA



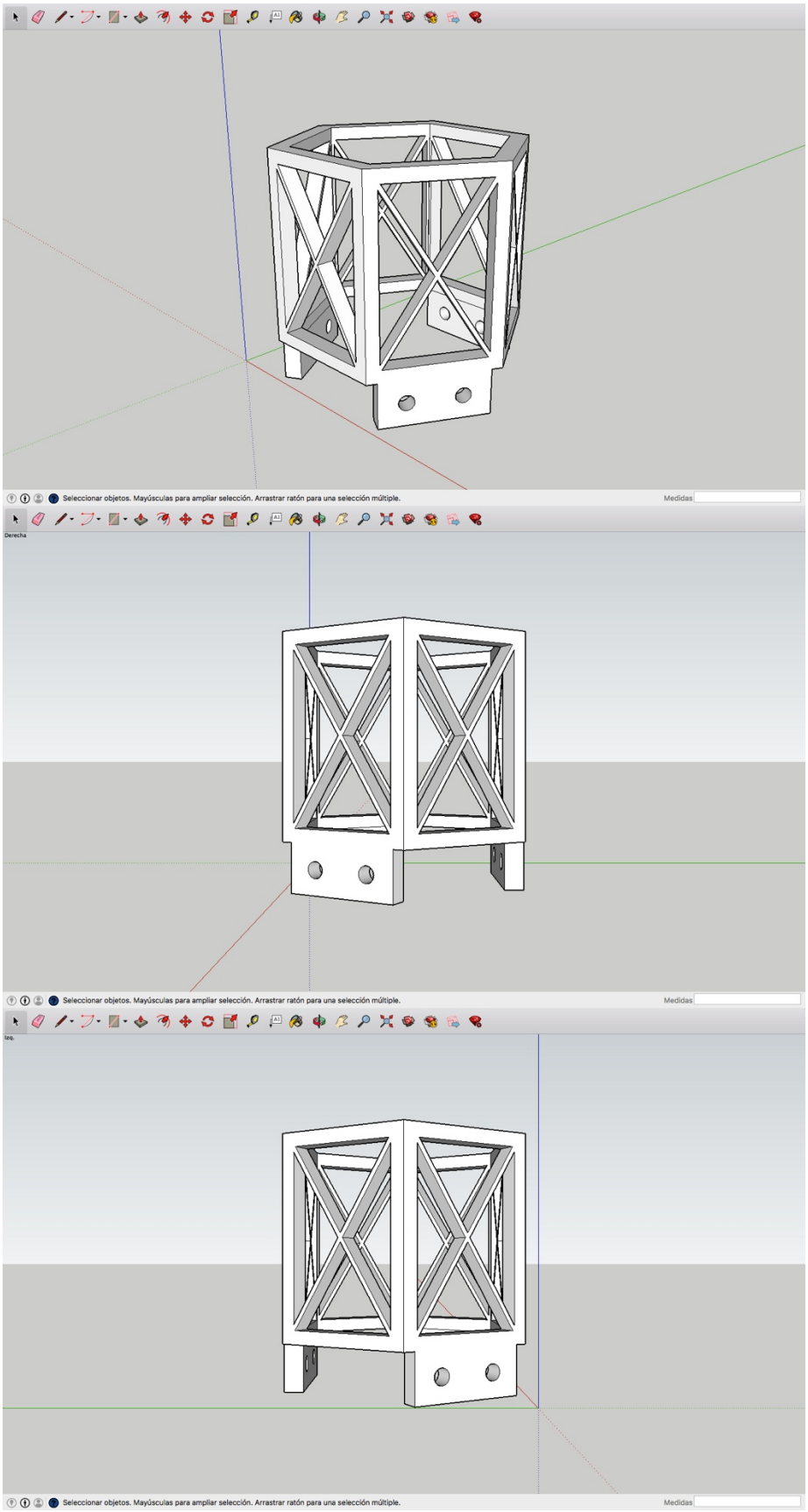

Atrás
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

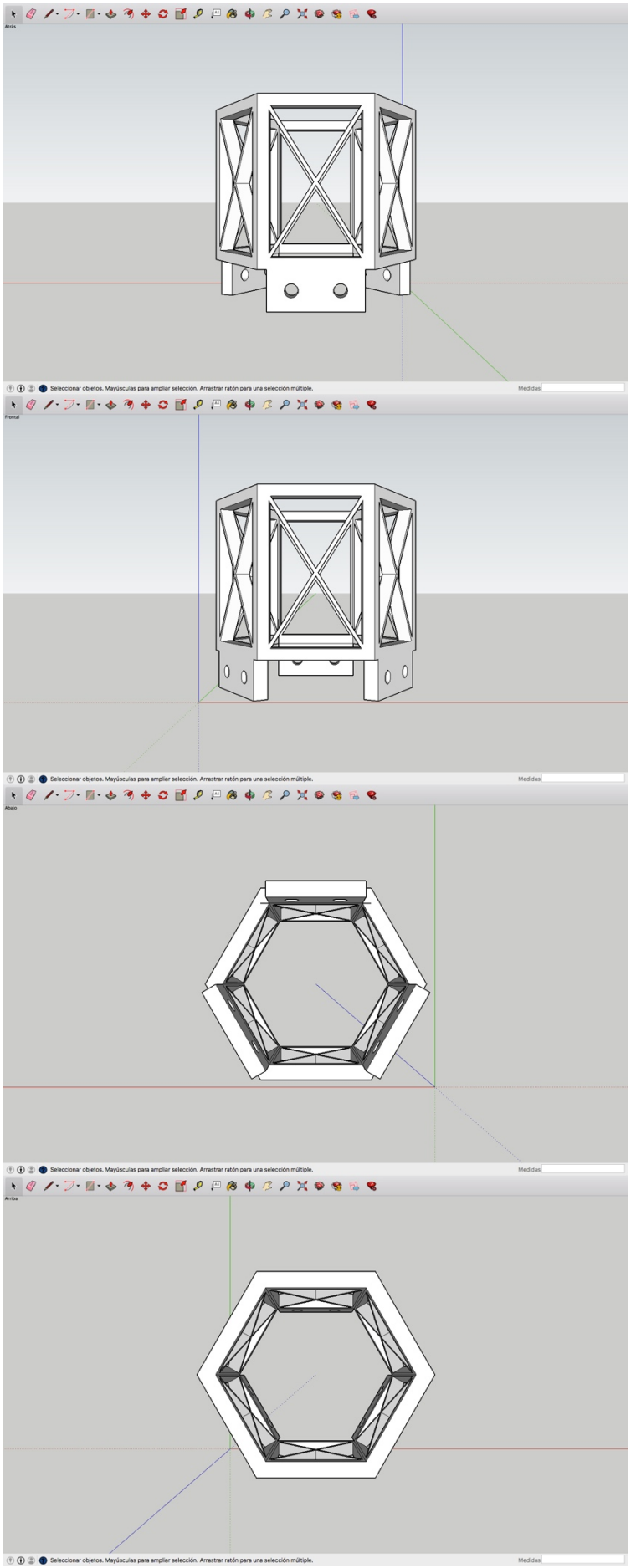

Frontal
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas


Abajo
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas


Arriba
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

C.3 Diseño para el emplazamiento del objeto receptor




Arriba
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas
Frontal
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas
Abajo
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas
Arriba
Seleccionar objetos. Mayúsculas para ampliar selección. Arrastrar ratón para una selección múltiple.
Medidas

Anexo D.

Bibliografía

En este anexo se expone la bibliografía utilizada para el correcto desarrollo del proyecto, así como artículos de interés para la comprobación de teorías y métodos utilizados.

D.1 Localización y Trilateración

[1] Sistemas de Localización en Tiempo Real.
https://es.wikipedia.org/wiki/Sistemas_de_Localización_en_Tiempo_Real

[2] Triangulación y Trilateración.
http://ocw.upm.es/ingenieria-cartografica-geodesica-y-fotogrametria/topografia-ii/contenidos/Mis_documentos/Tema-9-Triangulacion-y-Trilateracion/Teoria_Triang_Tema_9.pdf

[3] Find X location using 3 known (X,Y) location using trilateration
https://math.stackexchange.com/questions/884807/find-x-location-using-3-known-x-y-location-using-trilateration

[4] Modelos Determinísticos y Probabilísticos
http://proyectoeypii.blogspot.com/

[5] Trilateración
https://es.wikipedia.org/wiki/Trilateración

[6] Localización
http://www.kramirez.net/Robotica/Material/Presentaciones/Localizacion.pdf

D.2 Módulo Ultrasonido HC-SR04

[7] ¿Cómo medir distancias con el sensor de ultrasonidos HC-SR04 con Arduino?
http://robotica.webs.upv.es/es/como-medir-distancias-con-el-sensor-de-ultrasonidos-hc-sr04-con-arduino/

[8] Medir distancias con Arduino y sensor de Ultrasonido HC-SR04
https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/

[9] Complete Guide for Ultrasonic Sensor HC-SR04
https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/

D.3 Arduino

[10] Primeros pasos con Arduino
http://arduino.cl/primeros-pasos-con-arduino/

[11] Tutoriales de Arduino
https://www.luisllamas.es/tutoriales-de-arduino/

[12] rosserial library for Arduino
https://github.com/ros-drivers/rosserial

[13] RF24L01 library for Arduino
https://github.com/maniacbug/RF24

[14] TimerThree library for Arduino
https://github.com/PaulStoffregen/TimerThree

D.4 Módulo radiofrecuencia RF24L01

[15] Comunicación inalámbrica a 2.4GHz con Arduino y NRF24L01
https://www.luisllamas.es/comunicacion-inalambrica-a-2-4ghz-con-arduino-y-nrf24l01/

[16] Tutorial básico NRF24L01 con Arduino
https://naylampmechatronics.com/blog/16_tutorial-basico-nrf24l01-con-arduino.html

[17] millis(). Arduino Multitasking
https://www.baldengineer.com/millis-tutorial.html

[18] Aprendiendo Arduino. Interrupciones.
https://aprendiendoarduino.wordpress.com/tag/isr/

[19] Módulos de radio NRF24L01
https://www.prometec.net/nrf2401/

D.5 Raspberry Pi3

[20] Install Raspbian for Robots on an SD Card for the Raspberry Pi
https://www.dexterindustries.com/howto/install-raspbian-for-robots-image-on-an-sd-card/

[21] Using your new Raspberry Pi3 as a WiFi Access point with Hostapd
https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/

[22] Install ROS in a Raspberry Pi with Raspbian Stretch
https://neverbenever.wordpress.com/2017/12/20/install-ros-and-opencv-in-raspberry-piraspbian-stretch/

[23] Arduino IDE Setup (rosserial)
http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino IDE Setup

[24] El robot Gopigo2
https://jjromeromarras.wordpress.com/2017/07/02/el-robot-gopigo-2/

D.6 Ubuntu y ROS

[25] Ubuntu Desktop
https://www.ubuntu.com/download/desktop

[26] Ubuntu install of ROS Kinetic
http://wiki.ros.org/kinetic/Installation/Ubuntu

[27] Manual de ROS
https://moodle2015-16.ua.es/moodle/mod/book/view.php?id=82546

[28] ROS introduction
https://erlerobotics.com/blog/ros-introduction-es/

D.7 Software utilizado

[29] Etcher
https://etcher.io/

[30] Arduino
http://arduino.cl/descargas/

[31] Fritzing
http://fritzing.org/download/

[32] Ultimaker Cura
https://ultimaker.com/en/products/ultimaker-cura-software

[33] SketchUp
https://www.sketchup.com/es/download

[34] VMware Fusion
https://www.vmware.com/products/fusion/fusion-evaluation.html?ClickID=bkzqzf61k1e1mmyyknfqumqvyfqmvfdevgne