TRABAJO FINAL DE GRADO

Universitat Politécnica de Catalunya

Proyecto de diseño de un sistema de comunicación óptico entre 2 cubesats

Grado en Ingeniería electrónica industrial y automática

Autor:
Alejandro Cáceres Gómez
Director:
Javier Gago Barrio

Co-Director:
Manuel Lamich Arocas

30 de Junio 2020

Declaración de honor

Yo, Alejandro Cáceres Gómez, declaro que el trabajo en este TFG es completamente obra mía, que ninguna parte de este trabajo final de grado ha sido copiada de otras personas sin dar la acreditación correspondiente, y que todas las referencias han sido citadas claramente. Entiendo que cualquier infracción de esta declaración, me deja sometido a acciones disciplinarias previstas por la Universidad Politécnica de Cataluña-Barcelona TECH.

Alejandro Cáceres Gómez 30/06/2020

RESUMEN

Este proyecto, forma parte de un proyecto conjunto con otros proyectistas de final de grado en el que se pretende diseñar e implementar una red de nanosatelites en órbita terrestre baja (OTB) con el fin de conseguir la transferencia de datos mejorando la velocidad, cobertura y fiabilidad de las comunicaciones.
En este contexto, el presente proyecto trata de diseñar e implementar a nivel experimental el software necesario para comunicar dos prototipos de cubesats mediante distintas modulaciones de señal y el diseño de un protocolo propio en la capa física. Para ello, se dispone del microcontrolador ARM STM32F103C8 (Emisor) y una placa de desarrollo Arduino UNO (Receptor), en dichas placas, se diseñará el código fuente para que sean capaces de enviar mensajes entre ellas. Inicialmente, en el alcance de este proyecto también se preveía implementar el código y los protocolos de comunicación para crear un enlace óptico utilizando pines GPIO de ambas placas, pero finalmente (y debido a las restricciones dada la situación actual en el momento del desarrollo de este proyecto (COVID-19)) se ha decidido realizar la comunicación cableada y prepararla para una implementación con la óptica de forma sencilla, simplemente integrando el hardware necesario y utilizando exactamente los mismos pines que en su versión cableada.

RESUM

Aquest projecte, forma part d'un projecte conjunt amb altres projectistes de final de grau en el qual es pretén dissenyar i implementar una xarxa de nanosatélits en òrbita terrestre baixa (OTB) amb la finalitat d'aconseguir la transferència de dades millorant la velocitat, cobertura i fiabilitat de les comunicacions.
En aquest context, el present projecte tracta de dissenyar i implementar a nivell experimental el programari necessari per a comunicar dos prototips de cubesats mitjançant diferents modulacions de senyal i el disseny d'un protocol propi en la capa física. Per a això, es disposa del microcontrolador ARM STM32F103C8 (Emissor) i una placa de desenvolupament Arduino UN (Receptor), en aquestes plaques, es dissenyarà el codi font perquè siguin capaces d'enviar missatges entre elles. Inicialment, en l'abast d'aquest projecte també es preveia implementar el codi i els protocols de comunicació per a crear un enllaç òptic utilitzant pins GPIO de totes dues plaques, però finalment (i a causa de les restriccions donada la situació actual en el moment del desenvolupament d'aquest projecte (COVID-19)) s'ha decidit realitzar la comunicació cablejada i preparar-la per a una implementació amb l'òptica de manera senzilla, simplement integrant el Hardware necessari i utilitzant exactament els mateixos pins que en la seva versió cablejada.

ABSTRACT

This project is part of a joint project with other final degree designers in which it is intended to design and implement a network of nanosatellites in low earth orbit (OTB) in order to achieve data transfer by improving speed, coverage and reliability of communications.
In this context, the present project tries to design and implement on an experimental level the software necessary to communicate two cubesats prototypes through different signal modulations and the design of an own protocol in the physical layer. For this, the ARM STM32F103C8 microcontroller (Emitter) and an Arduino UNO development board (Receiver) are available, on these boards, the source code will be designed so that they are capable of sending messages between them. Initially, the scope of this project also envisaged implementing the code and communication protocols to create an optical link using GPIO pins on both boards, but finally (and due to restrictions given the current situation at the time of development of this project (COVID-19)) it has been decided to carry out the wired communication and prepare it for an implementation with the optics in a simple way, simply integrating the necessary hardware and using exactly the same pins as in its wired version.

Índice

Índice de figuras

1. Introducción

El cubesat que se esta diseñando y desarrollando por todos los alumnos que forman parte del proyecto conjunto, se pretende que sea un cubesat modular. Lo cual, significa que se deben independizar, tanto a nivel físico como a nivel funcional, las diferentes áreas de desarrollo del satélite (potencia, comunicación, posicionamiento... etc), por ello, en el presente TFG se asume que el satélite dispondrá de un módulo de radiofrecuencia que enviará datos de su posición y actitud al satélite con el que se quiere comunicar.
Teniendo en cuenta esto, el flujo del proceso de comunicación entre dos cubesats deriva en los siguientes puntos:

1. Envío de datos de posición y actitud del emisor al receptor mediante una señal de radiofrecuencia (módulo RF).

2. Posicionamiento y corrección de ángulos de inclinación por parte del receptor o emisor (o ambos) para conseguir el alineamiento con el menor margen de error posible.

3. Comprobación de que los datos de posición y actitud de ambos satélites coinciden (están perfectamente alineados)

4. **Inicio del enlace óptico.**

5. **Transferencia de datos.**

6. Fin de la comunicación.

   En los puntos señalados del proceso anterior, es en los que se centrará este proyecto, mediante la emulación cableada entre dos placas de desarrollo: Arduino Uno actuando como el cubesat receptor de datos y STM32F103C8 actuando como el emisor de los datos.
   Estos datos van a ser enviados mediante 3 sistemas de comunicación distintos:

   - Modulación por posición de pulso (PPM)
   - Modulación por ancho de pulso (PWM)
   - Protocolo UART

   Se puede ver gráficamente el punto de actuación de la comunicación en la siguiente representación:

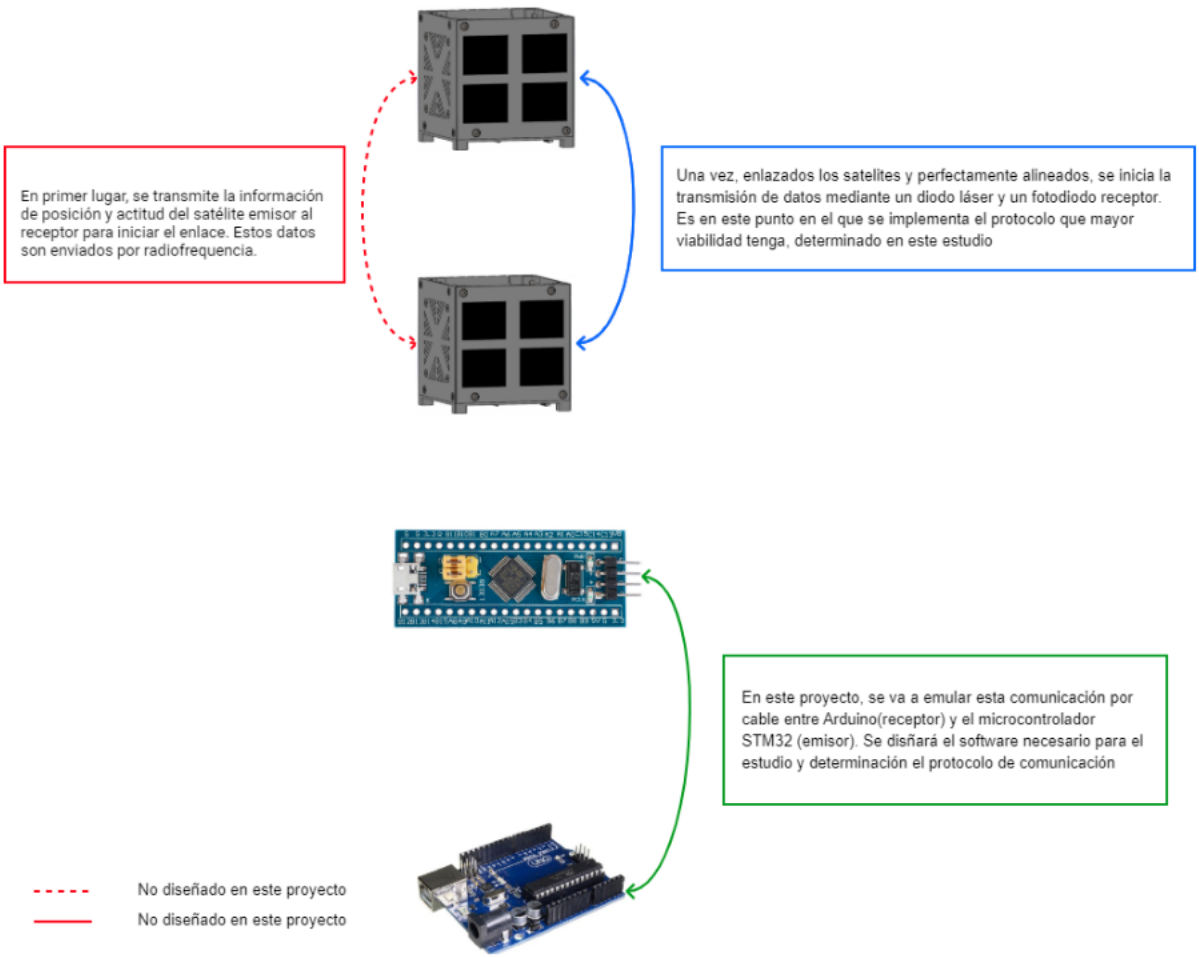

Figura 1: Representación gráfica del proceso de comunicación

1.1. Objeto del proyecto

El proyecto forma parte de un conjunto de proyectos desempeñados por estudiantes de distintas áreas de ingeniería. El objetivo final es diseñar un nano-satélite o cubesat capaz de comunicarse y enviar datos con otros cubesats con el fin de formar una red satelital que permita la transmisión óptima de información y proveer de esa conexión intersatelital al usuario para proporcionar una cobertura de comunicaciones mucho más amplía que tecnologías utilizadas actualmente.
Específicamente, este proyecto, tratará sobre el estudio de los distintos protocolos de comunicación o de onda modulada con el fin de determinar el sistema de transmisión de datos óptimo en cuanto a velocidad, fiabilidad y control de errores se refiere. Se determinará la viabilidad del envío de paquetes de información encapsulados con una trama de bits diseñada específicamente para este fin mediante; Modulación por posición de pulso (PPM), modulación por ancho de pulsos (PWM) y Transmisor-Receptor Asíncrono Universal (UART).
Junto con el proyecto de Pol Pla, cuyo objetivo es la implementación a nivel de hardware del sistema mediante un módulo de comunicaciones óptico que transmitirá los datos por un diodo láser que se enlazará con otro cubesat sincronizado, estos dos proyectos, forman la etapa de comunicaciones del cubesat final.

1.2. Alcance

- Estudio y desarrollo del emisor y receptor mediante envío de datos por PPM haciendo uso de la placa de desarrollo Arduino Uno y microcontrolador STM32F103C8. El programa se desarrollará en lenguaje C y módulos en C++.
- Estudio y desarrollo del emisor y receptor mediante envío de datos por PWM haciendo uso de la placa de desarrollo Arduino Uno y microcontrolador STM32F103C8. El programa se desarrollará en lenguaje C y módulos en C++.
- Estudio y desarrollo del emisor y receptor mediante envío de datos por UART haciendo uso de la placa de desarrollo Arduino Uno y microcontrolador STM32F103C8. El programa se desarrollará en lenguaje C y módulos en C++.
- Desarrollado cada uno de los sistemas de comunicación (PPM ,PWM, UART), de diseñará la trama de bits a enviar implementando; sincronismo, control de errores, datos a enviar, información del destino y origen.
- Realización del estudio de velocidad, fiabilidad de transmisión de datos, y cuantificación estadística de errores sucedidos entre emisor-receptor cuando se envían distintos paquetes de datos por conexión cableada entre Arduino Uno y el microcontrolador "Blue pill".

1.3. Estado del arte y contexto actual

En la última década, muchas empresas y organizaciones han llevado a cabo varios proyectos de constelaciones satélitales con el fin de desarrollar un entorno de comunicación global para el intercambio de información. Tanto es así, que ha llegado a causar cierta preocupación en instituciones astronómicas ya que estas constelaciones pueden suponer una amenaza significativa a estas infraestructuras.

> " La IAU es una organización de ciencia y tecnología que estimula y protege los avances en esas áreas. Aunque se ha realizado un esfuerzo significativo para mitigar los problemas con las diferentes constelaciones de satélites, recomendamos encarecidamente que todos los interesados en esta nueva frontera, en gran medida no regulada, de la utilización del espacio trabajen en colaboración para su beneficio mutuo. Las constelaciones de satélites pueden suponer una amenaza significativa o debilitante para las infraestructuras astronómicas importantes existentes y futuras, e instamos a sus diseñadores y desplegadores, así como a los responsables políticos, a trabajar con la comunidad astronómica en un esfuerzo concertado para analizar y comprender el impacto de las constelaciones de satélites. También instamos a las agencias apropiadas a que diseñen un marco regulatorio para mitigar o eliminar los impactos perjudiciales en la exploración científica tan pronto como sea práctico." — International Astronomical Union [5]

Sin embargo, y con la globalización mundial de la información se hace necesario abrir el horizonte de tecnologías para compartir dicha información y, en el paradigma actual, en que el dominio de la tecnología determina la supremacía empresarial, son muchas las organizaciones que priorizan este campo de investigación y desarrollo para llegar primeros al control y monopolio de nuevos

sistemas de intercambio de información ya que es un campo en auge constante. Prueba de ello son las diferentes iniciativas y proyectos llevados a cabo en los ultimos años, dejando a un lado los ya conocidos proyectos Starlink y One Web (que los mencionaré más adelante), se han desarrollado otras iniciativas en otras empresas mastodónticas del sector aeroespacial, destaco aquí algunos de ellos:

Constelación	**Fabricante**	**Año**	**Enlace**	**Estado actual**
Iridium Next	Thales Alenia + ATK orbital	2009	Radiofrecuencia	Completo
Boeing	Boeing Satellite	2016	–	Transferido a proyecto OneWeb
LeoSat	Thales Alenia	2015	Óptico	Primeros lanzamientos en 2021
O3b	Thales Alenia + Boeing	2017	–	En desarrollo
Telesat LEO	Airbus	2016	Óptico	Lanzamiento en 2018

Tabla 1: Constelaciones satelitales de internet más destacadas en los últimos años.[6]

Estas iniativas (y otras) demuestran que es un mercado emergente y, como se puede apreciar en el lapso de tiempo de desarrollo, estan a la vanguardia, cabe decir que tecnologías como el GPS, que usamos a diario utilizan una constelación satelital.
Como he mencionado anteriormente, StarLink y OneWeb son los proyectos más ambiciosos en este campo. StarLink es una iniciativa del físico sudafricano Elon Musk y pretende instalar una red de satelites que rodearían el planeta con el fin de brindar acceso a internet en cualquier lugar, sin necesidad de cableados ni instalaciones terrestres además de un muy bajo precio y con una latencia que puede ir a la par de la fibra óptica. Esta red, se situaría a partir de los 550km de altura, con lo que el impacto espacial sería mínimo pero no abandonaría la órbita terrestre baja (OTB), si este proyecto se finaliza e implementa con éxito es muy posible que sea la tecnología más utilizada para conectarse a la red.[7]
Por su parte, OneWeb es una compañía con sede en Londres dedicada al desarrollo de una constelación satelital, tambien en OTB, que pretende facilitar el acceso a internet a una mayor velocidad de transmisión. Actualmente ya han realizado 74 lanzamientos de los 600 previstos, cubriendo un alto espectro global y, demostrando velocidades de más de 400 Mbps con latencias de 32ms, además han experimentando una gran demanda de sus servicios de alta velocidad. Sin embargo, con a raíz de las fallidas negociaciones con su principal inversor e impulsado por la crisis del COVID-19, con fecha de 27 de Marzo de 2020, Adrian Steckel, actual CEO de OneWeb, anuncia que se hallan en trámites con el tribunal de cuentas de EEUU con el fin de evitar la bancarrota de la compañía, y , por lo tanto, sus actividades en este campo se han visto en suspensión, no obstante, Steckel indica en el comunicado oficial que se pretende retomar el proyecto en el caso de consigan futura financiación, véase [8].
En Abril del año 2019, según informó el Jet Propulsion Laboratory (JPL), la NASA desarrolló con éxito un experimento con el fin de demostrar la viabilidad de la transferencia de datos mediante comunicación óptica con tecnología laser entre dos cubesats [9]. En dicho experimento, la agen-

cia espacial estadounidense utilizó el Optical Communications and Sensor Demostration (OCDS) actuando como el emisor del haz de luz y como receptor el módulo CubeSat Multispectral Observation System(CUMULOS) a bordo de la nave Integrated Solar Array and Reflectarray Antenna (ISARA). En la noticia, se detalla, que en el momento de la transmisión los cubesats se encontraban alrededor de 451 km sobre la tierra (OBT) y alrededor de 2,414 km de distancia entre ellos. En las

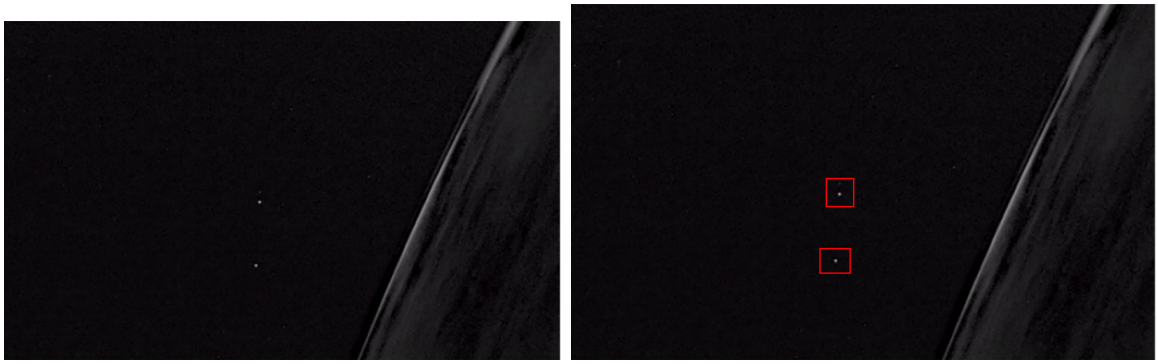

Figura 2: Enlace óptico con tecnología laser. Fotografía proporcionada por la NASA.

imagenes facilitadas por el JPL, se puede apreciar el momento de la transmisión como un destello de luz proviniente del láser del OCDS incidiendo en ISARA. Este hecho, demuestra que es posible desarrollar una tecnología optimizada, basada en este principio, para la transferencia de datos entre cubesats en órbita baja, en palabras de Rich Welle, uno de los investigadores principales de la misión OCDS de la nasa:

> “ Este éxito demuestra que es posible construir y operar enlaces de comunicaciones ópticas de satélite a satélite en sistemas que son sustancialmente más pequeños y más simples de lo que se consideró en el pasado. El futuro de las comunicaciones espaciales es óptico, y este resultado puede ser el primer paso en un camino para hacer que las comunicaciones ópticas sean ubicuas en la órbita de la Tierra, incluso en los satélites más pequeños.”
> — EuropaPress - Rich Welle [10]

La optimización de esta capacidad podría permitir a las constelaciones de pequeños satélites transferir datos de alto volumen entre sí en la órbita baja terrestre o incluso en órbita alrededor de la Luna.

Parte I
Diseño y arquitectura del sistema

Como se ha mencionado, este proyecto forma parte de un proyecto mayor en el que se esta diseñando un cubesat (nanosatelite) de $10cm \times 10cm \times 10cm$, se pretende, que este satelite esté constituido por módulos que encapsulen las funciones de cada uno de ellos independizandolos del sistema, de forma que puedan ser sustituidos o eliminados fácilmente con el fin de mejora/actualizar el sistema si ello se requiere.
Varios proyectistas estan llevando a cabo estudios del diseño de cada uno de los módulos para desarrollar la integración final en el cubesat, cumpliendo los requisitos de tamaño que se ha establecido (anteriormente mencionado), el diseño del nanosatelite se puede apreciar en la figura 2:

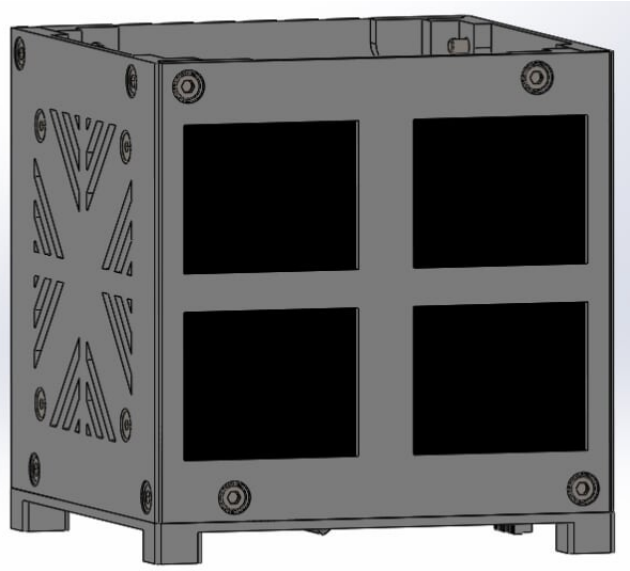

Figura 3: Diseño 3D actual del cubesat

2. Sistema modular del cubesat

El proyecto conjunto de todos los módulos esta en diseño constante actualmente y es posible que se realicen modificaciones en cuanto a los componentes se refiere, pero a fecha de redacción de este documento, el cubesat estaría formado por los componentes principales detallados a continuación:

- **Antena de radiofrecuencia - TI-CC1 101:** Dispositivo encargado de transferir datos de posicionamiento y actitud de un cubesat a otro a una frequencia determinada
- **Arduino Nano:** Microcontrolador encargado de realizar el control de sistema y el tratamiento de los datos de cada uno de los sensores.
- **GPS - GY-GPS6MV2:** Unidad encargada de calcular el posicionamiento del cubesat

- **IMU (Inertial Measurement Unit) - GY-521:** Sensor encargado de determinar los valores de actitud del cubesat.
- **Bluetooth - HC-06:** Transmisor necesario para establecer la conexion con el PC que va a realizar el control del cubesat (a nivel experimental, es posible que no se instalase a nivel de producción).
- **Diodo Láser - L850P010:** Diodo encargado de realizar la transmision óptica de datos mediante un haz de luz dirigido directamente al receptor óptico del cubesat con el que se establece la comunicación.
- **Receptor óptico - FDS1010:** Fotoreceptor encargado de recibir el haz de luz incidente por parte del diodo del cubesat emisor y de calcular el incremento de fotones recibido para llevar el dato a su posterior tratamiento en el microcontrolador.
- **Panel Solar - SLDM121H10L:** Panel encargado de generar una intensidad/voltaje a partir de los rayos solares que recibe. En este caso, tambien se esta llevando a cabo el estudio de la viabilidad de usar como fotoreceptor estos paneles solares.

Detallaré, en este proyecto, las unidades principales que hacen referencia al sistema de comunicación cuyo software se va a diseñar. Cabe indicar, que, además de estos componentes también se esta desarrollando todo un sistema de potencia para la alimentación y magnetorques para posicionar el cubesat, pero no vamos a detallar sus especificaciones y funcionamiento ya que carece de interés para este proyecto.
A continuación, se va a realizar una breve introducción conceptual a los módulos que proporcionan datos de interés para establecer un enlace de comunicación entre dos cubesats que se quieren comunicar.

2.1. Módulo de determinación de actitud

La actitud del satélite nos proporcionará la información imprescindible para realizar el alineamiento con otro cubesat para iniciar la transferencia de información, la actitud, nos proporciona datos sobre la orientación del cubesat con respecto a un determinado sistema de referencia.
Para ello, éste modulo cuenta con un circuito integrado llamado MPU-9250 que forma parte de la unidad GY-91 (figura 3), tambien llamada, IMU por sus siglas en inglés (Inertial Measurement Unit).

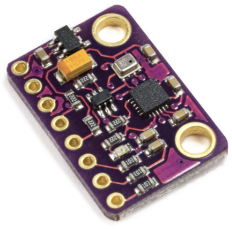

Figura 4: Módulo IMU GY-91 para Arduino

Este IC (MPU-9250) contiene un acelerómetro de 3 ejes y un giroscopio de 3 ejes con lo que dispone de 6 grados de libertad y nos permite, conocer toda la información necesaria para determinar la

orientación y aceleraciones a las que está sometido. Con lo cual, con los datos de orientación y aceleración recibidos del acelerómetro y giroscopio interno, el circuito, determina mediante el cálculo de cuaterniones (entre otros arreglos matriciales), la actitud actual del satelite. Dado que la IMU dispone de protocolo de comunicación I2C puede transmitir esta información al microcontrolador (Arduino) para su tratamiento posterior. Tambien, indicar que su voltaje de alimentación es de 3.3 - 5 VDC lo cual se adapta perfectamente a nuestras necesidades. Además de ello, el módulo MPU-9250 dispone también de magnetómetro para determinar el norte magnético y tener una referencia común en todos los cubesats.
Con este módulo, ya tenemos disponible la orientación exacta del satélite, veamos ahora, como obtenemos los datos de su posición.

2.2. Módulo de posicionamiento

En un entorno real, los satelites no disponen dispositivos de posicionamiento, su posición, se calcula mediante los denominados elementos orbitales Keplerianos [11]:

- Época
- Inclinación orbital
- Ascensión recta del nodo ascendente
- Argumento del perigeo
- Excentricidad
- Movimiento medio
- Anomalía media
- Arrastre (opcional)

Mediante estos parámetros (y otras determinaciones como las perturbaciones), se calcula la órbita exacta del satélite, y, en consecuencia, su posición en cada momento.
En nuestro caso, el prototipo no se halla en órbita, con lo cual, se utiliza un módulo GPS que proporcionará la posición, emulando la situación anterior y obviando los parámetros detallados ya que ni son necesarios ni se pueden determinar en tierra.
La unidad GPS utilizada para determinar la posición del satelite es la GY-GPS6MV2 del fabri-

Figura 5: Módulo GPS GY-GPS6MV2 para Arduino

cante Ublox. El motor de posicionamiento Ublox 6 de 50 canales cuenta con un Time-To-First-Fix (TTFF) de menos de 1 segundo, el TTFF es una medida de tiempo utilizada en los dispositivos de

navegación y nos indica la cantidad de tiempo necesaria para la adquisición de señales de satélite, datos de navegación y el cálculo de una solución para facilitar la posición. El motor de adquisición dedicado, con 2 millones de correlacionadores, es capaz de realizar búsquedas masivas de espacio en paralelo de tiempo/frecuencia, lo que le permite encontrar satélites al instante [12]. El diseño y la tecnología de que dispone, suprimen las fuentes de interferencia y mitigan los efectos de múltiples rutas, lo cual proporciona una mayor precisión.
El módulo, dispone de varios protocolos de comunicación (USB, UART, SPI y DDC), y envía los datos calculados en un formato de encapsulamiento determinado para que sea tratado por el microcontrolador. Para la transferencia de estos datos el último circuito del que se tiene conocimiento, es el reflejado en el proyecto de final de carrera de Juan Palomares (ver [13]), en el que se utiliza el protocolo UART, lo cual permite una trasferencia rápida de los datos.
De esta forma, ya disponemos también de los datos de posición en el microcontrolado. El sistema esta listo para enviar estos datos al otro cubesat mediante radiofrecuencia e iniciar el alineamiento y la comunicación óptica.

2.3. Módulo de comunicación

Una vez obtenidos los datos de posición y actitud del satélite, necesitamos enviar esta información al cubesat con el que deseamos establecer la comunicación, de ahora en adelante, el receptor. Para ello, se utiliza la antena de radiofrecuencia TI-CC1 101, que emitirá la señal con la información

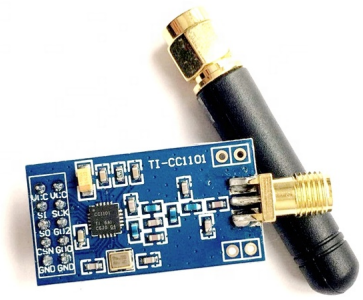

Figura 6: Módulo RF TI-CC1 101 para Arduino

que le proporcionará el microcontrolador(que se habrá encargado mediante otros procesos de descodificar/codificar y tratar los datos que le han sido proporcionados por la IMU y el GPS).
La antena TI-CC1 es un transceptor low-cost de menos de 1 GHz diseñado por Texas Instruments para aplicaciones inalámbricas de muy baja potencia. El circuito está destinado principalmente al sector industrial, científico y médico, actúa en bandas de frecuencia de 315, 433, 868 y 915 MHz, pero puede ser programado fácilmente para operar en otro rango de frecuencias (300-348 MHz, 387-464 MHz y bandas de 779-928 MHz)[14]. En nuestro caso, este dispositivo facilitará la información, crucial, para realizar el primer paso en el proceso de comunicación (el alineamiento y enlace).

Este proceso, aun por detallar con exactitud, se inicia con el flujo de información tanto del **receptor** como del satélite que desea iniciar la comunicación, en adelante, el **emisor**. El objetivo es que tanto emisor como receptor dispongan de los datos de actitud y posición el uno del otro, una vez dispongan de estos datos, otros subsistemas(sistema de propulsión magnética) se encargarán de corregir la orientación del satélite para converger en un mismo punto, en este momento, se inicia el enlace óptico.
Cuando se ha producido el alineamiento, el microcontrolador tiene que dar la orden al fotodiodo para que inicie la secuencia de sincronismo con el emisor. Para ello, el diodo láser que se va a utilizar es el L850P010 de capaz de trabajar a una longitud de onda de 835-865 λ_p [15]. Para que

Figura 7: Diodo laser L850P010

el receptor pueda procesar este haz de luz recibido, se va a utilizar el fotoreceptor FDS1010 que se encargará de recibir los incrementos de luz recibidos (datos) y transferirlos al circuito hardware que realizará el tratamiento de la señal y la facilitará al microcontrolador para poder convertirla en el mensaje final recibido. El diseño y simulaciones del hardware adicional en este punto para el

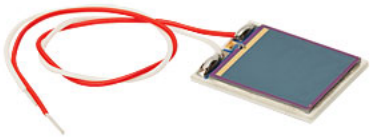

Figura 8: Receptor óptico FDS1010

tratamiento de estas señales electricas, su acondicionamiento, filtrado y amplificación se detalla en el proyecto de final de grado de Pol, titulado , que se ha confeccionado en paralelo a este mismo proyecto.
El tema principal que se desarrolla en el presente proyecto es el diseño del software completo para la comunicación óptica (asumiendo que ambos satélites, emisor y receptor, se encuentran ya alineados). Se van a desarrollar tres alternativas de protocolo para la comunicación de ambos satélites utilizando, para el emisor , un microcontrolador STM32 y, para el receptor, una placa Arduino UNO que emularan el comportamiento de dos prototipos de satelite.
Pero antes de entrar en detalle al desarrollo de código, se detallan, a continuación algunos conceptos claves para contextualizar el enfoque y objetivos de cada uno de los programas que se han diseñado.

3. Conceptos previos

Con el fin de introducir el entorno en el que se va a desarrollar el software, se van a detallar una serie de conceptos para plasmar el enfoque sobre el que se esta trabajando.
Para realizar cualquier proceso de transferencia de datos como puede ser internet, pero también, cualquier proceso en el que se incluya un emisor/receptor, un mensaje (datos), y una manera de transferirlo (protocolo), se debe describir todo un sistema que describa las especificaciones en cada punto del proceso. En nuestro caso, disponemos, efectivamente, de un emisor, un receptor, un mensaje y un medio para transferirlo, lo cual nos obliga a describir un protocolo de comunicación para que el mensaje pueda ser enviado/recibido y "entendido" por ambos cubesats.
Se define protocolo como un conjunto de normas y reglas para la transferencia de datos en la misma capa (lo veremos ahora). Pero, ¿porque debemos realizar este proceso, y no podemos enviar datos directamente?, el motivo, es que ambos cubesats deben estar preparados para abstraerse de lo que sucede por " debajo" cuando reciben un mensaje, simplemente tienen que estar programados para procesar ese mensaje y entenderlo de forma nativa.
Pongámos un ejemplo para ilustrarlo:
Imaginemos que Alice y Bob, dos filósofos, estadounidense e inglés, respectivamente, se desean comunicar y enviar un telegrama con un artículo sobre filosofía moderna. Pues bien, si se implementase un diseño completamente monolítico, Alice y Bob, , a parte de filósofos, deberían ser telegrafistas, para descodificar el telegrama e ingenieros para construir y conectar los telégrafos, Se hace evidente, que no es una forma eficiente de proceder. En este ejemplo, se simplifica para

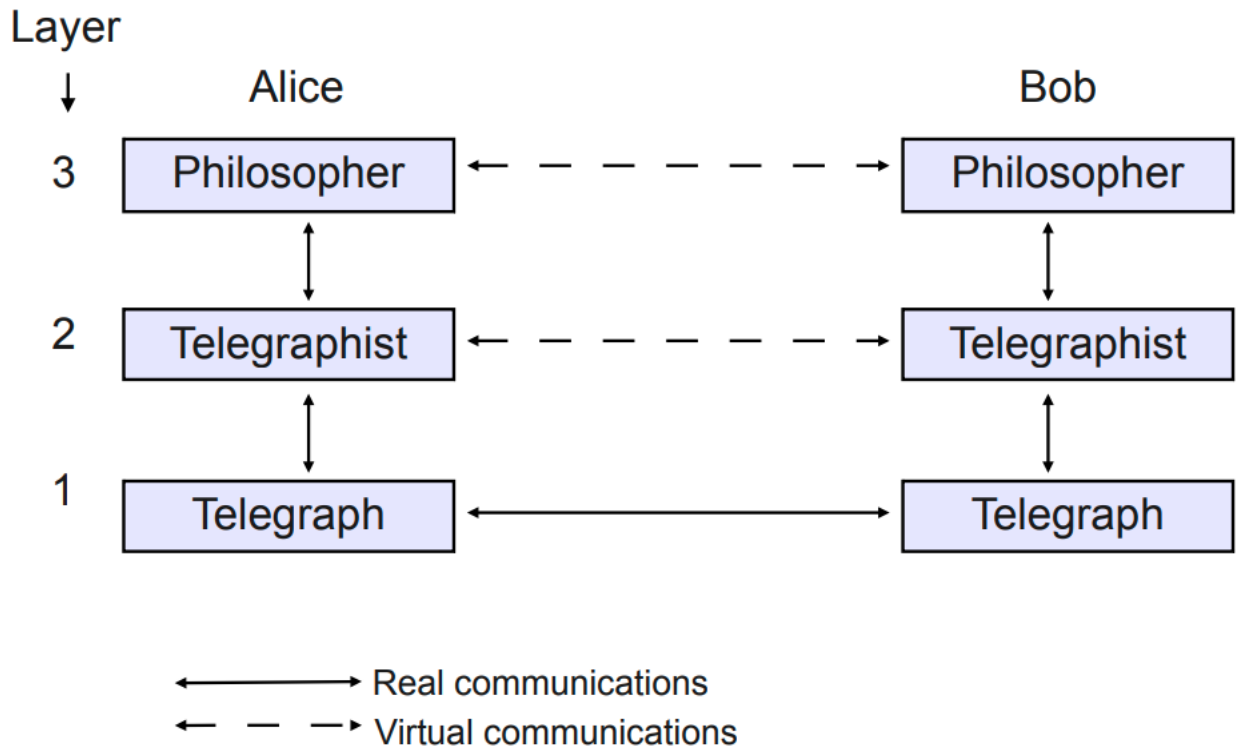

Figura 9: Representación gráfica de modelo de juguete (Toy Model). UPC.[1]

poder entender la necesidad real de crear capas de abstracción para la comunicación y cada capa,

con sus normas y reglas para enviar el mensaje entre ellas, los protocolos. Veremos a continuación, como se implementa esto en el mundo real.

3.1. Modelo OSI

Sobre este paradigma, la Organizacion Internacional para la Normalización (ISO), propuso e implementó un modelo estándar para implementar la abstracción en concepto de capas, interfaces y protocolos. Se elaboró el denominado, modelo OSI (Open System Interconnection) para solucionar el problema. Este modelo, detalla las diferentes capas de abstracción y qué protocolos actúan en cada una de ellas, es un modelo estandarizado para los esquemas de redes pero aplicable a otros tipos de comunicación, podemos ver gráficamente en la siguiente figura un esquema simplificado del modelo OSI y como esta implementado actualmente en el campo de los sistemas:

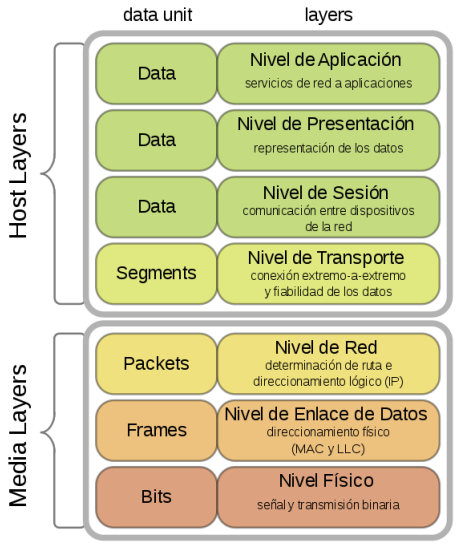

Figura 10: Modelo OSI [2]

Este modelo es aplicado a las comunicaciones de redes (principalmente internet), pero también lo podemos aplicar a la comunicación entre dos nodos separados por un medio,

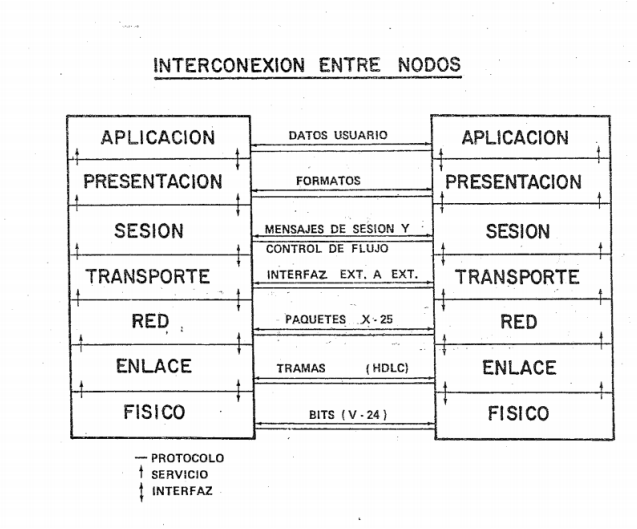

Figura 11: Arquitectura de nodos OSI [3]

En este proyecto se tratarán las capas 1 y 2 (Capa física y capa de enlace), ya que las siguientes aplicaciones no intervienen en el objetivo de este proyecto. Veamos un poco más en detalle de que tratan estas capas.

3.2. Capa física y capa de enlace

La capa física

En esta capa, se llevan a cabo todo el tratamiento de las señales eléctricas para transmitir bits de un nodo a otro por un medio. En nuestro caso, emisor/receptor son cubesats separados en el espacio, lo cual, delimita en extremo el medio de transmisión de los bits, pero, es posible esta transmisión en el campo de la óptica ya que la comunicación no se realiza por cables que contienen un flujo de electrones si no por óptica que utiliza un flujo de fotones, lo cual, no necesita un medio físico para moverse.

En el presente proyecto, se ha decidido realizar la comparativa en cuanto a viabilidad de transmisión de información mediante 3 tipos de protocolos (los veremos a continuación) a nivel físico, sin embargo en una aplicación real, la transmisión mediante la óptica debería ser una onda modulada, pero veremos que en esta fase experimental la comunicación se simula mediante cableado y por lo tanto, podemos tratar que ocurre con un protocolo de onda no modulada (UART). LA capa física se encarga de otras funciones y servicios como pueden ser, especificaciones eléctricas, codificación de línea, conmutación de circuitos, topología de redes...etc, en este proyecto, me centro en la modulación, dejando la parte del desarrollo del hardware para el filtrado y procesado de la señal a otros proyectos.

La capa de enlace

Esta capa, se encarga de recibir los datos recibidos en su capa inferior (la física) y desarrollar un encapsulamiento determinado de estos datos con el fin de garantizar un envío de información fiable y libre de errores. Este encapsulamiento, se denomina **trama** y es un conjunto de bloques de información que confinan el mensaje y que aportan seguridad en la transmisión. El Institute of Electrical and Electronics Engineers (IEEE o IE^3) se encara de desarrollar estándares y normativas en el campo de la electrónica, y en este caso para el modelo OSI se recoge la norma IEEE 802.2 [16]

que indica como debe ser la trama para todos los tipos más habituales de redes (Ethernet, Wi-fi, WiMAX...), sin embargo, para el prototipo de cubesat, crearemos una trama propia adecuándola a nuestras necesidades ya que para comunicaciones espaciales existen una serie de protocolos de comunicación de mucha complejidad que nos aportarían demasiada funcionalidad para nuestro objetivo. Veremos más adelante, en detalle cómo estará formada nuestra trama.

3.3. Transmisión de la señal

En esta sección, se va a detallar el tipo de señal que vamos a enviar y cómo se va a modular. Mediante dos microcontroladores conectados por cable se emulará el enlace óptico que se podruciría en el laboratorio o directamente en el espacio, a nivel de software. Con el código que se va a escribir se prepararán los pines GPIO necesarios del microcontrolador, para poder hacer el cambio de cable a óptica sin necesidad de modificar el software.
Pero antes de ello, explicaré, a nivel teórico que señal vamos a enviar.
Como se ha explicado anteriormente, el dispositivo que se utilizará para la comunicación óptica, es un diodo láser para el caso del emisor, y un fotoreceptor para el caso del receptor.
El diodo láser, como un led convencional, esta formado por una unión p-n que perminte el flujo de electrones cuando se polariza en directa liberando una parte de la energía en forma de radiación térmica y otra parte en forma de fotones o luz. La diferencia entre el led convencional y un diodo láser, es que el diodo láser es capaz de emitir estos fotones con coherencia espacial dando lugar a un haz de luz concentrado mediante una cavidad óptica resonante en la que intervienen espejos

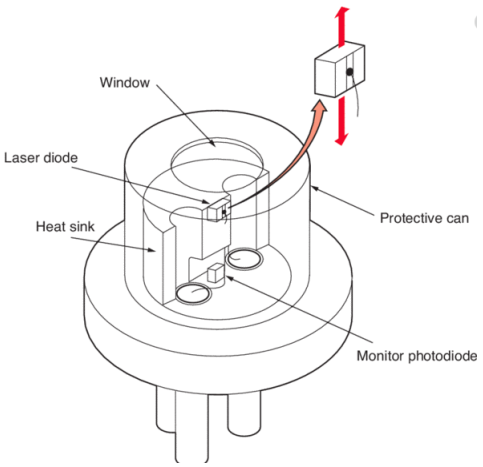

Figura 12: Diodo láser

que amplifican la reflectividad de la luz y la concentran en un único punto de salida.
El fotorreceptor, por su parte se encarga de convertir la luz incidente en corriente eléctrica, esta corriente puede ser convertida a voltaje incluyendo una resistencia de carga R_L al circuito, dando lugar a la siguiente expresión,

$$V_o = P \times \Re \times R_L$$

donde, P es la potencia y $\Re$ es la responsividad, parámetro que nos indica cuan sensible es el fotorreceptor a la potencia de luz incidente, que en nuestro modelo (el FDS1010) tiene un valor de $\Re = 0{,}725 A/W$, para longitudes de onda de entre 350 - 1100 nm. La resposividad, es la relación

entre la corriente generada y la potencia de luz incidente

$$\Re_\lambda = I_p/P$$

y varía fuertemente en función de la longitud de onda, en el modelo FDS1010 sigue la siguiente curva: ya que nuestro diodo láser puede trabajar entre 835 - 865 nm dispondríamos, aproximada-

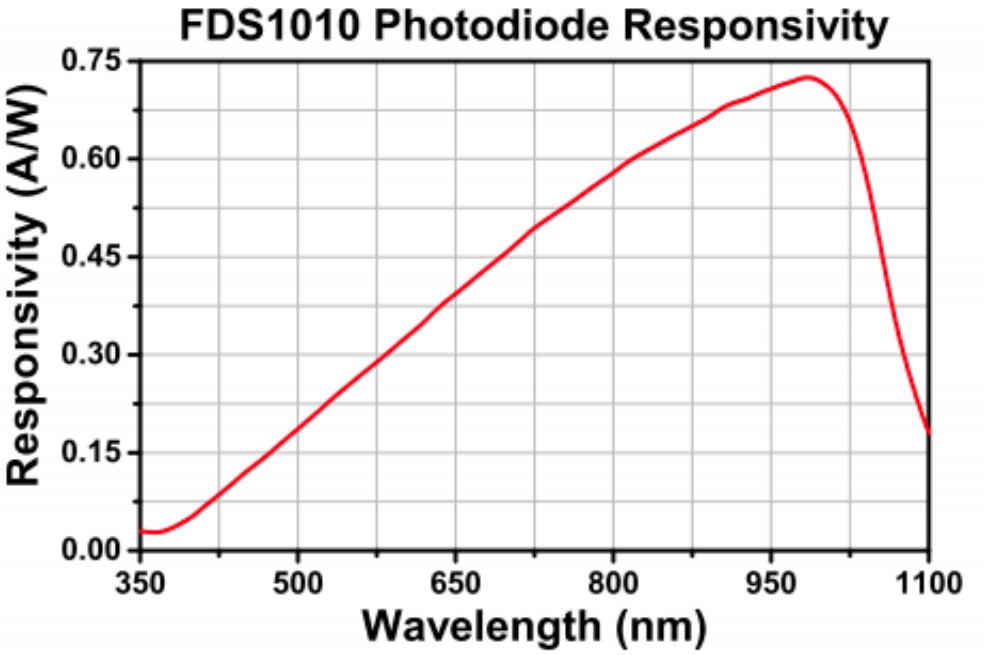

Figura 13: Responsividad frente a longitud de onda
[17]

mente, de unos $0{,}65 A/W$. Dado que el cubesat se encontraría en el espacio, a nivel de hardware, lo que se detectarán, son incrementos de luz (de voltaje) para poder diferenciar el haz del láser de la luz solar, por eso, a nivel de software, es importante desarrollar una etapa de sincronismo que se incluirá dentro de la trama a enviar.

3.3.1. Modulación PPM

La modulación, es el conjunto de técnicas que se utilizan para transmitir información mediante una onda portadora y existen de varios tipos, en este caso, la modulación PPM (Pulse Position Modulation), o modulación por posición de pulsos, codifica la señal en función la posición del pulso transmitido dado un período determinado. En la modulación por posición de pulso la información está contenida en la posición en la que ocurre el pulso.El pulso está desplazado respecto a un tiempo dereferencia de acuerdo al valor de la señal de entrada.Cada periodo de muestro marca un tiempo de referencia en la modulación.Podemos verlo gráficamente en la siguiente representación:

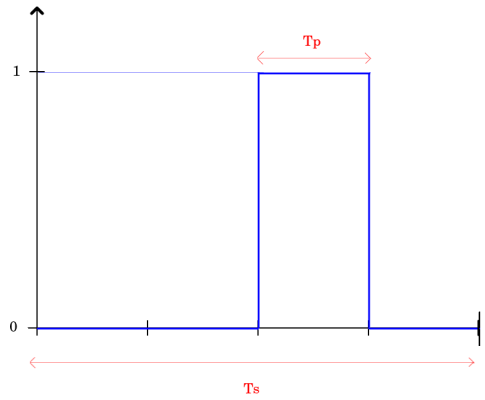

Figura 14: Representación PPM

El PPM pretende transmitir una palabra de M bits dado un tiempo o período enviando un número de bits n_b por palabra, además debemos tener en cuenta los siguientes parámetros que facilitan información muy útil para el cálculo de la eficiencia de este sistema de modulación:

- M = Palabra o símbolo que se desea transmitir. También, indica el orden de la modulación.
- n_b = Número de bits por palabra
- r_b = Velocidad de transmisión
- T_p = Duración del pulso
- T_{sym} = Duración del símbolo o palabra

estos parámetros, guardan la siguiente relación entre ellos:

$$n_b = log_2 M$$

$$T_{sym} = MT_p$$

$$r_b = \frac{n_b}{T_{sym}}$$

En la figura 14, vemos, de forma muy esquemática cómo se codificaría una palabra de 4 bits en la que el pulso se posiciona en el tercer bit con una duración de palabra de T_s y una duración de pulso de T_p. Técnicamente, el PPM se modula de forma secuencial según la onda que se desea transmitir, pero veremos, que a nivel digital, y con la posibilidad de programar el microcontrolador que vamos a utilizar, redefiniremos el PPM para que la duración de la palabra no sea secuencial, y por lo tanto se optimizará la velocidad de transmisión. Más adelante, se detallará qué palabra o símbolo se va a transmitir y cómo funcionará, concretamente el PPM que se va a implementar para este proyecto en concreto.

3.3.2. Modulación PWM

Si bien el PPM, modula la onda en función de la posición de un pulso determinado dado un período de tiempo, el PWM (Pulse Width Modulation o Modulación por ancho de pulso) utiliza la duración de dicho pulso para transmitir el dato. Dado un período T de tiempo, y asumiendo, que, como ocurría con el PPM los estados altos y bajos del pulso serán tensiones fijas que nos indican o 1 o 0 para la transmisión del bit digital, con el PWM, debemos establecer lo que se conoce como un *duty cicle* que no es más que el parámetro que indica el tiempo de vida del pulso respecto la duración total del período, un *duty cicle* del 100 % denota que el pulso ha durado todo el período, 50 % la mitad del período... etc.
Se pude apreciar en la siguiente figura, precisamente, aplicando el PWM que proporciona arduino por defecto:

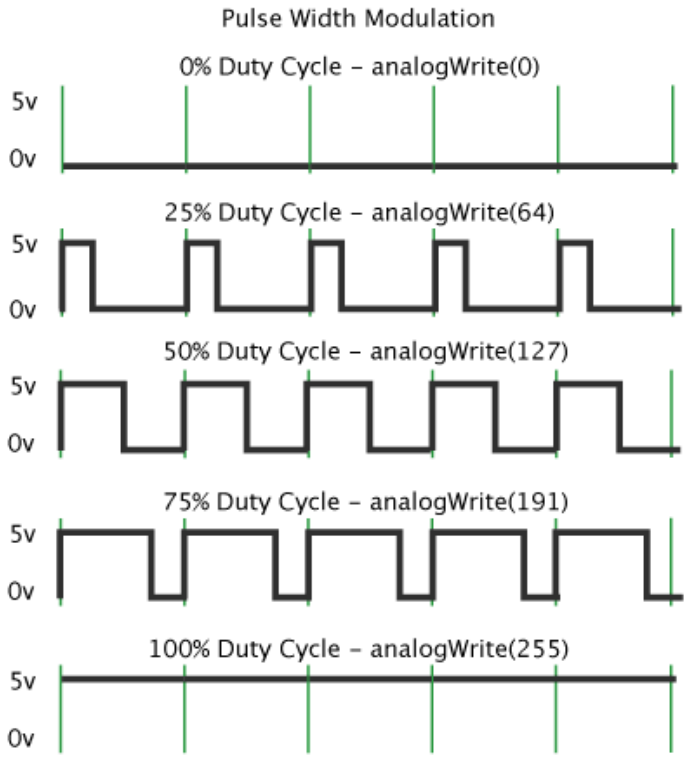

Figura 15: Representación PWM

en este caso, nos interesan los parámetros:

- D es el ciclo de trabajo (duty cicle)
- τ es el tiempo en que la función es positiva (ancho del pulso)
- T es el período de la función

y cuya relación es, simplemente:

$$D = \frac{\tau}{T}$$

De nuevo, igual que en el PPM, readaptaremos este sistema de forma que se ajuste a nuestras necesidades mediante la programación de los timers internos de cada microcontrolador, lo veremos en la implementación del software.

3.3.3. UART

Por último, se utilizará el protocolo UART, simplemente para la implementación por cable dado que no es recomendable usarlo en transmisiones ópticas, pero nos servirá para realizar la comparativa de la transmisión bit a bit con respecto a la modulación anteriormente explicada.
El protocolo UART (Universal Asynchronous Receiver-Transmitter o Transmisor-Receptor asíncrono universal) es un protocolo ampliamente utilizado en el campo de la electrónica y, realmente, es un dispositivo de hardware usado habitualmente para la transmision rápida y segura de los datos, tambien, se suele usar para *flashear* o reprogramar memorias o microcontroladores. La UART, toma bytes de datos y transmite los bits individuales de forma secuencial. En el destino, un segundo UART vuelve a ensamblar los bits en bytes completos.
Cada UART contiene un registro de desplazamiento, que es el método fundamental de conversión entre formas series y paralelas. La transmisión en serie de información digital (bits) a través de un solo cable u otro medio es menos costosa que la transmisión en paralelo a través de múltiples cables. El UART generalmente no genera ni recibe directamente las señales externas utilizadas entre diferentes elementos del equipo. Se utilizan dispositivos de interfaz separados para convertir las señales de nivel lógico del UART hacia y desde los niveles de señalización externos, que pueden ser niveles de voltaje estandarizados, niveles de corriente u otras señales. La comunicación puede ser:

- **simple** Solo en una dirección, sin que el dispositivo receptor pueda enviar información al dispositivo transmisor.
- **full duplex** Ambos dispositivos envían y reciben al mismo tiempo.
- **half duplex** Los dispositivos se turnan para transmitir y recibir.

Figura 16: Transmisión bit a bit UART

Este dispositivo, es realmente útil ya que podemos crear una trama de sincronismo entre los transmisores/receptores que se van a comunicar y se puede realizar una transmisión segura de los

datos ya que también podemos generar bits de paridad o CRC como veremos más adelante de forma que se minimicen considerablemente los errores que puedan surgir. A priori, en este proyecto se va a realizar una comunicación simple, pero, desarrollar una comunicación halfduplex o incluso full duplex, no sería complejo de implementar. En la siguiente imagen podemos apreciar una aplicación real de una UART mediante el estándar RS-232:

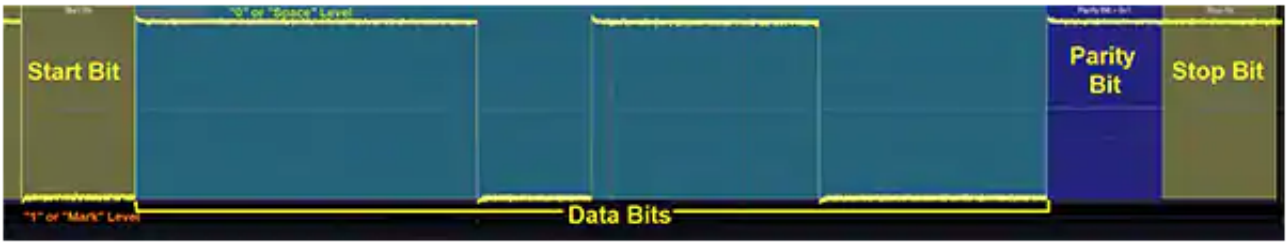

Figura 17: Un paquete de datos RS-232 consta de un bit de inicio, 5 a 8 bits de datos (se muestran 8), un bit de paridad (opcional) y 1, 1.5 o 2 bits de parada. (Fuente de la imagen: Digi-Key Electronics).

4. Componentes principales del sistema de comunicación

Para el desarrollo del presente proyecto, se ha decidido utilizar materiales low cost que proporcionen la funcionalidad deseada con unas especificaciones mínimas. En este caso, se han utilizado dos microcontroladores de bajo coste, open source y con amplio abanico de posibilidades en cuanto a programación se refiere, dichos controladores son el STM32F103C8T6 y el ATMEGA328P-PU, pertenecientes, respectivamente a las placas de desarrollo 'Blue Pill' y Arduino UNO.
Como se ha especificado anteriormente, el sistema se ha diseñado de forma que la Blue Pill actúe como emisor y Arduino UNO actúe como receptor, pese a que, como veremos más adelante, el código se ha diseñado pensando en la escalabilidad de proyecto de forma que se ha preparado una capa de abstracción (una API) para, si fuese necesario, aplicar un protocolo half-duplex, o full-duplex, utilizando las mismas funciones que actualmente están diseñadas.
El cubesat, debe cumplir unas dimensiones máximas (10cm x 10cm x 10cm) y con esa especificación, este tipo de placas de desarrollo se acoplarían perfectamente en el diseño debido a su reducido tamaño y a que pueden realizar casi cualquier tipo de funcionalidad (lo cual, permite que ejecute tareas provinientes de los distintos módulos del cubesat, no solo de comunicación).
A parte de las placas de desarrollo, en esta sección detallaré, también, el entorno de desarrollo, en cuanto a la programación del código se refiere, que se ha utilizado, tanto IDE's como compiladores y flasheadores.Con el fin de facilitar el entendimiento del proyecto, se va a dar una intruducción de como se ha diseñado el código en función del flujo de información que se va a transmitir:

1. En primer lugar, se ha configurado, pines GPIO, pines de RCC y de debug en el ARM(STM32) (veremos más adelante cómo)

2. Con el micro preparado, se configuran las interrupciones del procesador.

3. Dentro de cada interrupción, se programa las funciones a las que se llamarán, cuando una IRQ se produzca.

4. Se programa el *main* con la rutina de programa que se quiera implementar, en nuestro caso, el envío de información.

5. Con esto, el emisor estará preparado para la transmisión de bits. En el receptor, dependiendo del protocolo del emisor, se deberá realizar también distintas funciones en las IRQ para que pueda llevar a cabo la parte más importante de la comunicación, el sincronismo.

6. Cuando el emisor y receptor estén ya programado para llamar a las funciones necesarias para transmitir según el protocolo que se desee, se añade una capa de abstracción al sistema. Esta capa, nos permite olvidarnos de que hacen internamente los micros para transmitir la información, y nos permitirá trabajar en lo que se va a transmitir, la trama.

7. La trama de datos es el elemento mediante el cual pasamos de la capa física del sistema a la capa de enlace. Como las funciones internas en cada microprocesador ya están programadas, podemos desarrollar una trama a más alto nivel en la que incorporaremos una etapa de control de errores y seguridad.

8. A partir de aquí, se podrían añadir capas a más alto nivel, añadiendo nueva funcionalidad, como encriptación de los datos, headers con información del origen y destino, identificadores únicos... etc, pero esto queda fuera del alcance de este proyecto.

4.1. Microcontrolador STM32

El STM32f103C8 (en adelante, el ARM) es un microcontrolador de 32bits basador en arquitectura ARM de Cortex con un clock rate de CPU de 72MHz, distribuido y firmado por STMicroelectronics, empresa líder en el diseño de microcontroladores de distintas características.

Figura 18: Imagen STM32 de STMicroelectronics

El STM32 , dispone de 12 ADC, 3 timers de propósito general (los que usaremos) de 16bits y uno de control avanzado que usaremos para el PWM, así como interfaces de I2C y SPI, USART, USB o CAN. Este dispositivo, funciona con una fuente de alimentación de 2 a 3.6V y tiene una temperatura de trabajo de entre -40 ºC a 85 ºC. Es un microcontrolador ampliamente utilizado debido a su versatilidad y funcionalidad en relación a su precio, se podría utilizar en aplicaciones como: accionamientos de motor, control de aplicaciones, equipos médicos, periféricos para PC y videojuegos, plataformas GPs, industria, control de PLC, inversores...etc. En la siguiente tabla, se especifica un resumen de la funcionalidad que nos proporciona esta familia de STM32:

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	General-purpose	3		3		3		3	
	Advanced-control	1		1		1		1	
Communication	SPI	1		2		2		2	
	I^2C	1		2		2		2	
	USART	2		3		3		3	
	USB	1		1		1		1	
	CAN	1		1		1		1	
GPIOs		26		37		51		80	
12-bit synchronized ADC Number of channels		2 10 channels		2 10 channels		2 16 channels(1)		2 16 channels	
CPU frequency		72 MHz							
Operating voltage		2.0 to 3.6 V							
Operating temperatures		Ambient temperatures: -40 to +85 °C / -40 to +105 °C (see *Table 9*) Junction temperature: -40 to + 125 °C (see *Table 9*)							
Packages		VFQFPN36		LQFP48, UFQFPN48		LQFP64, TFBGA64		LQFP100, LFBGA100, UFBGA100	

Figura 19: Tabla de características STM32 - Datasheet

En nuestro caso, el microcontrolador estará integrado en la placa de desarrollo Blue Pill, que nos proporcionará, esta funcionalidad, a un precio ridículamente bajo (2.00 €) y que dispone del siguiente pinout para aplicado la funcionalidad anteriormente detallada:

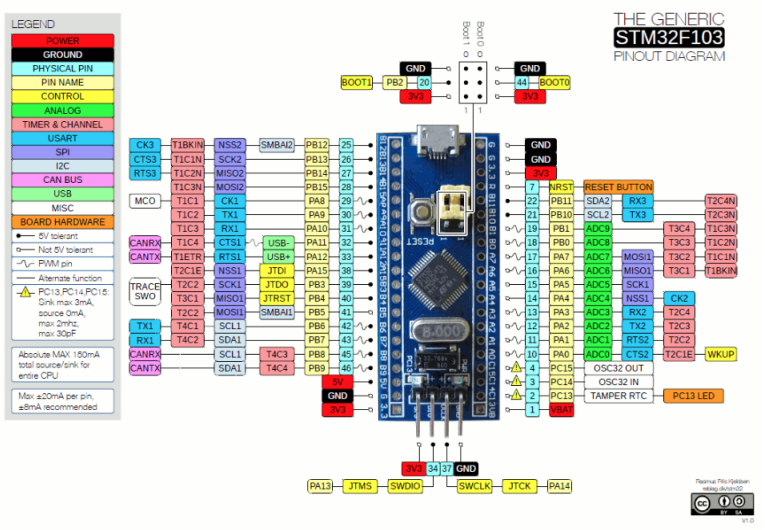

Figura 20: Pinout STM32

4.2. Enlace ST-Link

Para poder flashear, es decir grabar el programa diseñado en la memoria del STM , se podría hacer mediante los pines de programación simplemente cambiando de posición el jumper de que dispone la Blue Pill, pero en este caso, y ya que se disponía en el laboratorio de ello, se va a utilizar el enlace, ST-Link.
ST-Link es un dispositivo que se conecta por vía USB al PC y sirve para programar el microcontrolador de forma rápida y segura sin peligro a que se puedan reescribir puntos en la memoria delicados (como el bootloader).
El ST-Link es sencillo de instalar y de utilizar, en el caso de Windows requiere de la instalación de ciertos drivers para estar operativo, en este proyecto, se ha utilizado Linux, por lo que la único requerimiento ha sido conceder permisos de escritura en el punto de montaje del ST-Link y ya está preparado para programar. Cabe mencionar, que ST-Link, es en si mismo un microcontrolador STM32, que dispone de los siguientes pines de conexión:

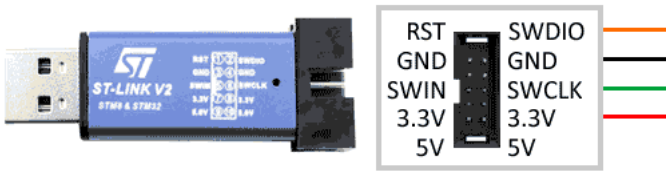

Figura 21: STLINK v2

Para nuestro propósito, utilizaré los pines:

- 2. SWDIO : Pines de transmisión de datos
- 4. GND : Tierra
- 6. SWCLK : Reloj

- 8. 3.3 V : Alimentación

De esta forma, y sin modificar ningun jumper, se ha podido ir programando y además, flashear de forma casi instantánea para debuggear(*Seguir la ejecución del programa línea a línea) mediante ciertos comandos y compiladores que se detallarán más adelante.

4.3. Arduino UNO

Arduino es una compañía de desarrollo de software y hardware libres, así como una comunidad internacional que diseña y manufactura placas de desarrollo de hardware para construir dispositivos digitales y dispositivos interactivos que puedan detectar y controlar objetos del mundo real. Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Los productos que vende la compañía son distribuidos como Hardware y Software Libre, bajo la Licencia Pública General de GNU (GPL) y la Licencia Pública General Reducida de GNU (LGPL),permitiendo la manufactura de las placas Arduino y distribución del software por cualquier individuo. Las placas Arduino están disponibles comercialmente en forma de placas ensambladas o también en forma de kits. [18]
Para este proyecto, se ha decidido utilizar para el receptor, una placa de Arduino UNO debido a su bajo coste y a su facilidad de programación. La placa Arduino UNO es el modelo R3 y dispone del microcontrolador ATMEGA328P-PU, un microcontrolador que dispone de:

- 32Kb de memoria flash
- 14 pines de entradas y salidas digitales, 6 de los cuales proporcionan salida PWM.
- Voltaje de funcionamiento a 5V
- 6 pines de entrada analógica.
- SRAM de 2Kb y EEEPROM de 1Kb
- 16 MHz

El dato más importante para este proyecto son esos 16MHz de velocidad, dado que el ARM funciona a 72MHz, debido a esto, deberemos tener en cuenta el sincronismo a la hora de hacer saltar las interrupciones. Para ello, se ha ideado un sistema por el cual las distintas velocidades de reloj no se se ven afectadas y se puede realizar una conversión de los datos de una forma "semiasíncrona"que detallaré más adelante, pero que permitirá trabajar con la máxima velocidad en ambos microcontroladores.

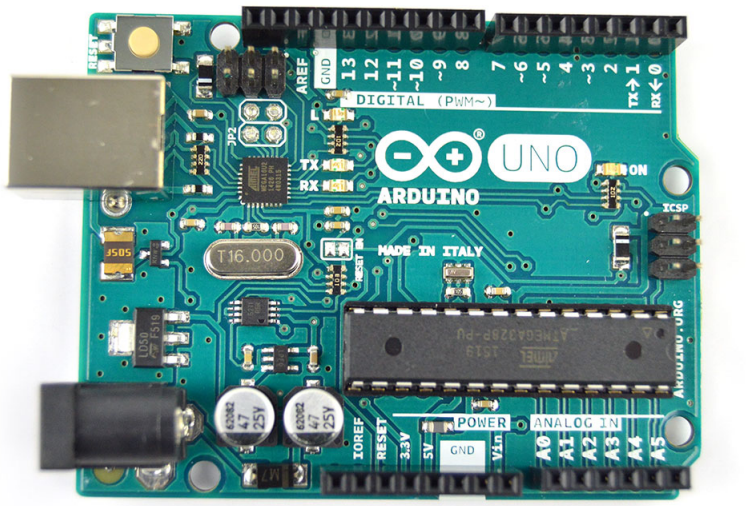

Figura 22: Arduino UNO

4.4. Entorno de desarrollo

Para el entorno de desarrollo y poder poner en funcionamiento el código generado se requieren de diversas herramientas:

- El IDE (Integrated Development Enviroment), es la aplicación que permite el desarrollo de código y ejecución del mismo en el caso de los lenguajes compilados.
- Herramienta de configuración del micro. En este caso, para el ARM se ha utilizado CubeMX, una herramienta gráfica que permite configurar los pines del microcontrolador y genera automáticamente el código de configuración para su posterior modificación e implementación en el código del usuario. Si bien es cierto que se pueden configurar los pines y , en general, cualquier aspecto del microcontrolador a nivel de código, es más eficiente hacerlo mediante esta herramienta que, además la proporciona el fabricante, y permite realizar configuraciones seguras y sin tener que escribir código.
- El compilador. Tanto Arduino, como ARM se desarrollan en lenguaje C, por lo que es necesario un compilador de C para poder debuggear y flashear ambos microcontroladores.
- Para el desarrollo de código del ARM, el fabricante, proporciona una HAL (Hardware Abstraction Layer). La HAL es un conjunto de API (Aplication Programming Interface) que facilitará en extremo, el manejo del hardware desde software, dado que proporciona un conjunto muy extenso de funciones que permiten trabajar sobre todas las areas del microprocesador (Pines GPIO, Interrupciones, Timers...).

En esta sección, es de interés detallar cual ha sido el entorno y la Toolchain utilizada para el desarrollo del proyecto, dado que una de las partes mas importantes para poder realizar cualquier proyecto de software es configurar un entorno idóneo para lo que se va a desarrollar, por lo que se va a detallar que se ha utilizado en cada caso de los elementos anteriormente mencionados.

1. **IDE:** El fabricante de STM32 , proporciona en su página web diversas herramientas para el desarrollo de los microcontroladores, en este caso, la herramienta que proporciona para el desarrollo de software en microcontroladores STM32F103 entre muchos otros, es STM32CubeIDE. STM32CubeIDE , es una herramienta de desarrollo *all-in-one* que reúne TreSTUDIOforSTM32 + STM32 CubeMX, basado en Eclipse, es un IDE para C/C++ pensado para trabajar con estos micrcontroladores y con su código de forma unificada en un solo software, proporcionando tanto CubeMX como el compiladore y el debugger se puede configurar, desarrollar, flashear y debuggear el micro con una sola herrmaienta.

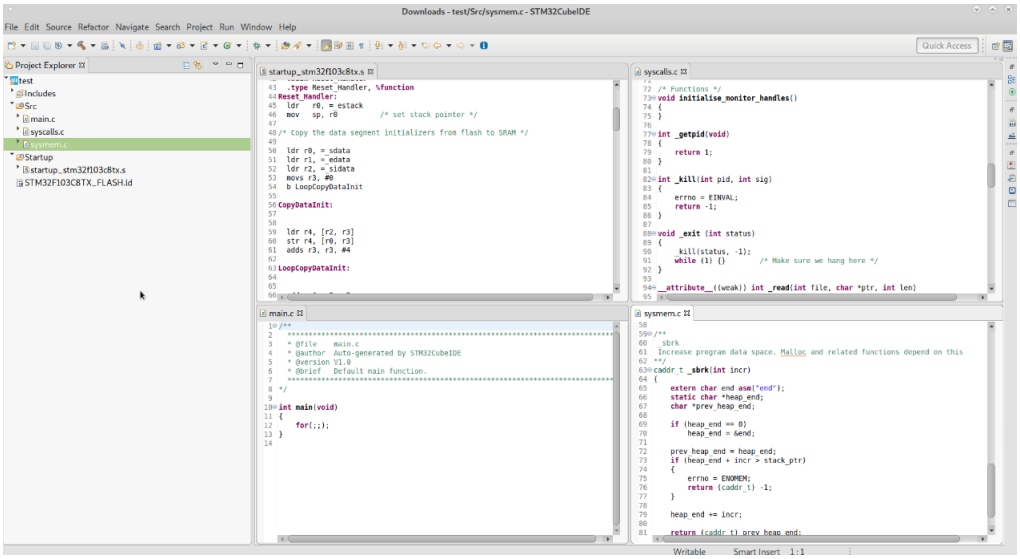

Figura 23: STM32CubeIDE

Para el caso de Arduino, se utilizó el conocido IDE proporcionado por Arduino, el cual permite desarrollar y compilar ademas de construir el código en el micro, también incluye código de ejemplo y algunas librerías de utilidad.

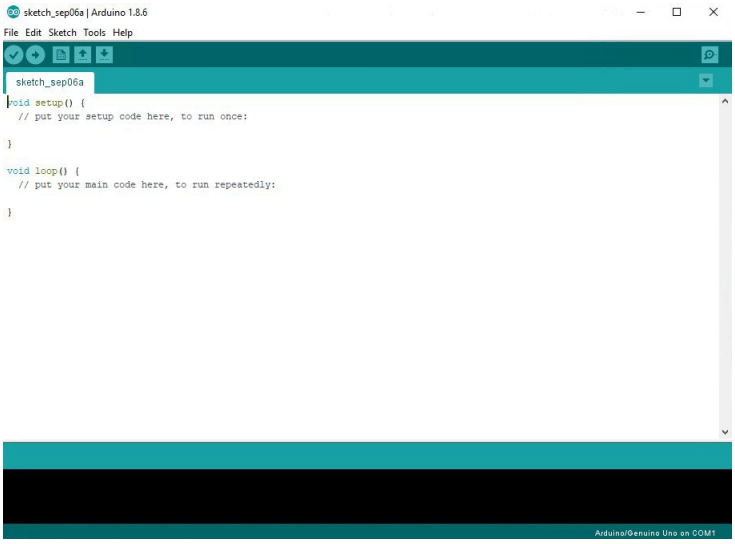

Figura 24: Arduino IDE

Sin embargo, y aunque parte del código se ha desarrollado con estas herramientas, finalmente decidí configurar un sistema completo de desarrollo con un único IDE preparado tanto para Arduino como para STM32 de forma que se ha utilizado Visual Studio Code con alguna extensiones para que permita flashear tanto ARM como Arduino a la vez y permita compilar ambos lenguajes C/C++. Se consiguió implementar este sistema, instalando a mano el compilador y descargando ciertas extensiones, además, se añadió el monitor serie de Arduino en el propio IDE para poder visualizar todas los datos de entrada en la placa. Este es el resultado del entorno personalizado para este proyecto:

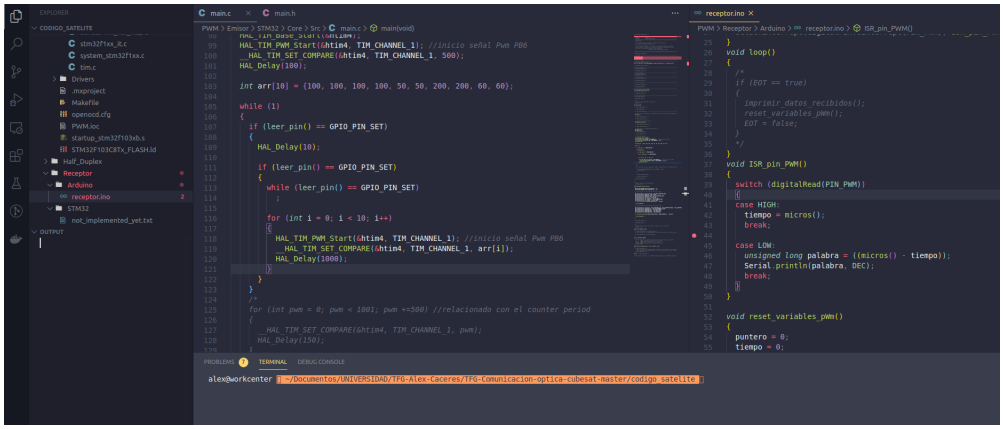

Figura 25: Visual Studio Code modificado para Arduino + STM

Se puede apreciar, a la izquierda, el árbol de rutas de ficheros, justo abajo, la salida OUTPUT del monitor serie, a la derecha el entorno de desarrollo con la terminal para flashear STM32.

2. **CubeMX** Esta herramienta, como se ha especificado antes, permite la configuración del microcontrolador ARM de forma que no es necesario hacerlo por código. Es un entorno gráfico, que proporciona el conjunto de pines del micro y sus distintas opciones de configuración.

De esta forma, cuando ya tenemos establecido como van a trabajar cada pin (incluso podemos nombrarlos o asignarles etiquetas), el programa genera el código de dicha configuración automáticamente, de forma que ya podemos trabajar en este código desde VSCode, simplemente tenemos que trabajar dentro de las etiquetas /*USER CODE BEGIN */ /*USER CODE END*/

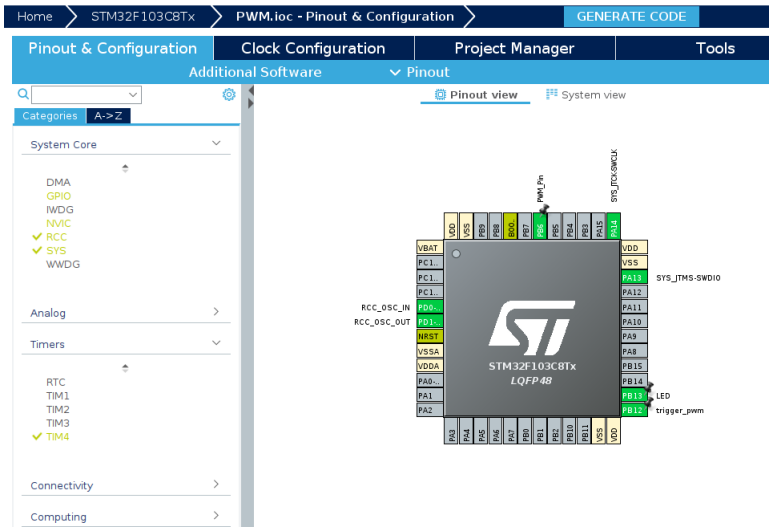

Figura 26: CubeMX

3. **El compilador:** Para compilar el código se ha diseñado un sistema de forma que el compilador pueda pasar el binario a otro software capaz de comunicarse con STLink y este, pueda *quemar* el código generado al microcontrolador.Para ello, como compilador , se ha utilizado ARM GCC no EABI(interfaz binaria de aplicaciones) además de OpenOCD que es una herramienta gratuita de depuración en chip, programación en el sistema y prueba de escaneo de límites para varios sistemas ARM, MIPS y RISC-V.Con estas herramientas, podemos generar un binario para poderlo flahsear, el esquema que sigue este sistema es el siguiente:

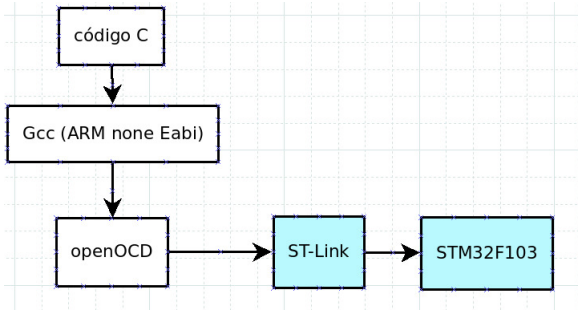

Figura 27: Esquema de flasheo de memoria

4. **Sistema de control de versiones (VCS):** Para proyectos de programación, es crucial utilizar un sistema de control de versiones integrado para poder almacenar cada cambio y poder volver atrás si es necesario o incluso, crear una rama de desarrollo distinta que después se pueda unir a la rama principal. No se va a detallar como funcionan los VCS en este proyecto,

pero si cabe mencionar que para su desarrollo se ha utilizado Git y Github como repositorio remoto, que es público, y se puede consultar en el siguiente enlace:

https://github.com/raiben23/TFG-Comunicacion-optica-cubesat

Parte II
Implementación de software

En esta parte de la presente memoria, se va detallar el diseño y desarrollo del código para cada protocolo y las consideraciones que se han tomado para el ensayo y error de envío de mensajes mediante cada uno de los protocolos. Con las herramientas anteriormente explicadas se ha creado un entorno de trabajo que proporciona un sistema de comunicación a pequeña escala y un entorno de pruebas con el que se ha ido adaptando el código en función de los resultados visualizados en el hardware.
Comenzaré detallando el procedimiento para el diseño de cada protocolo de comunicación, indicando el montaje de hardware en protboar, conexionado, pines utilizados y sistema de debug. También, se va a explicar esquemáticamente, cómo se han adaptado los protocolos y cómo se están aplicando en cada uno de los programas diseñados, y, finalmente, se explicará las funciones diseñadas y el flujo de ejecución para hacer más entendible que esta sucediendo cuando se transmite la información.
Para la aquitectura de hardware del sistema, voy a detallar los materiales que se han utilizado en cada desarrollo:

Figura 28: Arduino UNO Modelo R3 - Placa de desarrollo

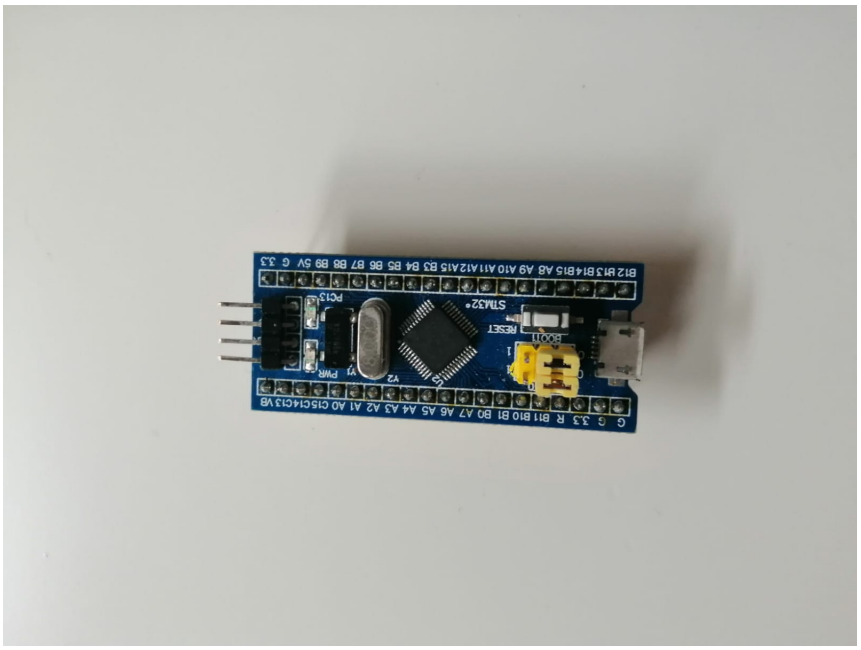

Figura 29: STM32F103C8 "Blue Pill Placa de desarrollo

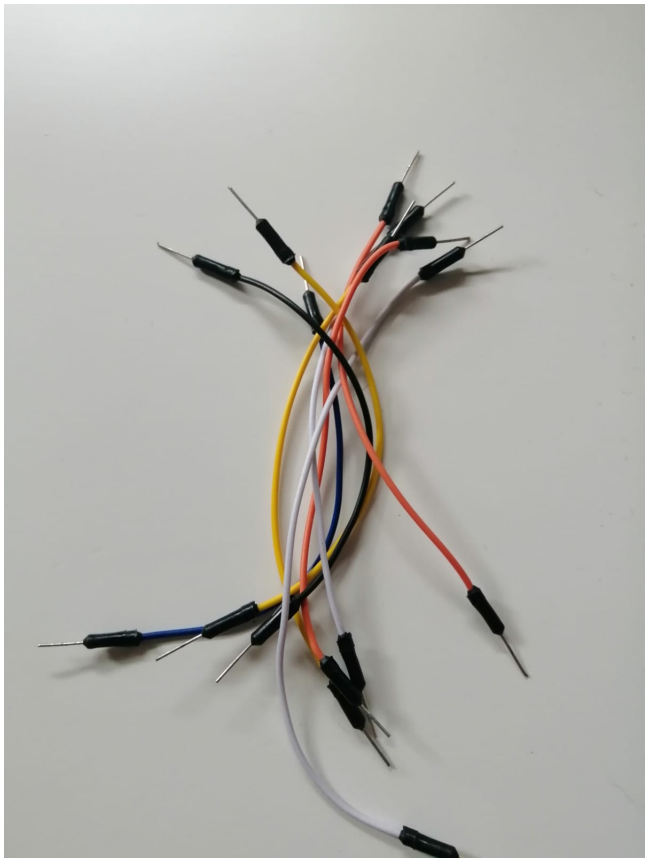

Figura 30: Cable de conexionado macho-macho

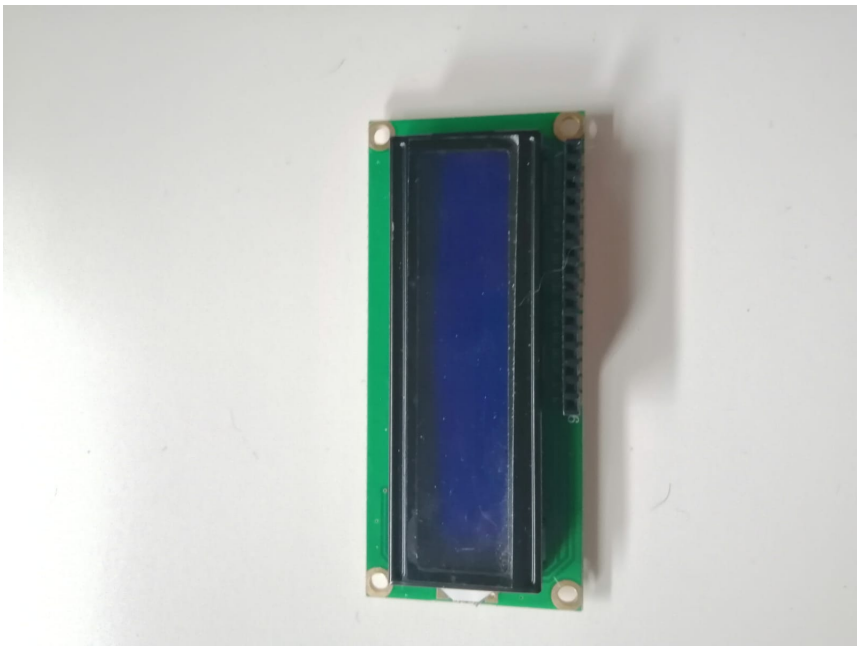

Figura 31: Display LED 1602 A

Figura 32: Diodos LED RGB

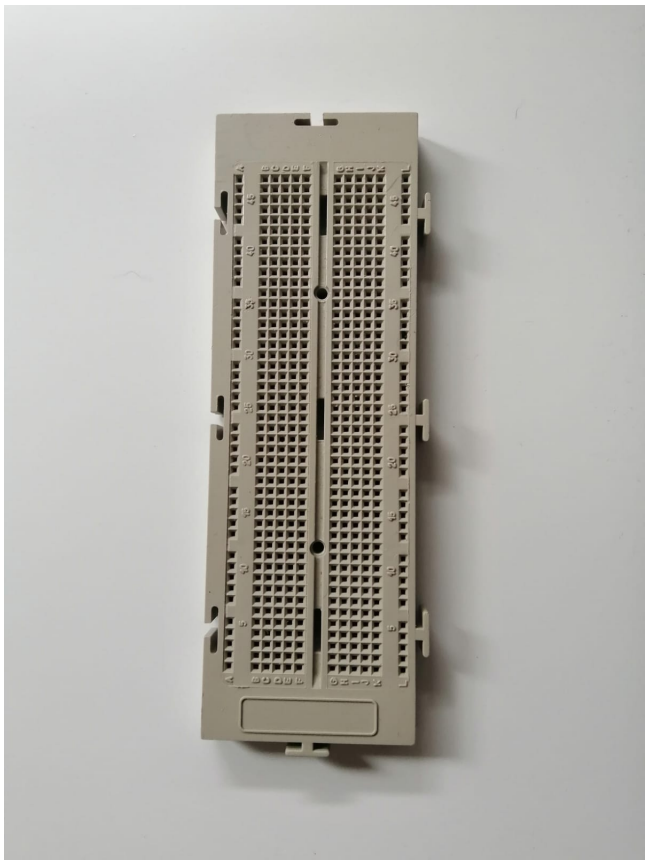

Figura 33: Placa de prototipado Bareboard o Protoboard

Figura 34: Pulsadores de 4 pines

Figura 35: Resistencias Ohmnicas varias

Figura 36: Programador STLink V2

En ciertos materiales, como por ejemplo los LED, los pulsadores o el display, se han utilizado exclusivamente para realizar pruebas de debug como puede ser en el caso de los LED enviar cierta señal, o realizar un RESET con los pulsadores, y en el caso del display, se ha utilizado para mostrar cada mensaje como si del monitor serie se tratase, es posible implementarlo también el proyecto ya que permite comprobar la recepción del mensaje de una forma mucho más visual.
En los siguientes puntos, se va a mostrar el montaje respectivo utilizando estos materiales y se va a detallar el cómo y el porqué se ha desarrollado cada código y se ha utilizado cada pin.

5. Implementación de los protocolos

Para poder implementar en la comunicación basada en estos microcontroladores, los protocolos de comunicación se deben adaptar a nuestras necesidades de diseño, por ello, a nivel de código se ha decido desarrollar una adaptación a estos protocolos para poder vincular, por un lado, la seguridad del sistema a la hora de enviar mensajes, y la compatibilidad entre ambos microcontroladores. Veamos un pequeño recordatorio de como funcionan los protocolos de envío de datos que vamos a utilizar:

- PPM Como su propio nombre indica, la comunicación PPM se basa en el envío de pulsos durante un período T, en el que la amplitud y la duración de estos pulsos son fijas pero su posición es variable. El período y la posición, determinará la forma de codificar los datos a enviar. Una característica crítica que debemos tener en cuenta en este protocolo, es la sincronización de ambos micros ya que la señal de reloj del ARM funciona a más frecuencia que la de Arduino, en el siguiente punto (capítulo 5), se muestra como se ha logrado el sincronismo para que ambas IRQ trabajen sincronizadas.A continuación se expone un ejemplo para enviar 3 pablras de 3 bits, concretamente: 001, 100 y 111. Se realizan en 3 slots de tiempo Tf en los que se envía un pulso de duración Ts en una posición de 8 posibles en el tiempo Tf'.

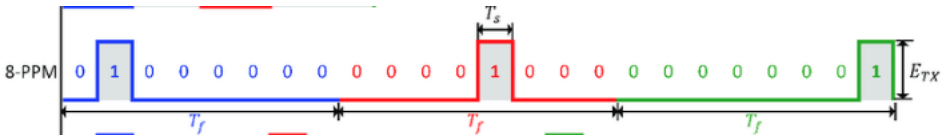

Figura 37: Ejemplo ilustrativo del uso del protocolo PPM2 3bits

- PWM Para el caso del PWM, la amplitud, y la posición del pulso, pueden ser fijas, pero su duración es la variable que nos permite codificar los datos. Recordemos, que esta variable (la duración del pulso), es una relación directa entre el tiempo que el pulso se encuentra encendido (enviando un 1 digital) y la duración máxima del período, de esta forma podemos determinar qué dato es el que se envía conociendo cuánto ha durado el pulso recibido. De nuevo, se podruce el problema del sincronismo entre ambos dispositivos, el cual se ha solucionado de una forma muy similar al PPM y reaprovechando código. Podemos apreciar, en la siguiente imagen, el mismo mensaje que en el ejemplo anterior (Figura 37) utilizando el protocolo PWM, en este caso se dividen los slots de tiempo entre 10 con lo cual para obtener 001 Ton debe ser igual al 20 %(azul) , para obtener 100 Ton debe ser igual al 50 % (rojo) y, para obtener 111, Ton debe ser igual al 80 %:

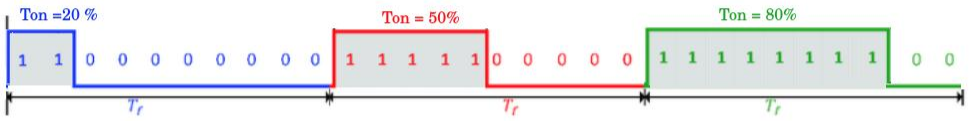

Figura 38: Ejemplo ilustrativo del uso del protocolo PWM

- UART La UART es el controlador de que disponen ambas placas para la comunicación en serie bit a bit, con esta comunicación, podemos envíar datos directamente pero no podemos programar con tanta facilidad las IRQ para lograr la máxima velocidad de transmisión. Cabe recalcar, en cuanto al sincronismo, que ambos microcontroladores deben estar trabajando al mismo *baudrate*. El *baudrate*, es la medida que nos indica la cantidad de bits que podemos enviar por segundo y tiene una relacion proporcional inversa con el tiempo de bit, que nos indica cuánto dura un bit en la trama:

$$T_b = \frac{1}{baudrate}$$

En este proyecto, se ha estado trabajando a un *baudrate* de 9600 baudios, lo cual, implica un tiempo de bit de:

$$T_b = \frac{1}{9600} = 104{,}2\mu s$$

Pero en el diseño, no existe ningún impedimento para trabajar a más baudios, el único condicionante, es que ambos microcontroladores deben trabajar al mismo baudrate para que se puedan entender.

Ilustrando el ejemplo anterior, En UART el 001 seria 0(bit de start) 001 1(bit de stop), es decir: 00011:

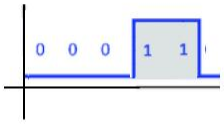

Figura 39: Ejemplo ilustrativo del uso del protocolo UART

5.1. PPM

Se ha diseñado un sistema de comunicación por PPM entre los dos microcontroladores que se disponen de forma que sea capaz de enviar mensajes codificados en ASCII. El sistema, consta del módulo ARM que actúa de emisor, y el microcontrolador de Arduino que hace la función de receptor. Se ha dividido así mismo el código como veremos más adelante, pero, el funcionamiento principal viene dado por los pines configurados en ambos microcontroladores, que actuarán enviando un uno lógico en el emisor y que se recibirá como entrada digital en el receptor. Este uno lógico, esta montado sobre un protocolo PPM modificado a fin de lograr el sincronismo entre ambos módulos ya que tienen tiempos de ejecución distintos. Para el desarrollo del PPM, se ha tenido que rediseñar el protocolo teniendo muy en cuenta la sincronización de ambos microcontroladores. Sabemos que el PPM, codifica los datos determinando la posición del pulso que se envía, en este caso, se ha creado por software una serie de funciones que determinan la posición de este pulso recibido ya sea para el caso del emisor (momento en el que tiene que enviar el pulso) o para el receptor (momento en el que se ha recibido el pulso). Para ello, se deben sincronizar ambos microcontroladores ya que si enviasemos un pulso simplemente, el receptor no tendría referencias para saber en que posicion/momento se ha producido ese pulso.
Una vez sincronizados, y diseñado un sistema de referencia, el pulso recibido, proporciona el dato dada su posición, y este dato, no es más que el decimal del código ASCII del carácter a enviar, en este punto, simplemente se realiza la descodificación (pasar de *int* a *char*) y se da por recibido este paquete a la espera de recibir el siguiente.
Cabe indicar, que para intereses de la aplicación no se va a usar enteramente la tabla ASCII ya que como veremos más adelante, algunos caracteres especiales nos servirán como BIT de STOP. Este sistema, es comúnmente utilizado en el diseño de tramas de comunicación, por ejemplo, para el diseño de software, dado que en el código ASCII se disponen de 8 bits, los primeros carácteres, son los llamados **Caracteres de control** que proporcionan códigos de control especiales para poder ejecutar ciertas acciones[19]:

Caracteres ASCII de control		
00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Figura 40: Tabla ASCII de caracteres de control.

Primeros caracteres de la tabla ASCII, comúnmente utilizados en el diseño de software para realizar acciones específicas. También, muy utilizados en el lenguaje ensamblador para la programación con registros y llamadas a sistema.

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Figura 41: Tabla ASCII de caracteres imprimibles.

Para nuestro diseño, el dato a enviar, dentro de la trama de envío, serán caracteres imprimibles ya que con estos, será sufiente para el propósito de la comunicación. No se va a implementar ningún sistema de seguridad de alto nivel, como podría ser encriptación del dato por claves SHA o RSA (que, en cuyo caso, si que pudieran ser necesarios caracteres especiales ya que estas claves generan un dato calculado por algoritmos y utilizan casi toda la tabla ASCII) , ya que este proyecto, como ya se ha comentado, se quedaría en la capa física y la capa de enlace. En resumen, nuestro paquete de datos, irá codificado en ASCII de forma que el rango será de 32 (espacio) - 126 (equivalencia o tilde).

5.1.1. Emisor - ARM

Configuración del microcontrolador ARM En primer lugar, se ha realizado el código del emisor y, para ello, se deben generar los binarios de configuración del microcontrolador. Ya que este es un micro *open source*, CUBEMX nos permite realizar cualquier configuración que queramos adaptar para nuestra aplicación, para ello, pone a nuestro disposicion un archivo de extensión .ioc que nos proporciona la configuración interna del microcontrolador. Este archivo, se puede modificar manualmente, o bien, utilizar el software proporcionado por el fabricante (CUBEMX). Si bien es cierto, que en algunos momentos durante el debug este archivo se ha modificado a mano, normalmente, se ha utilizado CUBEMX ya que genera el nuevo código de configuración, respetando el código que el usuario ha introducido. Para este sistema de comunicación por PPM, la configuración del microcontrolador y tanto sus pines I/O , como sus pines de programación , y el reloj, han sido configurados de la siguiente forma:

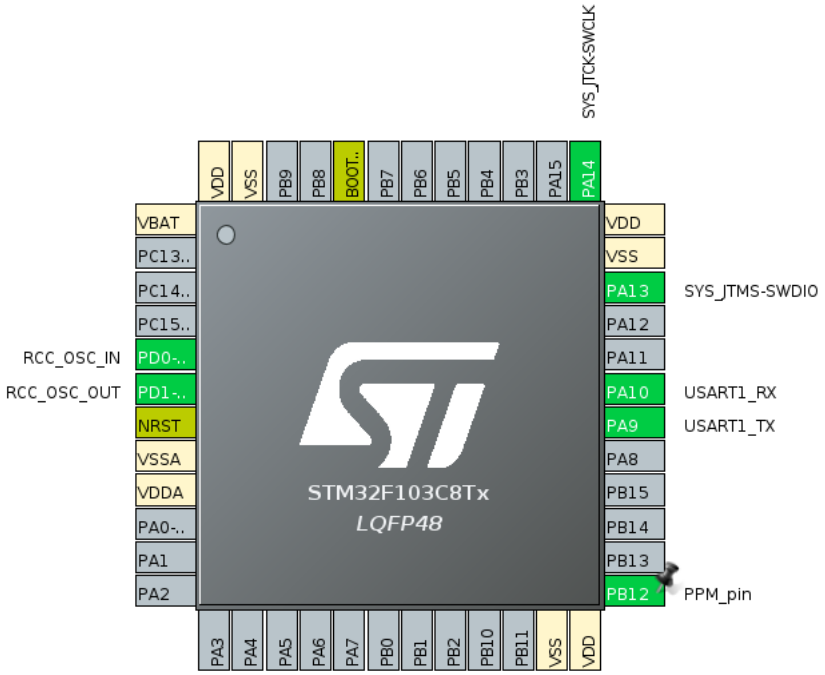

Figura 42: Pinout ARM para PPM

Lo cual, nos genera automáticamente, el siguiente .ioc:

```
#MicroXplorer Configuration settings - do not modify
Mcu.Family=STM32F1
RCC.PLLSourceVirtual=RCC_PLLSOURCE_HSE
ProjectManager.MainLocation=Core/Src
RCC.MCOFreq_Value=72000000
ProjectManager.ProjectFileName=PWM.ioc
PD1-OSC_OUT.Mode=HSE-External-Oscillator
ProjectManager.KeepUserCode=true
PA10.Mode=Asynchronous
Mcu.UserName=STM32F103C8Tx
Mcu.PinsNb=10
ProjectManager.NoMain=false
USART1.BaudRate=9600
RCC.PLLCLKFreq_Value=72000000
PB6.GPIO_PuPd=GPIO_PULLDOWN
ProjectManager.functionlistsort=1-MX_GPIO_Init-GPIO-false-HAL-true,2-SystemClock_Config-RCC-false-HAL-fals
RCC.ADCFreqValue=36000000
ProjectManager.DefaultFWLocation=true
PB6.GPIO_Label=pulsador
PD0-OSC_IN.Signal=RCC_OSC_IN
```

```
PB12.Locked=true
ProjectManager.DeletePrevious=true
USART1.IPParameters=VirtualMode,BaudRate
RCC.APB1CLKDivider=RCC_HCLK_DIV2
PinOutPanel.RotationAngle=0
RCC.FamilyName=M
RCC.SYSCLKSource=RCC_SYSCLKSOURCE_PLLCLK
ProjectManager.StackSize=0x400
PD1-OSC_OUT.Signal=RCC_OSC_OUT
PA13.Signal=SYS_JTMS-SWDIO
Mcu.IP4=USART1
RCC.FCLKCortexFreq_Value=72000000
Mcu.IP2=SYS
NVIC.SVCall_IRQn=true\:0\:0\:false\:false\:true\:false\:false
Mcu.IP3=TIM4
Mcu.IP0=NVIC
PA9.Mode=Asynchronous
Mcu.IP1=RCC
Mcu.UserConstants=
ProjectManager.TargetToolchain=Makefile
VP_TIM4_VS_ClockSourceINT.Signal=TIM4_VS_ClockSourceINT
TIM4.IPParameters=Prescaler,Period
Mcu.ThirdPartyNb=0
RCC.HCLKFreq_Value=72000000
Mcu.IPNb=5
ProjectManager.PreviousToolchain=
RCC.APB2TimFreq_Value=72000000
PA9.Signal=USART1_TX
PB6.Signal=GPIO_Input
PB12.PinState=GPIO_PIN_RESET
Mcu.Pin6=PA14
Mcu.Pin7=PB6
Mcu.Pin8=VP_SYS_VS_Systick
RCC.USBFreq_Value=72000000
Mcu.Pin9=VP_TIM4_VS_ClockSourceINT
TIM4.Prescaler=36
RCC.AHBFreq_Value=72000000
Mcu.Pin0=PD0-OSC_IN
Mcu.Pin1=PD1-OSC_OUT
GPIO.groupedBy=Group By Peripherals
Mcu.Pin2=PB12
PD0-OSC_IN.Mode=HSE-External-Oscillator
Mcu.Pin3=PA9
Mcu.Pin4=PA10
Mcu.Pin5=PA13
```

```
ProjectManager.ProjectBuild=false
NVIC.UsageFault_IRQn=true\:0\:0\:false\:false\:true\:false\:false
NVIC.DebugMonitor_IRQn=true\:0\:0\:false\:false\:true\:false\:false
board=custom
NVIC.SysTick_IRQn=true\:0\:0\:false\:false\:true\:false\:true
ProjectManager.LastFirmware=true
RCC.PLLMUL=RCC_PLL_MUL9
PB12.GPIO_Label=PWM_pin
RCC.VCOOutput2Freq_Value=8000000
ProjectManager.FirmwarePackage=STM32Cube FW_F1 V1.8.0
MxDb.Version=DB.5.0.60
RCC.APB2Freq_Value=72000000
ProjectManager.BackupPrevious=false
MxCube.Version=5.6.1
PA14.Mode=Serial_Wire
File.Version=6
VP_SYS_VS_Systick.Mode=SysTick
NVIC.NonMaskableInt_IRQn=true\:0\:0\:false\:false\:true\:false\:false
NVIC.PendSV_IRQn=true\:0\:0\:false\:false\:true\:false\:false
PB6.Locked=true
TIM4.Period=5000
PA13.Mode=Serial_Wire
ProjectManager.FreePins=false
RCC.IPParameters=ADCFreqValue,AHBFreq_Value,APB1CLKDivider,APB1Freq_Value,APB1TimFreq_Value,APB2Freq_Value
ProjectManager.AskForMigrate=true
Mcu.Name=STM32F103C(8-B)Tx
PB6.GPIOParameters=GPIO_PuPd,GPIO_Label
ProjectManager.HalAssertFull=false
ProjectManager.ProjectName=PWM
ProjectManager.UnderRoot=false
RCC.PLLMCOFreq_Value=36000000
ProjectManager.CoupleFile=true
RCC.SYSCLKFreq_VALUE=72000000
Mcu.Package=LQFP48
RCC.TimSysFreq_Value=72000000
NVIC.ForceEnableDMAVector=true
KeepUserPlacement=false
NVIC.MemoryManagement_IRQn=true\:0\:0\:false\:false\:true\:false\:false
USART1.VirtualMode=VM_ASYNC
ProjectManager.CompilerOptimize=6
ProjectManager.ToolChainLocation=
VP_SYS_VS_Systick.Signal=SYS_VS_Systick
PA10.Signal=USART1_RX
PA14.Signal=SYS_JTCK-SWCLK
ProjectManager.HeapSize=0x200
```

```
NVIC.HardFault_IRQn=true\:0\:0\:false\:false\:true\:false\:false
ProjectManager.ComputerToolchain=false
VP_TIM4_VS_ClockSourceINT.Mode=Internal
NVIC.PriorityGroup=NVIC_PRIORITYGROUP_4
RCC.APB1TimFreq_Value=72000000
NVIC.BusFault_IRQn=true\:0\:0\:false\:false\:true\:false\:false
NVIC.TIM4_IRQn=true\:0\:0\:false\:false\:true\:true\:true
PB12.GPIO_PuPd=GPIO_NOPULL
RCC.APB1Freq_Value=36000000
ProjectManager.CustomerFirmwarePackage=
ProjectManager.DeviceId=STM32F103C8Tx
PB12.GPIOParameters=PinState,GPIO_PuPd,GPIO_Label
PB12.Signal=GPIO_Output
ProjectManager.LibraryCopy=1
```

Como se puede ver, es un archivo de configuración del pinout del microcontrolador, lo cual, indicará a través de código como deberá escoger los modos de cada pin internamente. Para este sistema, se detalla a continuación la configuración esencial de cada pin que estamos usando:

- Pin PA14: SYS-TICK-SWCLK, Pin PA13: SYS-TMS-SWDIO
  - Debug: Configurado en Serial Wire para poder flashear la memoria mediante el serial conectado a ST-LINK (es decir, por USB sin usar JTAG)
  - Timebase Source: SysTick
- Pin PD0: RCC-OSC-IN , Pin PD1: RCC-OSC-OUT
  - High Speed Clock (HSC): Crystal/Ceramic Resonato - Configurado a velocidad máxima del cristal (72 MHz)
  - Low Speed Clock (LSC): Disabled.
  - Volaje a 3.3 V (VDD = 3.3V)
- Pin PB12: PPM-Pin: Pin utilizado para generar el PPM, este pin, irá asociado a la interrupción que saltará debido a la configuración del Timer 4. El timer 4 es un timer de proposito general que dispone de 2 preescalers en serie en continuo conteo, estos preescalers se pueden configurar desde CUBEMX también, y después, mediante código, capturar la interrupción generada cuando el contador llega al límite.
  - Counter Period (Preescaler 1) - 16 bits value: 256
  - Counter Mode: Up - Cuenta ascendente.
  - Preescaler (PSC - Preescaler 2) - 16 bits value: 36
  - Internal Clock division: No division

Es decir, con esta configuración, el contador comenzará a contar de forma ascendente a la velocidad máxima del cristal (72MHz), comenzará a contar hasta que llegue a 256, en ese momento, el

preescaler 2 sumará 1, el primer contador se reinicia hasta llegar de nuevo a 256, que sumará 2 al preescaler 2, se repetirá de forma periódica hasta llegar a 36 (valor configurado en el preescaler 2), en ese momento, envía interrupción del sistema:

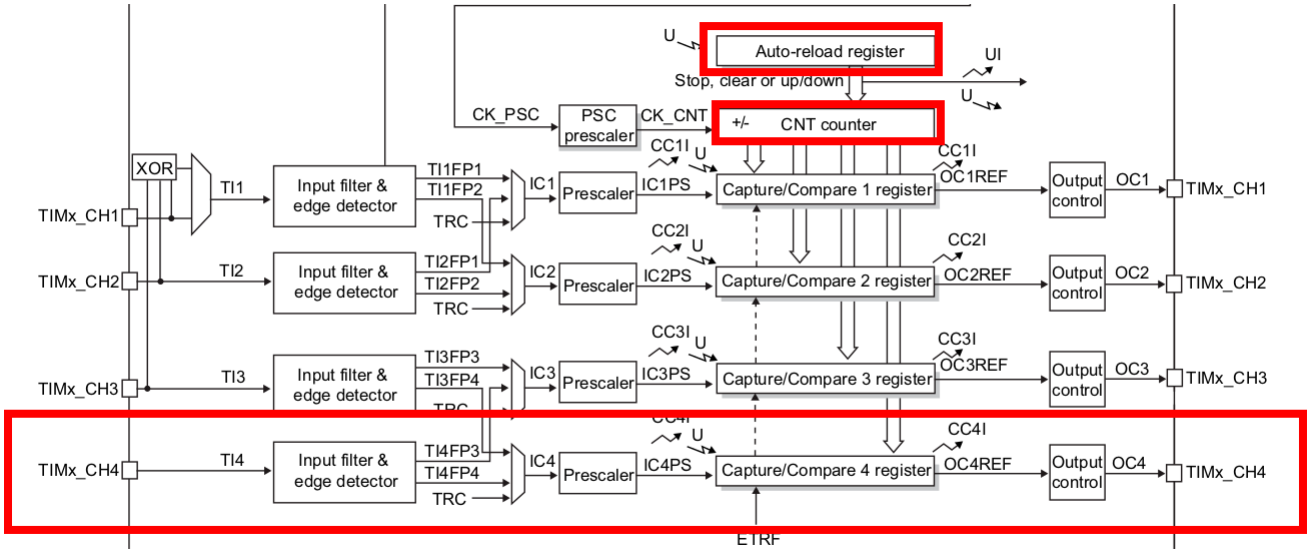

Figura 43: Timer 4 - Esquema de funcionamiento

Como se puede ver en la imagen son contadores ascendentes en serie que se autocargan automáticamente, de forma que el salto de la IRQ se encuentra en un bucle infinito.

Código Fuente

En el caso de ARM y dado que el código base lo genera automáticamente CUBEMX se crean una serie de directorios asociados al proyecto y se preparan los archivos del Core para empezar a programar, en este caso, se ha incluido los archivos:

transmisor.h , tim.h , transmisor.c , tim.c *

y se ha trabajado sobre los archivos:

tim.c , transmisor.c , main.c , gpio.c

*Tambien se han incluido archivos uart.c y usart.c con fines de debug para poder visualizar los mensajes.

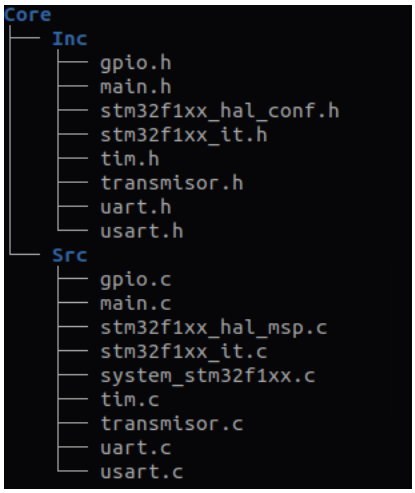

Figura 44: Árbol de código PPM

Se detallará a continuación, el código escrito dando explicación en cada caso, obviando la parte de código generada por CUBEMX ya que es la parte que se ha configurado desde su GUI y que sirve simplemente para configurar el microcontrolador pero no añade lógica al programa:

mainx.h

```
/* Private defines -----------------------------------------------------------*/
#define PPM_pin_Pin GPIO_PIN_12
#define PPM_pin_GPIO_Port GPIOB
```

Se definen las variables del puerto GPIOPort del pin que vamos a utilizar como pin emisor del PPM y el propio pin PIN12

mainx.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "uart.h"
#include "transmisor.h"

/* USER CODE END Includes */

/* Initialize all configured peripherals */
  MX_GPIO_Init();
```

```
16   MX_TIM4_Init();
17   MX_USART1_UART_Init();
18
19 /* USER CODE BEGIN 2 */
20
21   HAL_TIM_Base_Start_IT(&htim4);
22
23   HAL_Delay(1500);
24
25   send_byte(2); // Señal de START
26         while (can_send == SENDING);
27
28   send("Mensaje a enviar");
29   send_byte(4); // señal stop
30
31 /* USER CODE END 2 */
```

- **1-12**: En primer lugar se incluyen las librerías
- **14-19**: Se inicializa los GPIO, timers y usart, en nuestro caso, PB12, TIM4 y UART
- **19-31**:
  - Línea 21. Se inicializa el timer pasado por parámetro (timer4) en modo de interrupción
  - Línea 23. Se le da un delay al sistema de 1.5s, se puede eliminar este delay, es simplemente para fines de debug
  - Línea 25. Se llama a la función *send_ byte()* (25)(veremos cómo funciona, a continuación) pasándole un 2 como parámetro, podría ser otro int, ya que simplemente nos enviará la señal de start hacia Arduino.
  - Línea 26. Esperamos hasta que el flag de envío cambie a modo preparado para enviar de nuevo otro paquete.
  - Línea 28. Enviamos paquete con la función *sent()*
  - Línea 29. Enviamos señal de stop para comunicar el final de transmisión.

transmisor.h

```
1  #ifndef _transmisor_h_
2  #define _transmisor_h_
3
4  // Macros
5          #define FREE        0
6          #define SENDING        1
7
8  // variables
```

```
9         extern volatile unsigned char _dato_a_enviar;
10        extern unsigned char can_send;
11
12        void send(unsigned char* texto);
13        void send_byte(unsigned char dada);
14
15  #endif
```

En este *header file* se definen las constantes o macros y variables que van a ser utilizadas en las funciones:

- FREE : Variable que determina que se puede enviar un nuevo paquete
- SENDING : Variable que indica que la salida está ocupada y que por lo tanto, se debe esperar antes de enviar un nuevo paquete.
- _dato_a_enviar : Variable que almacenará el dato.
- can_send : Variable que almacenará SENDING o FREE
- Se crean por último las funciones abstractas para que sean definidas en el .c

transmisor.c

```
1   #include "transmisor.h"
2   #include "tim.h"
3
4   volatile unsigned char _dato_a_enviar = 0;
5   unsigned char can_send = 0;
6
7   void send(unsigned char *texto)
8   {
9           for (int i = 0; i < sizeof(texto);)
10          {
11                  send_byte(texto[i]);
12                  i++;
13                  while (can_send == SENDING);
14          }
15  }
16
17  void send_byte(unsigned char dada)
18  {
19          contador = 0;                   // reset contador
20          _dato_a_enviar = dada; // cargamos dato para la IRQ
21          can_send = SENDING;           // a partir de aqui, estamos enviando un paquete
22                                                  // y no podremos enviar el siguiente hasta
                                                    ↪  que
```

```
23                                                  // no esté listo el primero
24  }
```

- **1-5** : Se incluyen los headers transmissor.h y tim.h. Se declaran las variables *_dato_a_enviar* y *can_send* y se inicializan a 0.
- **7-15** : Se define la función *send()*, que recibe un array de chars como parámetro. Esta función se encarga de recorrer el array del texto a enviar y llamará a la función *send_byte* para cada caracter pasándolo como parámetro y esperando a que la variable *can_send* sea distinta de "SENDING", es decir, que haya terminado de enviar el carácter anterior.
- **17-24** : Se define la función *send_byte()*, recibe un único carácter como parámetro. Esta función, resetea el contador, carga el dato en la variable que utilizará la IRQ (*_dato_a_enviar*) y deja la variable *can_send* igual a "SENDING", de esta manera, cada vez que salte la interrupción tendrá disponible un char para enviarlo a Arduino.

tim.h

```
1  /* Includes ------------------------------------------------------------------*/
2  #include "main.h"
3
4  /* USER CODE BEGIN Includes */
5  extern unsigned int contador;
6  /* USER CODE END Includes */
7
8  extern TIM_HandleTypeDef htim4;
```

- **Includes** : En este fichero, únicamente es destacable que se crea la variable *contador* y *htim4*, la primera, se encargará de ir acumulando el numero de veces que suman los timers hasta que coincida con el dato a enviar, la segunda, se utilizará para activar la interrupción causada por el timer 4.

tim.c

```
1  /* Includes ------------------------------------------------------------------*/
2  #include "tim.h"
3
4  /* USER CODE BEGIN 0 */
5  #include "transmisor.h"
6  unsigned int contador = 0;
7  .
8  .
9  .
10 .
```

```
11  .
12  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
13  {
14    if (htim->Instance == TIM4)
15    {
16      if (can_send == SENDING)
17      {
18        if (contador == _dato_a_enviar)
19        {
20          HAL_GPIO_WritePin(PPM_pin_GPIO_Port, PPM_pin_Pin, 1);
21        }
22        else if (contador == (_dato_a_enviar + 1))
23        {
24          HAL_GPIO_WritePin(PPM_pin_GPIO_Port, PPM_pin_Pin, 0);
25          can_send = FREE;
26        }
27        contador++;
28      }
29    }
30  }
```

- (Se ha omitido la parte generada automáticamente por CUBEMX y sólo se ha incluido el código de usuario)
- **1-6** : Se incluyen los headers *tim.h* y *transmisor.h* y se crea la variable contador inicializandola a 0.
- **12-30** : *HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)* mediante esta función (proporcionada por la HAL de ST) establecemos que se va a hacer cada vez que salte la IRQ. En primer lugar, se comprueba si la IRQ es debida al timer 4, seguida de la comprobación de que la variable *can_send* se encuentra en estado "SENDING"(seteada en transmisor.c), después, se verifica si el contador es igual al dato a enviar, si esto es así se inicia el envío. Para enviar, se pone el Pin de salida del PPM a 1. En la siguiente iteración, el pin se pone a 0 y se le indica a la variable *can_send* que está preparada para enviar el siguiente dato ("FREE"), en cada interación de la interrupción, se incrementa en 1 el contador.

Con esta configuración y el código diseñado, el emisor ARM mediante PPM, está preparado para ir enviando cada carácter del mensaje que se desee transmitir.

5.1.2. Receptor - Arduino

Dado que Arduino no requiere de configuración previa del microcontrolador,se va a realizar el sincronismo mediante código y con un único archivo:

receptor.ino

```
// ARM, PB12 (PPM)      => Arduino, 3 (PPM)
#define PIN_PPM 3
#define BIT_START 0
#define BIT_DATA 1
#define BIT_STOP 4

#define SIZE_BUFFER 5 // relacionado con el nº de paquetes que envía el ARM

volatile unsigned long tiempo = 0;
volatile char paquetes[(SIZE_BUFFER)];
volatile byte puntero = 0;
volatile byte bit_trama = BIT_START;
volatile bool EOT = false;

void setup()
{
  Serial.begin(9600);
  Serial.println("Init!");

  pinMode(PIN_PPM, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN_PPM), ISR_pin_PPM, CHANGE); // cambio estado pin =
  ↪  ISR PPM
}

void loop()
{
  if (EOT == true)
  {
    imprimir_datos_recibidos();
    reset_variables_ppm();
    EOT = false;
  }
}

void ISR_pin_PPM()
{
  switch (bit_trama)
  {
  case BIT_START: // Sólo se usa para iniciar la trama
    bit_trama++;
    break;

  case BIT_DATA: // Data
  default:

```

```
45      switch (digitalRead(PIN_PPM))
46      {
47      case LOW:
48        tiempo = micros();
49        break;
50
51      case HIGH:
52        char palabra = ((micros() - tiempo) / 133);
53
54        if (palabra <= BIT_STOP) // if señal de stop => end of transmission
55        {
56          EOT = true;
57        }
58        else // byte normal
59        {
60          paquetes[puntero] = palabra;
61          puntero++;
62        }
63        break;
64      }
65      break;
66    }
67  }
68
69  void reset_variables_ppm()
70  {
71    puntero = 0;
72    bit_trama = BIT_START; // reset start para la próxima irq
73    tiempo = 0;
74  }
75
76  void imprimir_datos_recibidos()
77  {
78    Serial.print("Data: ");
79
80    for (int i = 0; i < sizeof(paquetes); i++)
81    {
82      Serial.print((char)paquetes[i]);
83    }
84    Serial.println("");
85  }
```

- **1-13** : En este fragmento de código se definen y declaran todas las variables a utilizar y se inicializan todas ellas

- **15-22** : En el setup principal, se inicia la uart para imprimir el dato recibido (con fines de debug) y se le indica a Arduino que prepare el pin 3 (PIN_PPM) en modo INPUT. Por último, se le indica al micro que capture la interrupción por este pin cada vez que detecte un cambio (modo CHANGE), es decir, cuando se produzca un flanco ascendente o descendente.

- **24-32** : En el bucle principal, se comprueba en cada iteración si **End of transmision** es igual a true, para saber si se han recibido datos. Si es asi, se llama a las funciones *imprimir_datos_recibidos(), reset_variables_ppm()* y por último, se pone EOT a false.

- **34-67** : Esta es la función que se ejecutará cada vez que salte la IRQ y consta de un gran switch-case controlado por la variable *bit_trama*:

  - *bit_trama == BIT_START* En este caso sabemos que es el primer bit, es decir, el bit que indica el inicio de la transmisión, simplemente incrementamos *bit_trama* para que la siguiente vez que salte la interrupción, pase directamente al siguiente caso.
  - *bit_trama == BIT_DATA* En este momento, empieza la recepción de los datos, para ello, se lee el pin PPM, si es un flanco descendente, se toma el tiempo en ese momento (*micros()*), cuando el bit vuelve a cambiar, se comprueba que (como ya se espera), es un flanco ascendente y se vuelve a tomar el tiempo. En este punto, se realiza la resta de tiempos para saber cuántos microsegundos han pasado entre IRQ lo cual, determina la posición del pulso y, con ello, el dato recibido. *Dado que Arduino y ARM funcionan a velocidades muy diferentes, se ha tenido que aplicar un factor correctivo en este punto para evitar posibles errores en la descodificación del carácter en ASCII. La última comprobación que se hace en este caso, es verificar si el dato recibido coincide con el carácter de control que se usa para indicar el fin de la transmisión (carácter 4, se puede ver en la Figura 40), si es así, se indica EOT a true, si no, se guarda el dato detro de la variable *paquete* y se incrementa el puntero para guardar el siguiente dato, con esto , finaliza la IRQ.

- **69-74** : Esta función, se encarga de reinicializar las variables en cada nueva transmisión

- **76-85** : Con esta función, podemos imprimir el mensaje recibido por el monitor serie para comprobar el correcto envío de paquetes.

5.2. PWM

Para el caso del PWM se ha diseñado el protocolo de comunicación utilizando como base, el del PPM tanto en ARM como en Arduino. Recordemos que en una comunicación mediante protocolo PWM, el dato viene codificado en la relación existente entre el tiempo que se encuentra en alto la señal con respecto del período total, lo que se denomina *duty cicle*. Para nuestro proyecto, lo que debe codificar el duty cicle del PWM es, precisamente, los caracteres en ASCII que debemos envíar, de modo que el PWM se ha diseñado teniendo en cuenta que T = 255, de forma que podemos hacer coincidir el duty cicle , directamente con la codificación en ASCII. Es decir, en la siguiente imagen, se puede ver un duty cicle del 50 % , lo cual significa, que si descodificamos el integer 50 en ASCII obtenemos un **2**:

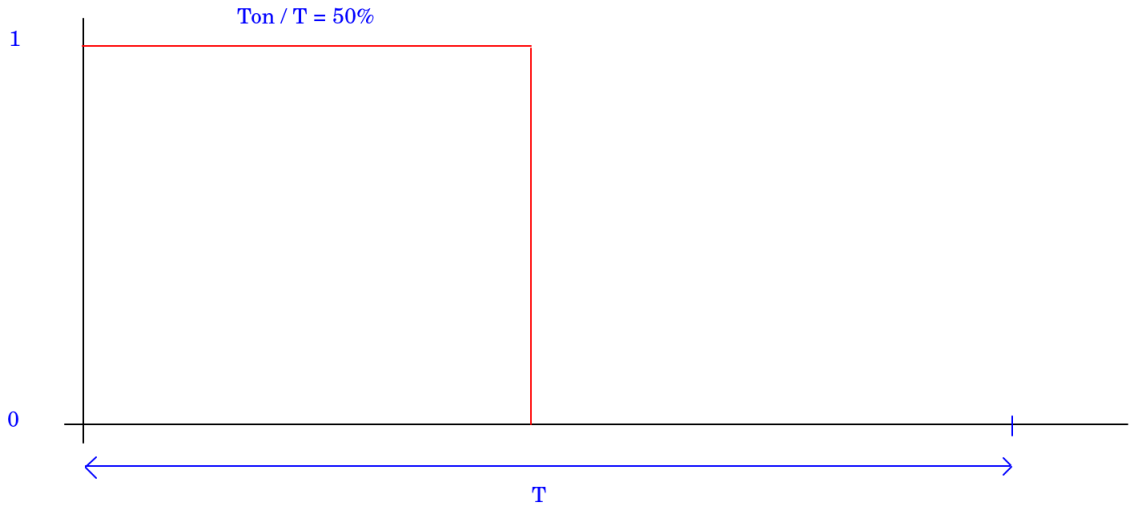

Figura 45: Tabla ASCII de caracteres imprimibles.

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Figura 46: Tabla ASCII de caracteres imprimibles.

Con esta relación llevada al código podemos enviar cualquier información que conste de caracteres ASCII imprimibles regulando el tiempo en que el Pin del PWM se mantiene encendido.

5.2.1. Emisor

Para el emisor, la configuración de ARM es muy similar a la establecida en el PPM , se utiliza el mismo timer, misma configuración de cristal, misma configuración para la programación y flasheo, etc... .Por ello, se va a detallar los pines que se utilizan distintos en esta implementación:

- Pin PB6 - Pin del pulsador: Se ha utilizado durante el diseño, un pulsador en la protoboard para lanzar la IRQ manualmente, a modo de debug meramente.
  - Input mode
  - Pull-down
  - Label: Pulsador
- Pin PB12 - Pin PWM: Este pin es el que enviará la señal con el dato codificado a Arduino.
  - Output level low
  - Output push pull
  - No pull up and no pull down
  - Label: PWM_pin

main.h

```
/* Private defines -----------------------------------------------------------*/
#define PWM_pin_Pin GPIO_PIN_12
#define PWM_pin_GPIO_Port GPIOB
#define pulsador_Pin GPIO_PIN_6
#define pulsador_GPIO_Port GPIOB
```

- En este fichero se definen las variables con los nombres utilizados con el único objetivo de hacer más entendible el código.

main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "uart.h"
#include "transmisor.h"

#define leer_pin_pulsador() HAL_GPIO_ReadPin(pulsador_GPIO_Port, pulsador_Pin)
```

```
13
14 int main(void)
15 {
16
17   /* USER CODE BEGIN 2 */
18
19   HAL_TIM_Base_Start_IT(&htim4);
20
21   /* USER CODE END 2 */
22
23   /* Infinite loop */
24   /* USER CODE BEGIN WHILE */
25   while (1)
26   {
27     if (leer_pin_pulsador() == GPIO_PIN_SET)
28     {
29       HAL_Delay(10);
30
31       if (leer_pin_pulsador() == GPIO_PIN_SET)
32       {
33         while (leer_pin_pulsador() == GPIO_PIN_SET)
34           ;
35
36         send("Holas");
37         send_byte(4);
38         while(can_send == SENDING);
39       }
40     }
41     /* USER CODE END WHILE */
42
43     /* USER CODE BEGIN 3 */
44   }
45   /* USER CODE END 3 */
46 }
```

- **1-10** : Se incluyen las librerías de los otros ficheros
- **12** : Se hace el define de la funcion para leer el pin del pulsador , simplemente para hacer más sencilla la programación.
- **14-46** : Si obviamos la parte de las funciones del pulsador, ya que solo se utiliza en modo debug, básicamente, lo que esta haciendo el main de este archivo es enviar un mensaje en la línea 36 junto con un bit de stop y después, espera hasta que el emisor esté libre para enviar de nuevo el mismo mensaje.

tim.c

```
/* Includes ------------------------------------------------------------------*/
#include "tim.h"

/* USER CODE BEGIN 0 */
#include "transmisor.h"
unsigned int contador = 0;
.
.
.
.
/* USER CODE BEGIN 1 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM4)
  {
    if (can_send == SENDING)
    {
      if (contador == 0) // start
      {
        HAL_GPIO_WritePin(PWM_pin_GPIO_Port, PWM_pin_Pin, 1);
      }
      else if (contador == _dato_a_enviar)
      {
        HAL_GPIO_WritePin(PWM_pin_GPIO_Port, PWM_pin_Pin, 0);
      }
      else if (contador >= 256) // end
      {
        can_send = FREE;
      }

      contador++;
    }
  }
}
/* USER CODE END 1 */
```

- En este fichero, en primer lugar, se comprueba si el sistema esta disponible para enviar el dato. Posteriormente, si el contador esta a 0, lo cual significa que se enviará un nuevo paquete, se pone el pin del PWM a 1, y si no, si el contador vale igual al valor en ASCII del dato a enviar, se pone el pin del PWM a 0, de este modo, el pin PWM se encuentra en 1 únicamente el tiempo que es necesario para que coincida exactamente, el duty cicle con el número en ASCII. Después, cuando termina el período (255 ticks) se pone el contador a 0 y can_send = FREE para quedar a la espera de enviar un nuevo dato.

5.2.2. Receptor

receptor.ino

```
// ARM, PB12 (PWM)      => Arduino, 3 (PWM)
#define NUM_CALIBRACION 2568

#define PIN_PWM 3
#define BIT_STOP 4
#define SIZE_BUFFER 4 // relacionado con el nº de paquetes que envía el ARM

volatile unsigned long tiempo = 0;
volatile char paquetes[(SIZE_BUFFER)];

volatile byte puntero = 0;
volatile bool EOT = false;

void setup()
{
  Serial.begin(9600);
  Serial.println("Init!");
  pinMode(PIN_PWM, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN_PWM), ISR_pin_PWM, CHANGE); // cambio estado pin =
  ↪  ISR PWM
}
void loop()
{
  if (EOT == true)
  {
    imprimir_datos_recibidos();
    reset_variables_pwm();
    EOT = false;
  }
}

void ISR_pin_PWM()
{
  switch (digitalRead(PIN_PWM))
  {
  case HIGH:
    tiempo = micros();
    break;

  case LOW:
    unsigned long palabra = ((micros() - tiempo) / NUM_CALIBRACION );

```

```
42      if (palabra == BIT_STOP) // if señal de stop => end of transmission
43      {
44        EOT = true;
45      }
46      else // byte normal
47      {
48        paquetes[puntero] = palabra;
49        puntero++;
50      }
51
52      break;
53    }
54  }
55
56  inline void reset_variables_pwm()
57  {
58    puntero = 0;
59    tiempo = 0;
60  }
61
62  void imprimir_datos_recibidos()
63  {
64    Serial.print("Data: ");
65
66    for (int i = 0; i < SIZE_BUFFER; i++)
67    {
68      Serial.print(paquetes[i]);
69    }
70    Serial.println("");
71  }
```

- Se ha creado, como en el caso del PPM, un único archivo para el receptor:
  - **1-14** Se definen los pines y variables que se van a utilizar: PIN_PWM -¿3 , BIT_STOP -¿4, SIZE_BUFFER -¿4. Posteriormente, se declaran e inicializan las variables globales del programa, entre las que se encuentran, la variables tiempo, que controla los momentos de recepción de 1 o 0 lógico, la variable paquetes que es el array que almacena el dato recibido, la variable puntero que se utiliza para situar el dato en el array de paquete y, por ultimo, la variable EOT que indica el fin de la transmisión.
  - **14-20** En la función setup(), se inicializa el puerto serie para debuguear los datos, se configura el pin del PWM como input y por ultimo se configura la interrupción para que salte a cada cambio, ya sea por flanco ascendente o flanco descendente.
  - **21-30** En la función loop(),si se esta transmitiendo, se llama a la función para imprimir los datos recibidos, después se resetean las variables en cada iteración y por último se indica que se ha terminado de transmitir.

- **31-54** Cada vez que salta la interrupción, se entra dentro de este switch case controlado por el pin PWM, si es un flanco ascendente, se toma el tiempo en ese momento, si es un flanco descendente, se realiza la resta de tiempos para saber cuánto tiempo ha pasado en 1 el pulso, si este tiempo NO es la palabra de STOP, entonces es dato y se guarda en el array de paquete, si se da el caso de que lo que se recibe es el código de STOP (4) ,entonces se finaliza la transmisión.
- **56-71** La función de reset de variables, simplemente pone el puntero y el tiempo a 0 para reiniciar la recepción de datos. La función de impresión de datos, simplemente va imprimiendo en el monitor serie, los datos que se han ido recibiendo para facilitar el debug.

5.3. UART

El protocolo UART se ha implementado a modo de comparativa con los protocolos precedentes ya que es una comunicación solo disponible por el conexionado de cable, y por lo tanto, no aplicable a la finalidad de este proyecto, que es el diseño de la comunicación por vía óptica.
El protocolo UART, funciona de forma directa, no es necesario utilizar timers ni interrupciones ya que tanto ARM como Arduino, disponen de configuración de sus respectivos pines RX TX que sirven para establecer la comunicación de lectura y escritura respectivamente.

5.3.1. Emisor

uart.h

```
1  #ifndef uart_h_
2  #define uart_h_
3
4  #include "usart.h"
5  #include "stm32f1xx_hal.h"
6
7  #define UART huart1
8
9  UART_HandleTypeDef UART;
10
11 void uart_Init();
12 void uart_put_char(char caracter);
13 void uart_put_char16(uint16_t caracter);
14 void uart_print(char* text);
15 void uart_println(char* text);
16 void uart_print_int(size_t dada);
17 void uart_println_int(size_t dada);
18
19 #endif
```

- En este fichero, se crean todas las funciones a utilizar en la transmisión vía UART.

uart.c

```
#include "uart.h"

void UART_Init()
{
  UART.Instance = USART1;
  UART.Init.BaudRate = 9600;
  UART.Init.WordLength = UART_WORDLENGTH_8B;
  UART.Init.StopBits = UART_STOPBITS_1;
  UART.Init.Parity = UART_PARITY_NONE;
  UART.Init.Mode = UART_MODE_TX_RX;
  UART.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  UART.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&UART) != HAL_OK)
  {
    Error_Handler();
  }
}

void uart_put_char(char caracter)
{
        HAL_UART_Transmit(&UART, caracter, 1, 100);
}

void uart_put_char16(uint16_t caracter)
{
        uint8_t dades[2] =
        {
                (caracter & 0x00FF),
                ((caracter & 0xFF00) << 8)
        };
        HAL_UART_Transmit(&UART, dades, 2, 100);
}

void uart_print(char* text)
{
        HAL_UART_Transmit(&UART, text, strlen(text), 100);
}

void uart_println(char* text)
{
        uart_print(text);
        HAL_UART_Transmit(&UART, "\r\n", sizeof("\r\n"), 100);
}

```

```
45 void uart_print_int(size_t dada)
46 {
47         char text_dada[10]={0,0,0,0,0};
48         itoa( dada , text_dada, 10);
49         HAL_UART_Transmit(&UART, text_dada , sizeof(text_dada), 100);
50 }
51
52 void uart_println_int(size_t dada)
53 {
54         uart_print_int(dada);
55         uart_println("");
56 }
```

- En este fichero, se definen todas las funciones, se ha construido de forma que se puedan utilizar distintas formas de enviar ya sea por chars o bien por conjunto de chars, no entraremos en el detalle del diseño ya que carece de interés para el proyecto, simplemente conocer que este fichero en primer lugar se inicializa la UART con un baudrate de 9600 baudios y se hace la llamada a la función de la HAL de la UART (HAL_UART_Transmit())

main.c

```
1  /* Includes ------------------------------------------------------------------*/
2  #include "main.h"
3  #include "usart.h"
4  #include "gpio.h"
5
6  /* Private includes ----------------------------------------------------------*/
7  /* USER CODE BEGIN Includes */
8  #include "uart.h"
9  .
10 .
11 .
12 .
13 .
14   while (1)
15   {
16     uart_print("Hola");
17     uart_print("\n");
18     HAL_Delay(2000);
19     /* USER CODE END WHILE */
20
21     /* USER CODE BEGIN 3 */
22   }
23 .
```

```
.
.
.
```

- En el fichero main, se envían mensajes de prueba dentro del while principal, con saltos de linea entre ellos, y un delay de 2 segundos cada vez.

5.3.2. Receptor

receptor.ino

```
#include <SoftwareSerial.h>

// ARM PA9,  TX => Arduino 2, Rx
// ARM PA10, RX => Arduino 3, Tx

SoftwareSerial mySerial(2, 3); // RX, TX
String text;

void setup()
{
  Serial.begin(9600);
  mySerial.begin(9600);
}

void loop()
{
  if(mySerial.available())
  {
    text = mySerial.readStringUntil('\n');
    Serial.println(text);
  }
}
```

- Dado que Arduino también proporciona protocolo UART preestablecido directamente en 2 de sus pines, se puede realizar la recepción de los datos, simplemente inicializando dichos pines al mismo baudrate que ARM realiza el envío (9600 baudios). Posteriormente, conforme se va recibiendo el mensaje, se imprime por pantalla.

6. Desarrollo del protocolo

Con el código diseñado hasta este punto, ya disponemos del diseño completo de la capa más baja del prototipo de comunicación, la capa física. Cabe destacar que dado que el alcance del proyecto no lo contemplaba, no se han diseñado sistemas de seguridad como podría ser la encriptación de datos o comprobación de errores a este nivel, con lo cual, con el diseño actual, no habría problemas de comunicación pero sí que el sistema sería susceptible de ser interceptado por un tercero el cual dispondría de los datos transmitidos en plano, sin ningún tipo de encriptación. Dejando de lado lo anterior, en este momento el sistema está habilitado para implementar una nueva capa que utilice esta librería para enviar los datos, dado que el conjunto pertenece a la comunicación de cubesats, los datos que se van a enviar, serán en formato TLE. En el siguiente punto, se explica el diseño de esta capa superior que utiliza el código diseñado en el capítulo anterior, pero también, podría utilizarse para envíar cualquier tipo de mensajes siempre que se encuentren por encima de la capa física del modelo, la cual proporciona ya todas las funciones de la comunicación definidas para cada protocolo.

6.1. Capa de abstracción - TLE

Sea cual sea el protocolo o modulación implementado para la comunicación, el objetivo, sería poder transmitir un dato en formato TLE. El TLE (Two-Line Element), es un formato de transmisión de datos para envíar parámetros orbitales. Consta de dos líneas de datos de 69 caracteres que se pueden usar junto con el modelo orbital SGP4 / SDP4 de NORAD para determinar la posición y la velocidad del satélite asociado. Los únicos caracteres válidos en un TLE son los números 0-9, las letras mayúsculas A-Z, el punto, el espacio y los signos más y menos, ningún otro carácter es válido. Podemos ver, en la siguiente imagen un ejemplo de TLE del satélite NOAA 6:

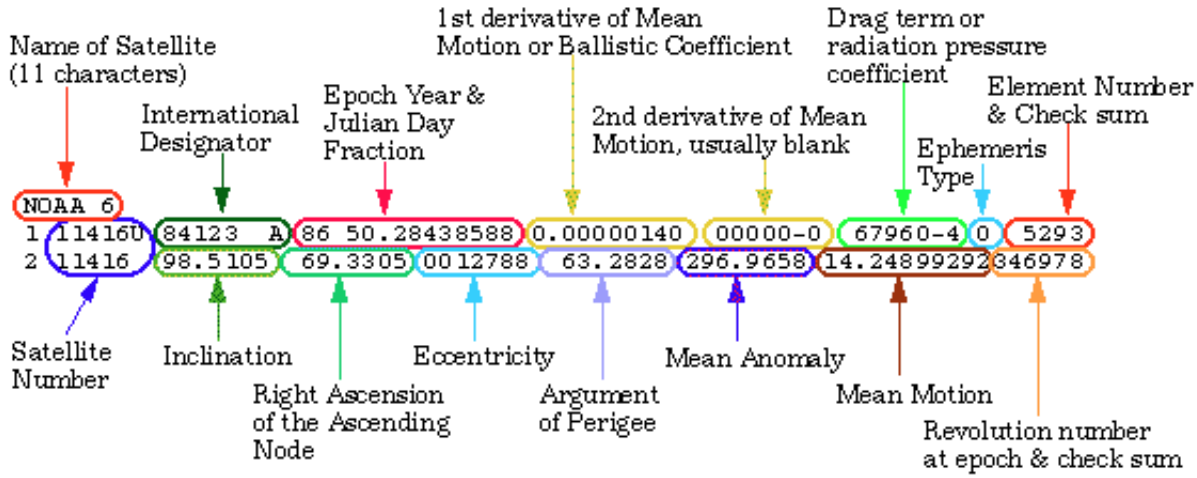

Figura 47: TLE sat NOAA6. NASA Spaceflight [4]

Veremos en la siguiente tabla, el significado y description de cado uno de los elementos que figuran en las 2 líneas que forman el TLE:

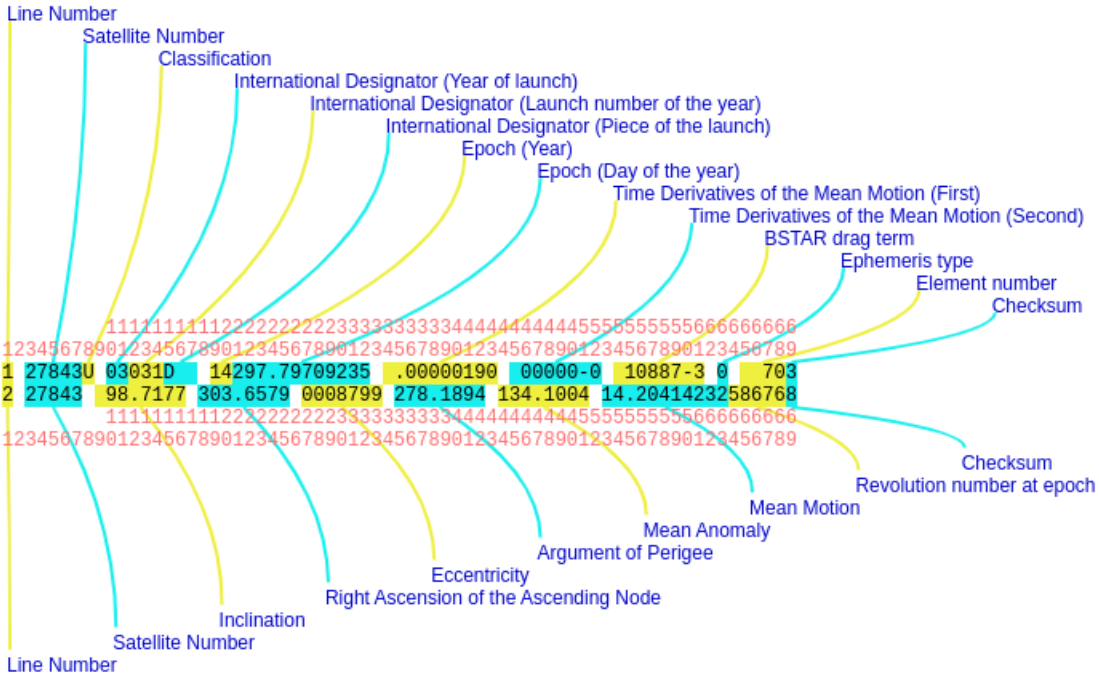

Figura 48: Ejemplo TLE. NASA Spaceflight [4]

1	2	3-7	8	9	10-17	18	19-32	33	34-43	44	45-52	53	54-61	62	63	64	65-68	69

1	2	3-7	8	9-16	17	18-25	26	27-33	34	35-42	43	44-51	52	53-63	64-68	69

Figura 49: Ejemplo TLE. [4]

Línea 1		
1	Número de línea, es decir, 1	
2	Espacio en blanco	
3-7	Número de satélite	
8	Clasificación (U=No clasificado)	
9	Espacio en blanco	
10-17	Designador internacional	10-11: Dos últimas cifras del año de lanzamiento 12-14: Número de lanzamiento del año 15-17: Identificación del componente del lanzamiento
18	Espacio en blanco	
19-32	Época	19-20: Dos últimas cifras del año 21-32: Día del año y fracción del día
33	Espacio en blanco	
34-43	Primera derivada de la velocidad orbital media: $\dot{n}/2$ (revoluciones/día^2)	
44	Espacio en blanco	
45-52	Segunda derivada de la velocidad orbital media: $\frac{1}{6}\ddot{n}$ (revoluciones/ día^3)	
53	Espacio en blanco	
54-61	Término de resistencia, B* (en R_e^{-1}, donde R_e es el radio de la Tierra)	
62	Espacio en blanco	
63	Número 0	
64	Espacio en blanco	
65-68	Número de elemento	
69	Suma de verificación (suma de todas las cifras numéricas individualmente, añadiendo un 1 por cada signo negativo, y pasado a formato 0-9)	

Figura 50: Ejemplo TLE. Line 1[4]

Línea 2	
1	Número de línea, es decir, 2
2	Espacio en blanco
3-7	Número de satélite
8	Espacio en blanco
9-16	Inclinación (grados)
17	Espacio en blanco
18-25	Ascensión recta del nodo ascendente (grados)
26	Espacio en blanco
27-33	Excentricidad
34	Espacio en blanco
35-42	Argumento del perigeo (grados)
43	Espacio en blanco
44-51	Anomalía media (grados)
52	Espacio en blanco
53-63	Velocidad orbital media (revoluciones/día)
64-68	Número de revoluciones en la época
69	Suma de verificación (suma de todas las cifras numéricas individualmente, añadiendo un 1 por cada signo negativo, y pasado a formato 0-9)

Figura 51: Ejemplo TLE. Line 2[4]

6.2. Implementación de código TLE

TLE_defines.h

```
/*
        --------------------Fuente--------------------
        https://en.wikipedia.org/wiki/Two-line_element_set
        ----------------------------------------
*/

#ifndef _TLE_defines_h_
#define _TLE_defines_h_

// ------------------------------------------------------------------------ Defines
↪  genéricos
#define LINE_TITLE 72 // cualquier número que no esté ocupado (1~69)
#define LINE1 73
#define LINE2 74

#define SIZE_LINES 71

// ------------------------------------------------------------------------ Defines Title
#define SIZE_LINE_TITLE 24
```

```
19
20 // ------------------------------------------------------------------------ Defines Line 1
21 #define LINE1_NUM 1 // 1. Line number
22
23 #define LINE1_CATALOG_NUM 3 // 3~7. Satellite catalog number
24 #define LINE1_CATALOG_NUM_END 7
25
26 #define LINE1_CLASSIFICATION 8 // 8. Classification (U=Unclassified, C=Classified, S=Secret)
27
28 #define LINE1_INTERN_DES_LAUNCH_YEAR 10 // 10~11. International Designator (last two digits
   ↪  of launch year)
29
30 #define LINE1_INTERN_DES_LAUNCH_NUM_YEAR 12 // 12~14. International Designator (launch number
   ↪  of the year)
31
32 #define LINE1_INTERN_DESIGN 15 // 15~17. International Designator (piece of the launch)
33
34 #define LINE1_EPOCH_YEAR 19 // 19~20. Epoch Year (last two digits of year)
35
36 #define LINE1_EPOCH_DAY 21 // 21~32. Epoch (day of the year and fractional portion of the
   ↪  day)
37 #define LINE1_EPOCH_DAY_END 32
38
39 #define LINE1_FIRST_DERIV_MEAN_MOTION 34 // 34~43. First Derivative of Mean Motion aka the
   ↪  Ballistic Coefficient [12]
40 #define LINE1_FIRST_DERIV_MEAN_MOTION_END 43
41
42 #define LINE1_SECOND_DERIV_MEAN_MOTION 45 // 45~52. Second Derivative of Mean Motion (decimal
   ↪  point assumed)
43 #define LINE1_SECOND_DERIV_MEAN_MOTION_END 52
44
45 #define LINE1_DRAG_TERM_COEF 54 // 54~61. Drag Term aka Radiation Pressure Coefficient or
   ↪  BSTAR (decimal point assumed)
46 #define LINE1_DRAG_TERM_COEF_END 61
47
48 #define LINE1_EPHEMERIS 63 // 63. Ephemeris type (internal use only - always zero in
   ↪  distributed TLE data) [13]
49
50 #define LINE1_ELEM_SET_NUM 65 // 65~68. Element set number. Incremented when a new TLE is
   ↪  generated for this object.
51 #define LINE1_ELEM_SET_NUM_END 68
52
53 #define LINE1_CHECKSUM 69 // 69. Checksum (modulo 10)
54
55 // ------------------------------------------------------------------------ Defines Line 2
```

```
#define LINE2_NUM 1 // Line number

#define LINE2_CATALOG_NUM 3 // 3~7. Satellite catalog number
#define LINE2_CATALOG_NUM_END 7

#define LINE2_INCLINATION 9                 // 9~16. Inclination (degrees)
#define LINE2_INCLINATION_END 16

#define LINE2_RIGHT_ASCEN 18 // 18~25. Right Ascension of the Ascending Node (degrees)
#define LINE2_RIGHT_ASCEN_END 25

#define LINE2_ECCENTRICITY 27 // 27~33. Eccentricity (decimal point assumed)
#define LINE2_ECCENTRICITY_END 33

#define LINE2_PERIGEE 35 // 35~42. Argument of Perigee (degrees)
#define LINE2_PERIGEE_END 42

#define LINE2_MEAN_ANOMALY 44 // 44~51. Mean Anomaly (degrees)
#define LINE2_MEAN_ANOMALY_END 51

#define LINE2_MEAN_MOTION 53 // 53~63. Mean Motion (revolutions per day)
#define LINE2_MEAN_MOTION_END 63

#define LINE2_REVOLUTIONS 64 // 64~68. Revolution number at epoch (revolutions)
#define LINE2_REVOLUTIONS_END 68

#define LINE2_CHECKSUM 69 // 69. Checksum (modulo 10)

#endif
```

-

TLE.h

```
/*
        -------------------Fuente--------------------
        https://en.wikipedia.org/wiki/Two-line_element_set
        ---------------------------------------------
*/

#ifndef _TLE_h_
#define _TLE_h_

#ifdef __cplusplus // para que pille C, ya que arduino usa C++
```

```
extern "C"
{
#endif

#include "TLE_defines.h"

        void init_line(int opcion, char *_array); // qué linea tocar (title, line1, line2),
        ↪  array de la línea
        void set_line(int opcion, char *text, char* arr_retorno); //(title, line1, line2),
        ↪  texto a cambiar, retorno línea cambiada
        char *get_line(int opcion, char *arr_envio); //(title, line1, line2), retorno línea
        ↪  cambiada

#ifdef __cplusplus
}
#endif

#endif
```

TLE.c

```
#include <string.h>
#include "TLE.h"

void init_line(int opcion, char *_array)
{
    int i = 0;

    for (i = 0; i < SIZE_LINES; i++)
    {
        _array[i] = ' ';

        if (opcion == LINE1 && i == LINE1_NUM)
        {
            _array[i] = '1';
        }
        else if (opcion == LINE2 && i == LINE2_NUM)
        {
            _array[i] = '2';
        }
    }
    _array[SIZE_LINES - 1] = 0; // evitamos overflows
}

void set_line(int opcion, char *text, char *arr_retorno)
```

```
25  {
26      int pos_inicio = opcion;
27      int pos_final = 0;
28
29      switch (opcion)
30      {
31      // Genéricos
32      case LINE_TITLE:
33          pos_inicio = 0;
34          pos_final = strlen(text) - 1;
35          break;
36
37      // Line 1
38      case LINE1_CLASSIFICATION: // 1 solo char
39      case LINE1_EPHEMERIS:
40      case LINE1_CHECKSUM:    // también case LINE2_CHECKSUM:
41          pos_final = opcion;
42          break;
43
44      case LINE1_INTERN_DES_LAUNCH_YEAR: //2 chars
45      case LINE1_EPOCH_YEAR:
46      case LINE1_INTERN_DES_LAUNCH_NUM_YEAR: // 3 chars
47      case LINE1_INTERN_DESIGN:
48      case LINE1_ELEM_SET_NUM: // 4 chars
49      // 5 chars
50      case LINE1_CATALOG_NUM: // también case LINE2_CATALOG_NUM:
51      case LINE2_REVOLUTIONS:
52      case LINE2_ECCENTRICITY:             // 7 chars
53      case LINE1_SECOND_DERIV_MEAN_MOTION: // 8 chars
54      case LINE1_DRAG_TERM_COEF:
55      case LINE2_INCLINATION:
56      case LINE2_RIGHT_ASCEN:
57      case LINE2_PERIGEE:
58      case LINE2_MEAN_ANOMALY:
59      case LINE1_FIRST_DERIV_MEAN_MOTION: // 10 chars
60      case LINE2_MEAN_MOTION:             // 11 chars
61      case LINE1_EPOCH_DAY:               // 12 chars
62          pos_final = opcion + strlen(text) - 1;
63          break;
64      }
65
66      int j = 0;
67      for (int i = pos_inicio; i <= pos_final; i++)
68      {
69          arr_retorno[i] = text[j];
```

```
70          j++;
71      }
72  }
73
74  char *get_line(int opcion, char *arr_lectura)
75  {
76      static char arr_retorno[SIZE_LINES];
77      int pos_final = 0;
78      int j = 0;
79
80      switch (opcion)
81      {
82      case LINE_TITLE:
83          opcion = 0;
84          pos_final = SIZE_LINE_TITLE;
85          break;
86
87      case LINE1:
88      case LINE2:
89          opcion = 0;
90          pos_final = (SIZE_LINES - 1);
91          break;
92
93      case LINE1_NUM: // también case LINE2_NUM:
94      case LINE1_CLASSIFICATION: // 1 solo char
95      case LINE1_EPHEMERIS:
96      case LINE1_CHECKSUM:   // también case LINE2_CHECKSUM:
97          pos_final = opcion;
98          break;
99
100     case LINE1_INTERN_DES_LAUNCH_YEAR: //2 chars
101     case LINE1_EPOCH_YEAR:
102         pos_final = (opcion + 1);
103         break;
104
105     case LINE1_INTERN_DES_LAUNCH_NUM_YEAR: // 3 chars
106     case LINE1_INTERN_DESIGN:
107         pos_final = (opcion + 2);
108         break;
109
110     case LINE1_ELEM_SET_NUM: // 4 chars
111         pos_final = (opcion + 3);
112         break;
113
114         // 5 chars
```

```
115     case LINE1_CATALOG_NUM: // también case LINE2_CATALOG_NUM: , son el mismo
116     case LINE2_REVOLUTIONS:
117         pos_final = (opcion + 4);
118         break;
119 
120     case LINE2_ECCENTRICITY: // 7 chars
121         pos_final = (opcion + 6);
122         break;
123 
124     case LINE1_SECOND_DERIV_MEAN_MOTION: // 8 chars
125     case LINE1_DRAG_TERM_COEF:
126     case LINE2_INCLINATION:
127     case LINE2_RIGHT_ASCEN:
128     case LINE2_PERIGEE:
129     case LINE2_MEAN_ANOMALY:
130         pos_final = (opcion + 7);
131         break;
132 
133     case LINE1_FIRST_DERIV_MEAN_MOTION: // 10 chars
134         pos_final = (opcion + 9);
135         break;
136 
137     case LINE2_MEAN_MOTION: // 11 chars
138         pos_final = (opcion + 10);
139         break;
140 
141     case LINE1_EPOCH_DAY: // 12 chars
142         pos_final = (opcion + 11);
143         break;
144     }
145 
146     for (int i = opcion; i <= pos_final; i++)
147     {
148         if (arr_lectura[i] == 0) // si contiene un char final de string, romperá la línea
149         {
150             arr_lectura[i] = ' ';
151         }
152         else
153         {
154             arr_retorno[j] = arr_lectura[i];
155         }
156         j++;
157     }
158     arr_retorno[j] = 0;
159     return arr_retorno;
```

```
}
```

TLE.ino

```
/*
  Ejemplo de cómo acceder a los distintos strings del TLE.
  No necesita el emisor ARM, simplemente escribe y lee
  La librería está dentro de src/ porque sino la IDE
  de arduino no lo encuentra.
*/

#include "src/TLE/TLE.h"

void setup()
{
  char title[SIZE_LINES];
  char line1[SIZE_LINES];
  char line2[SIZE_LINES];

  init_line(LINE_TITLE, title); // las limpiamos
  init_line(LINE1, line1);
  init_line(LINE2, line2);

  Serial.begin(9600);

  Serial.println("-----------------------------------");
  Serial.println("Ejemplo implementación protocolo TLE");
  Serial.println("-----------------------------------\n\n");

  // Title
  Serial.println("----------------Title----------------\n");
  Serial.print("Título escrito: hola mundo    Título recibido: ");
  set_line(LINE_TITLE, "hola mundo", title);
  Serial.println(get_line(LINE_TITLE, title));

  // Line 1
  Serial.println("----------------Line 1----------------\n");

  // Line title
  Serial.print("Line number recibido: ");
  Serial.println(get_line(LINE1_NUM, line1));

  // Catalog number
  Serial.print("Catalog number escrito: 25544.    Recibido: ");
  set_line(LINE1_CATALOG_NUM, "25544", line1);
```

```
42  Serial.println(get_line(LINE1_CATALOG_NUM, line1));
43
44  // Classification
45  Serial.print("Classification escrito: U.    Recibido: ");
46  set_line(LINE1_CLASSIFICATION, "U", line1);
47  Serial.println(get_line(LINE1_CLASSIFICATION, line1));
48
49  // Launch year
50  Serial.print("Launch year escrito: 98.    Recibido: ");
51  set_line(LINE1_INTERN_DES_LAUNCH_YEAR, "98", line1);
52  Serial.println(get_line(LINE1_INTERN_DES_LAUNCH_YEAR, line1));
53
54  // Num year
55  Serial.print("Num year escrito: 067.    Recibido: ");
56  set_line(LINE1_INTERN_DES_LAUNCH_NUM_YEAR, "067", line1);
57  Serial.println(get_line(LINE1_INTERN_DES_LAUNCH_NUM_YEAR, line1));
58
59  // International designator
60  Serial.print("International designator escrito: A.  Recibido: ");
61  set_line(LINE1_INTERN_DESIGN, "A", line1);
62  Serial.println(get_line(LINE1_INTERN_DESIGN, line1));
63
64  // Epoch year
65  Serial.print("Epoch year escrito: 08.    Recibido: ");
66  set_line(LINE1_EPOCH_YEAR, "08", line1);
67  Serial.println(get_line(LINE1_EPOCH_YEAR, line1));
68
69  // Epoch day
70  Serial.print("Epoch day escrito: 264.51782528.    Recibido: ");
71  set_line(LINE1_EPOCH_DAY, "264.51782528", line1);
72  Serial.println(get_line(LINE1_EPOCH_DAY, line1));
73
74  // 1 derivative mean motion
75  Serial.print("First derivative mean motion: -.00002182.    Recibido: ");
76  set_line(LINE1_FIRST_DERIV_MEAN_MOTION, "-.00002182", line1);
77  Serial.println(get_line(LINE1_FIRST_DERIV_MEAN_MOTION, line1));
78
79  // 2 derivative mean motion
80  Serial.print("Second derivative mean motion: 00000-0.    Recibido: ");
81  set_line(LINE1_SECOND_DERIV_MEAN_MOTION, "00000-0", line1);
82  Serial.println(get_line(LINE1_SECOND_DERIV_MEAN_MOTION, line1));
83
84  // drag term
85  Serial.print("Drag term: -11606-4.    Recibido: ");
86  set_line(LINE1_DRAG_TERM_COEF, "-11606-4", line1);
```

```
87   Serial.println(get_line(LINE1_DRAG_TERM_COEF, line1));
88
89   // ephem type
90   Serial.print("Ephemeris type: 6.    Recibido: ");
91   char s10[1];
92   set_line(LINE1_EPHEMERIS, "6", line1);
93   Serial.println(get_line(LINE1_EPHEMERIS, line1));
94
95   // Element set number
96   Serial.print("Element set number: 292.    Recibido: ");
97   set_line(LINE1_ELEM_SET_NUM, "292", line1);
98   Serial.println(get_line(LINE1_ELEM_SET_NUM, line1));
99
100  // checksum
101  Serial.print("Checksum: 7.    Recibido: ");
102  set_line(LINE1_CHECKSUM, "7", line1);
103  Serial.println(get_line(LINE1_CHECKSUM, line1));
104
105  // Line1 completa
106  Serial.print("Line 1 completa: ");
107  Serial.println(line1);
108
109  // Line 2
110  Serial.println("-----------------Line 2-----------------\n");
111
112  // Line number
113  Serial.print("Line number recibido: ");
114  Serial.println(get_line(LINE2_NUM, line2));
115
116  // Satellite Catalog number
117  Serial.print("Catalog number escrito: 25544.    Recibido: ");
118  set_line(LINE2_CATALOG_NUM, "25544", line2);
119  Serial.println(get_line(LINE2_CATALOG_NUM, line2));
120
121  // Inclination (degrees)
122  Serial.print("Inclination degrees escrito: 51.6416.    Recibido: ");
123  set_line(LINE2_INCLINATION, "51.6416", line2);
124  Serial.println(get_line(LINE2_INCLINATION, line2));
125
126  // Right Ascension of the Ascending Node (degrees)
127  Serial.print("Right Ascension Ascen. Node escrito: 247.4627.    Recibido: ");
128  set_line(LINE2_RIGHT_ASCEN, "247.4627", line2);
129  Serial.println(get_line(LINE2_RIGHT_ASCEN, line2));
130
131  // Eccentricity (decimal point assumed)
```

```
132   Serial.print("Eccentricity escrito: 0006703.    Recibido: ");
133   set_line(LINE2_ECCENTRICITY, "0006703", line2);
134   Serial.println(get_line(LINE2_ECCENTRICITY, line2));
135
136   // Argument of Perigee (degrees)
137   Serial.print("Eccentricity escrito: 130.5360.    Recibido: ");
138   set_line(LINE2_PERIGEE, "130.5360", line2);
139   Serial.println(get_line(LINE2_PERIGEE, line2));
140
141   // Mean Anomaly (degrees)
142   Serial.print("Mean Anomaly escrito: 325.0288.    Recibido: ");
143   set_line(LINE2_MEAN_ANOMALY, "325.0288", line2);
144   Serial.println(get_line(LINE2_MEAN_ANOMALY, line2));
145
146   // Mean Motion (revolutions per day)
147   Serial.print("Mean Motion escrito: 15.72125391.    Recibido: ");
148   set_line(LINE2_MEAN_MOTION, "15.72125391", line2);
149   Serial.println(get_line(LINE2_MEAN_MOTION, line2));
150
151   // Revolution number at epoch (revolutions)
152   Serial.print("Mean Motion escrito: 56353.    Recibido: ");
153   set_line(LINE2_REVOLUTIONS, "56353", line2);
154   Serial.println(get_line(LINE2_REVOLUTIONS, line2));
155
156   // Checksum (modulo 10)
157   Serial.print("Mean Motion escrito: 7.    Recibido: ");
158   set_line(LINE2_CHECKSUM, "7", line2);
159   Serial.println(get_line(LINE2_CHECKSUM, line2));
160
161   // Line2 completa
162   Serial.print("Line 2 completa: ");
163   Serial.println(line2);
164 }
165
166 void loop()
167 {
168 }
```

Parte III
Conclusiones

Este proyecto, partía de una modularización de un proyecto mayor formado por un extenso equipo de estudiantes llevando a cabo, el desarrollo de una red satelital capaz de transmitir información en órbita baja. Concretamente, este proyecto tenía el objetivo de hacer que dos cubesats equipados cada uno de ellos con sus respectivos microcontroladores, fueran capaces de comunicarse mediante la emisión de luz con un diodo láser y su implementación a nivel de software, también junto a otro proyecto en este sentido cuya misión es el desarrollo del hardware.Lamentablemente, a fecha de realización de el presente proyecto, se ve afectado el desarrollo del mismo por la pandemia global causada por el COVID-19, con lo cual, y readaptandose a la actual situación se tomó la decisión de desarrollar el proyecto desde el marco más teórico posible ya que no se disponía de acceso a los laboratorios e implementar así la parte de la óptica en el proyecto. Con lo cual, se ha desarrollado y probado el código del software completo y se ha podido probar su ejecución mediante sistemas cableados en lugar de utilizar el diodo laser, quedando verificado su funcionamiento por cable, a falta de probarlo con un fotoemisor / fotoreceptor.
Independientemente del rediseño de los objetivos del proyecto, se ha podido desarrollar sin ningún problema el código que proporciona la capacidad de comunicación entre dos cubesats. Para ello, desde el inicio de la fase experimental del proyecto, se creó un repositorio de software en la conocida plataforma de GitHub para el control de las versiones de código. GitHub, nos proporciona algunos datos de interés para estas conclusiones:

- Commits: 150
- Branches 3
- 5 merged pull requests
- 95.5 % C 3.6 % C++ 0.9 % Other (además de más de 2200 líneas escritas en LaTeX)

Figura 52: Actividad en cambios de código en la duración del TFG

De cara a futuras implementaciónes, inicialmente, este proyecto consideró realizar un estudio de velocidades e implementación control de errores mediante el diseño de un CRC en la trama del protocolo, podrían ser los siguientes pasos a dar para mejorar la *performance* de la comunicación. A parte de lo anterior, sería conveniente utilizar micros de aun mayor velocidad, y , sobretodo, el mismo modelo para emisor / receptor dado que ello, supone la total eliminación de los problemas de sincronismo a distintas velocidades.
En conclusión, dado que el proyecto ha variado considerablemente de sus expectativas de objetivos inicial y se ha ido adaptando de forma dinámica a la situación en la que se ha desarrollado, y dado que ambos dispositivos implementados son capaces de comunicarse actualmente con los distintos protocolos que se establecieron desde el principio, además de ser capaces de enviar un TLE completo utilizando la librería creada, se podría afirmar que el proyecto asume su objetivo teniendo en cuenta las modificaciones explicadas anteriormente.

Referencias

[1] UPC Tech Talent Center, "Virtualization and networks," 2020.

[2] H. Zimmermann, "OSI Reference Model-The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, 1980.

[3] M. de educación y ciencia, "Proyecto Iris," 2005.

[4] K. Dismukes, "Human Space Flight (HSF) - Realtime Data," 2015.

[5] International Astronomical Union, "IAU Statement on Satellite Constellation," 2019.

[6] Wikipedia, "La Enciclopedia Libre," 2014.

[7] Space Exploration Technologies Corp., "Starlink Discussion National Academy of Sciences," 2020.

[8] OneWeb, "OneWeb Files for Chapter 11 Restructuring to Execute Sale Process," 2020.

[9] JPL, "News — NASA Demos CubeSat Laser Communications Capability."

[10] EuropaPress, "Dos cubeSats de la NASA prueban la comunicación láser en el espacio."

[11] O. Inclination, M. Motion, and M. Anomaly, "Keplerian Elements Tutorial," 2010.

[12] U-blox, "NEO-6 u-blox 6 GPS Modules Data Sheet NEO-6-Data Sheet This document applies to the following products: Name Type number ROM/FLASH version PCN reference," tech. rep., 2011.

[13] J. Moyano Palomares, "Estudio y diseno de dos placas de intercambio de datos de inclinaci ˜ on y posición entre dos cubesats," tech. rep.

[14] Texas Instruments, "Low-Cost Low-Power 2.4 GHz RF Transceiver," tech. rep., 2015.

[15] Thorlabs, "THORLABS - L850P010 Laser Diode."

[16] IEEE, "IEEE SA."

[17] ThorLabs, "FD11A - Si Photodiode, 400 ns Rise Time, 320 - 1100 nm, 1.1 mm x 1.1 mm Active Area."

[18] Arduino, "Arduino."

[19] C. Ascii, "Codigo ASCII."