Proyecto Fin de Grado

Grado en Ingeniería en Tecnologías Industriales

Diseño de plataforma de caracterización electrónica en CubeSat

Autor: Pablo Guijarro Bada

Tutores: Fernando Muñoz Clavero

José Ignacio Mateos Martín

Agustín García Saez

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Diseño de plataforma de caracterización electrónica en CubeSat

Autor:
Pablo Guijarro Bada

Tutores:
Fernando Muñoz Clavero
José Ignacio Mateos Martín
Agustín García Saez

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Proyecto Fin de Carrera: Diseño de plataforma de caracterización electrónica en CubeSat

Autor: Pablo Guijarro Bada

Tutores: Fernando Muñoz Clavero, José Ignacio Mateos Martín y Agustín García Saez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

A mi familia

A mis profesores

A la gente que me ha dado su apoyo en los momentos fáciles y difíciles

Agradecimientos

Dado que con este trabajo termino toda mi etapa de formación en la escuela, me gustaría agradecer a todos los profesores lo que han conseguido enseñarme y lo que han intentado enseñarme. Por otro lado, agradezco el apoyo que he tenido por parte de mi familia en todos los momentos y sobre todo en los momentos en los que me ha costado más sacar adelante todo lo que me he propuesto.

Agradezco a Fernando Muñoz la posibilidad de realizar este proyecto, que pese a que se complicó la finalización del mismo, he aprendido mucho y he conseguido profundizar mucho en todo lo relacionado con el mundo de la electrónica en el espacio.

Por último agradecer a todos mis amigos que han estado apoyándome y por los momentos de descanso que he necesitado para terminar este maravilloso periodo de mi vida. Gracias.

Pablo Guijarro Bada

Sevilla, 2022

Resumen

Este trabajo consiste en la elaboración de la programación que requiere un microcontrolador para el testeo de componentes en el espacio. Este microcontrolador forma parte de un sistema embebido que cumple las directivas del CubeSat. La misión concreta del microcontrolador es estudiar la degradación de una serie de componentes (un diodo y un BJT) cuando se ven afectados por la radiación. Usamos I2C para la comunicación y tomaremos medidas con varios ADC.

Abstract

This work consists of the elaboration of the programming that requires a microcontroller for the testing of components in space. This microcontroller is part of an embedded system that complies with the directives of the CubeSat. The specific mission of the microcontroller is to study the degradation of a series of components (a diode and a BJT) when they are affected by radiation. We use I2C for communication and will take action with various ADCs.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

SW	Software
ADC	Analog to digital Converter
DAC	Digital to Analogue Converter
COTS	Componentes electrónicos que se encuentran sitos para su uso
MCU	Microcontrolador
SCL	System Clock
SDA	System Data

1 INTRODUCCIÓN

1.1 Contexto del trabajo.

Durante mucho tiempo acceder al espacio ha sido muy complejo. Esto se debe a la cantidad de dinero y de tiempo que había que invertir para conseguir enviar algún satélite al espacio. Pero todo cambió en 1999 cuando se comenzaron a desarrollar los primeros nanosatélites.

Estos nanosatélites, al ser más económicos y requerir menos tiempo de diseño y construcción, facilitaban el acceso al espacio tanto a la comunidad científica universitaria como a países y empresas.

Por ello, este trabajo consiste en el diseño, y la implementación de todo el Software de un CubeSat.

1.2 Objetivos y metodología del trabajo.

El objetivo primordial del trabajo es el diseño del Software que requiere un Cubesat. Dado que vamos a utilizar varios ADC, dos I2C y algunos elementos más, necesitaremos comprender cómo funcionan y, a continuación, programar para su correcto funcionamiento.

Sabiendo el objetivo primordial, se ha tenido que realizar el trabajo siguiendo la siguiente metodología:

1. Recopilar información. Se ha investigado información sobre el proyecto, sobre proyectos parecidos y sobre las necesidades que tiene el proyecto.
2. Definimos el objeto de estudio. Se limita el objeto de estudio a unos puntos en concreto, que en nuestro caso son la parte del Software del CubeSat.
3. Información sobre CubeSat. Se han analizado los satélites bajo el estándar CubeSat y así se dispone de más información de alternativas para el diseño de la parte del Software.
4. Creación del software. Se ha realizado el código de programación que implementaremos en el CubeSat.
5. Comprobación. Se ha realizado un análisis de los resultados obtenidos para comprobar que la realización del código de programación ha sido correcta.

2 CONOCIMIENTOS PREVIOS Y ESTADO DEL ARTE

2.1 CubeSats.

Un Cubesat es un nano-satélite que se utiliza principalmente para la investigación espacial. La Universidad Estatal Politécnica de California, San Luis Obispo y el Laboratorio de Desarrollo de Sistemas espaciales de la Universidad de Stanford crearon el estándar CubeSat en 1999, con la finalidad de hacer más sencillo el acceso al espacio a los universitarios. Sin embargo, debido a varios factores, actualmente se utiliza también en el ámbito privado.

Además, este estándar es interesante ya que facilita el acceso frecuente y asequible al espacio, y genera muchas oportunidades de lanzamiento. Se consigue esto reduciendo en gran medida los costes y la cantidad de tiempo necesario para la fabricación de dichos satélites, por lo que hace más fácil su diseño y construcción. El estándar limita el volumen de un módulo CubeSat a 100mm x 100mm x 100 mm. Además, también limita su peso a 1,33 kg.

El módulo que cumple con las limitaciones antes mencionadas es conocido como 1U CubeSat. Un CubeSat puede estar formado de hasta veintisiete módulos de los mencionados, agrupados unos encima de otros. Existe el 2U CubeSat (satélite con dos módulos) y el 3U CubeSat (satélite con tres módulos). En la siguiente figura se muestran todos los módulos.

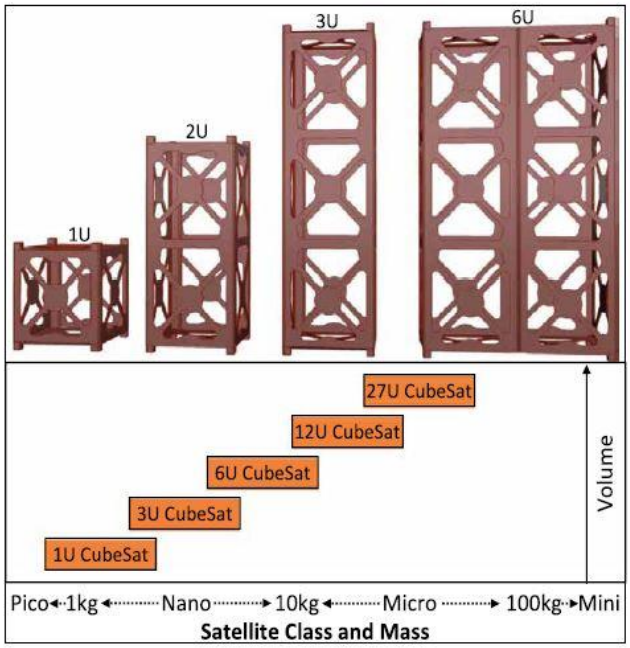

Ilustración 1 Configuraciones básicas de CubeSat

Además de las especificaciones mencionadas anteriormente, el estándar también especifica algunos materiales, cómo se debe de comportar el CubeSat durante el lanzamiento y más cosas.

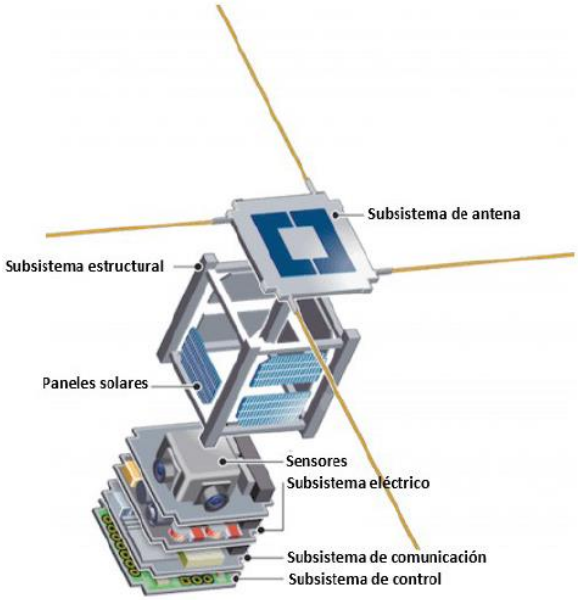

Ilustración 2 Subsistemas del CubeSat

Por otro lado, los CubeSat se pueden dividir en diferentes subsistemas como el estructural, de antena, de control, eléctrico, sensores y paneles solares. Podemos ver en la figura adjunta dónde se encuentra cada subsistema dentro del propio satélite.

Normalmente, la trayectoria de un CubeSat es circular y baja, es decir, entre 400 y 650 Km de altura viajando a 8 km/s aproximadamente. Estas órbitas se denominan LEO. A esa velocidad, el satélite completa entre 14 y 16 órbitas por día. También, como se encuentra a una altura inferior que muchos otros satélites, se ve afectado menos por la radiación solar y cósmica.

2.2 CubeSat frente a otros satélites.

La duración del desarrollo del lanzamiento de un satélite depende de la clase del satélite. En la siguiente tabla podemos ver la comparativa del tiempo de desarrollo con respecto a la clase de satélite.

Tipo de satélite	Peso (Kg)	Tiempo de desarrollo aproximado	Coste (€)
Femto-satélite	Menos de 0.1	Menos de 1 año	Menos de 20 k
Pico-satélite	0.1 - 1	1 año	20 – 150 k
Nano-satélite	1 – 10	1 año	0.15 – 1 M
Micro-satélite	10 – 100	1 año	1 – 8 M
Mini-satélite	100 – 500	3 años	8 – 30 M
Satélite estándar pequeño	500 - 1000	4 años	30 – 100 M
Satélite estándar grande	Más de 1000	Más de 5 años	Más de 100 M

Tabla 1 Satélites según peso, duración del desarrollo y coste

Por lo que podemos observar, el coste del desarrollo de cada satélite varía en función del peso de dicho

satélite, por lo que en el caso del CubeSat, dado que es de un tamaño reducido, observamos que también tiene un tiempo de desarrollo aproximado y un coste menor que otro tipo de satélites de mayor tamaño. En concreto, tardan una media de 8 meses de desarrollo.

Además, hay una gran diferencia en cuanto al lanzamiento de los CubeSats. Esto se debe a que, como ocupan poco volumen y masa, son fáciles de cargar en cualquier vehículo espacial con un coste pequeño. Otro dato interesante es la órbita. Los nanosatélites se lanzan en órbitas bajas, de 400 a 650 Km de altura y su velocidad es cercana a los 8 km por segundo. Por lo tanto, completan una media de 15 órbitas diarias, por lo que tienen condiciones óptimas para observar la Tierra y las comunicaciones, sabiendo que encima están mejor protegidos frente a la radiación solar.

Las aplicaciones para las que pueden seguir son desde la observación de la tierra, comunicaciones, el Internet de las cosas, geolocalización, logística, monitorización de señales y aplicaciones científicas.

2.3 Proyectos relacionados.

Como hemos mencionado anteriormente, en la actualidad existen muchos proyectos relacionados con nanosatélites tanto en el ámbito privado como en el ámbito universitario. Si investigamos proyectos relacionados, podemos encontrar que la ESA (Agencia Espacial Europea) ha realizado varios proyectos con nanosatélites.

Además, la NASA (National Aeronautics and Space Administration) también dispone de muchos programas relacionados con los nanosatélites. Por ejemplo, en 2018 lanzaron un CubeSat diseñado para el espacio profundo: Mars Cube One. Otra misión es la del GTOSat que observará los cinturones de radiación que rodean la Tierra.

Pero además de lo mencionado anteriormente, podemos encontrar empresas privadas que se dedican al diseño, desarrollo, construcción y venta de nanosatélites. Un ejemplo es la Pumpkin Space Systems, que se encuentra en San Francisco. Ha llevado a cabo muchas misiones espaciales tanto para organizaciones gubernamentales, comerciales y educativas.

3 DESCRIPCIÓN DEL HARDWARE

3.1 STM32.

En este proyecto hemos utilizado una placa de desarrollo llamada NUCLEO-L432KC STM32 Nucleo-32. Esta placa de desarrollo tiene varias características interesantes, como por ejemplo que es compatible con Arduino y que tiene varios I2C, que además, pueden utilizarse en configuración de esclavo y de maestro. Más adelante se explicará lo que significan estas dos configuraciones que son necesarias en nuestro proyecto.

Además de las características mencionadas anteriormente, esta placa dispone de un modo de consumo ultra-low-power, por lo que consume muy poco. Tiene tres LEDs: uno para la comunicación USB, otro LED de alimentación y el último es un LED de usuario. Además, dispone de un pulsador de reinicio y opciones flexibles de fuente de alimentación (como un ST-LINK USB VBUS o más fuentes externas).

Por otro lado, cuenta con un Depurador y programador ST-LINK/V2-1integrado con capacidad de reenumeración USB, con amplia compatibilidad en muchos entornos de desarrollo integrados y dispone de ADC de 12 bits 5 Msps. Además de lo mencionado anteriormente, la placa de desarrollo cuenta con gran variedad de elementos más, pero no van a ser necesarios en el proyecto actual, por lo que no van a ser mencionados.

A continuación muestro el pinout del núcleo que estamos utilizando para demostrar que tenemos los canales necesarios para el uso del ADC de nuestro proyecto.

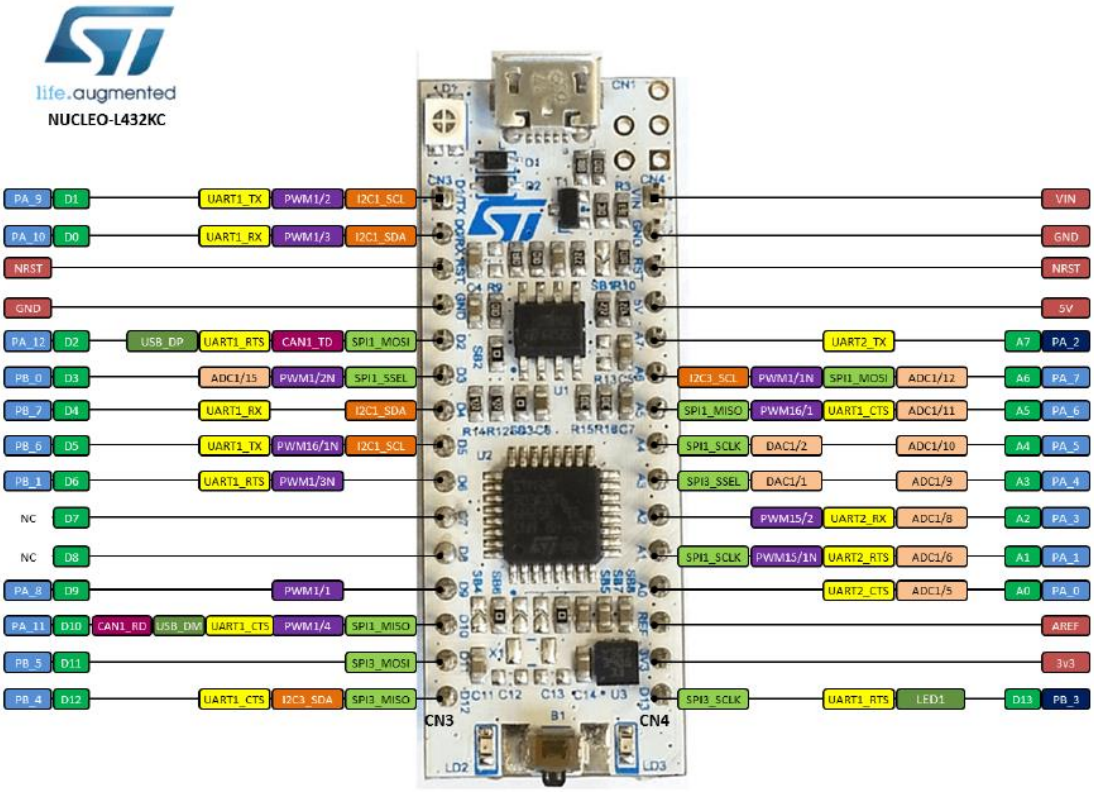

Ilustración 3 Pinout Núcleo L432KC

Nosotros utilizaremos los canales 6, 8, 9, 10, 11 y 12 del ADC que se encuentran en los pines A1, A3, A4, A5, A6 y A7 respectivamente. Por otro lado, dado que vamos a necesitar comunicarnos con un potenciómetro digital, que describiremos más adelante, necesitamos hacer uso del I2C. Para el I2C usaremos los pines A9 y A10. El A9 como SCL y el A10 como SDA.

Por último, necesitaremos hacer uso del USART, y para ello hemos seleccionado los pines A2 y A15. El A2 es para el TX y el A15 es para el RX. No necesitaremos utilizar más pines para ninguna cosa ajena a las mencionadas anteriormente.

3.2 Potenciómetro digital.

Un potenciómetro digital es un dispositivo que simula el funcionamiento de un potenciómetro analógico. El dispositivo que se ha utilizado en este proyecto es el MCP 446. Se controla mediante I2C y es de 7 o 8 bits, en función de lo que nos interese. En el caso de usar la configuración de 7 bits, dispondremos de 129 pasos. Si por el contrario se utilizara la configuración de 8 bits, dispondríamos de 257 pasos. Más adelante se describe con mayor detalle el funcionamiento detallado de nuestro potenciómetro.

3.3 ADC.

En este apartado se ha tratado de explicar cómo funciona un ADC. Un ADC es el dispositivo encargado de convertir una tensión analógica en un código binario. Esto lo consigue mediante un muestreo de la tensión analógica en un instante, después determina el valor binario que representa esa tensión analógica y lo envía por la salida del ADC.

Sin embargo, debido a que disponemos de un número limitado de bits para realizar la conversión a binario, se divide el voltaje del dispositivo en diferentes saltos o pasos y se realiza la comparación en función de dichos saltos. Existen diferentes arquitecturas ADC, pero las tres más populares son la de registros de aproximaciones sucesivas, la del convertidor de canalización y la de Delta-Sigma.

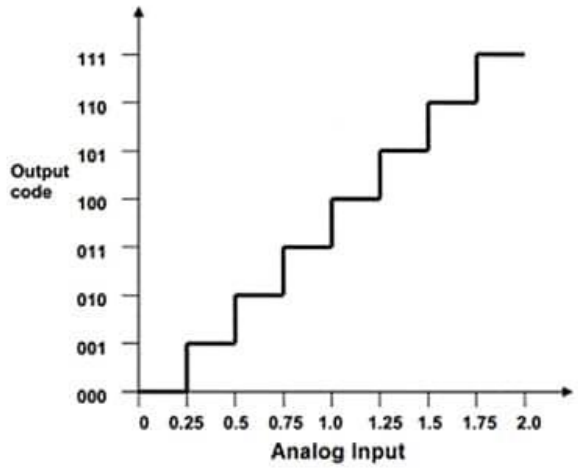

Ilustración 4 ADC de 3 bits

En esta imagen, podemos observar que, si tuviéramos un ADC de 3 bits con dos voltios de referencia, tendríamos en total 8 pasos o saltos distintos. A mayor número de bits, tendremos más precisión. En nuestro caso, el STM32 L432KC dispone de 12 bits, por lo que nos da una precisión mayor que uno de menor número de bits.

Además, la precisión de la conversión depende también del muestreo. A frecuencias de muestreo mayores, tendremos una mejor estimación de la señal analógica.

3.4 Placa de circuito impreso.

Se ha utilizado un circuito impreso realizado en Eagle. El circuito está compuesto de circuitos analógicos que se conforman de dos fuentes de corriente regulables y de tres circuitos con los componentes bajo ensayo: el diodo en polarización directa, el diodo en polarización inversa y el transistor BJT. Además, disponemos de un regulador de voltaje, ya que necesitamos adaptar la tensión de 5 V a los 3,3 V que requiere el microcontrolador para su alimentación.

Por otro lado, disponemos de un conector PC-104, de un conector auxiliar, del potenciómetro digital MCP 446 comentado anteriormente, de un adaptador de nivel para las líneas SDA y SCL del I2C que usaremos para comunicarnos con el potenciómetro digital. Por último, se dispone de conectadores de adaptación para el ensamblado de la placa de evaluación al microcontrolador.

4 DISEÑO DEL SOFTWARE

4.1 Desarrollo del software del dispositivo.

4.1.1 Protocolo I2C.

El protocolo I2C viene de Inter-IC, que significa inter integrated circuits. Lo diseñó Philips Semiconductors a principios de los 80s y está diseñado para conectar circuitos integrados. Una de sus características es que tiene un bus en el que se conectan varios maestros (varios chips) y que cualquiera puede actuar simplemente iniciando una transferencia de datos.

Este protocolo requiere únicamente dos líneas de señal. La velocidad de transferencia es aceptable, normalmente de unos 100 Kbits por segundo, aunque podemos encontrar cosas con mayor velocidad.

El funcionamiento de comunicación es en serie y síncrono. Lo que significa, que mientras una de las señales marca el tiempo (es la señal SCL), la otra señal es la que se utiliza para intercambiar los datos entre los dispositivos (SDA). También se utiliza una tercera señal que es la de referencia (GND).

Para el funcionamiento del I2C hemos de saber que tienen que existir maestros (que son los que inician la comunicación) y esclavos. Los maestros determinan los tiempos y la dirección del tráfico en el bus de datos. El maestro es el genera los pulsos de reloj para la línea SCL. El esclavo no puede generar los pulsos de reloj y reciben señales de comando y de reloj desde los dispositivos maestros.

Por otro lado, existen términos que debemos de conocer antes de ver el funcionamiento de dicho protocolo. El bus libre es el estado en el que podemos encontrar la línea de reloj y la de datos inactivas, es decir, que ambas presentan un estado lógico alto. En este estado es en el que un maestro puede comenzar a usar el bus de datos. En segundo lugar hemos de explicar que existen una secuencia de comienzo y una de parada. La de comienzo se genera cuando la línea de datos está en estado bajo mientras que la de reloj permanece alta y se usa cuando un maestro ocupa el bus. La de parada se produce cuando un maestro deja libre el bus y se produce cuando ambas líneas toman un estado lógico alto.

Las dos secuencias mencionadas anteriormente (la de comienzo y la de parada) son las únicas secuencias que permiten que la línea de datos cambien cuando la línea de reloj está a nivel alto, y como hemos mencionado, son las que señalan el comienzo y el final de la transmisión de un maestro con los dispositivos esclavos.

El funcionamiento de dicho protocolo se produce de la siguiente manera:

1. Cuando ambas señales están en estado lógico alto, se produce la condición inicial de bus libre. Esto significa que cualquier dispositivo que sea maestro puede comenzar a transmitir enviando el bit de inicio (o start). Esto es poniendo en estado bajo la línea de datos pero dejando en alto la línea de reloj

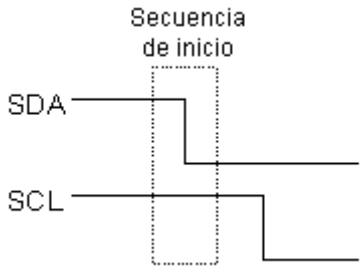

Ilustración 5 Secuencia inicio I2C

2. A continuación, el maestro envía 7 bits que son la dirección del dispositivo que desea seleccionar como esclavo y el octavo bit del primer byte se utiliza para expresar qué se quiere hacer con el dispositivo esclavo: si leer o escribir (R/W).
3. En el caso de que el dispositivo con la dirección enviada anteriormente (A0-A6) se encuentre en el bus, este dispositivo enviará un bit bajo justo después del octavo bit enviado por el dispositivo maestro (ACK). De esta forma, el maestro comprende que el esclavo está disponible y comienza el intercambio de información entre los dispositivos.

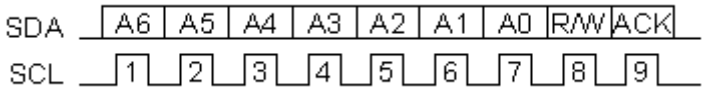

Ilustración 6 Funcionamiento I2C

4. En el caso de que el bit R/W (el de escritura o lectura del dispositivo) estuviera seleccionado a nivel bajo, es decir, para la escritura, el maestro enviará los datos al dispositivo esclavo. Esto se ha de mantener mientras se sigan recibiendo señales de reconocimiento y terminará el contacto cuando se hayan transmitido todos los datos con una secuencia de parada.

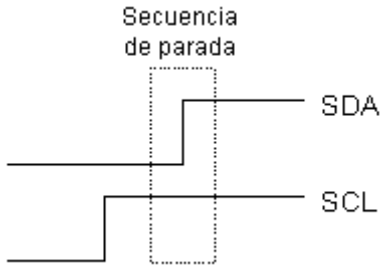

Ilustración 7 Secuencia parada I2C

5. En el otro caso, en el que el bit estaba a nivel alto, es decir de lectura, el maestro generará pulsos de reloj para que el dispositivo esclavo envíe los datos necesarios. Después de cada byte recibido por el maestro, entonces se genera un pulso de reconocimiento. El maestro dejará libre el bus haciendo la secuencia de parada mencionada anteriormente. Sin embargo, se puede producir que el maestro quiera seguir recibiendo, por lo que se puede generar una secuencia de inicio en vez de una de parada, y se suele utilizar para cambiar de dispositivo esclavo o para alterar el modo de escritura a lectura o viceversa.

Dos ejemplos concretos. Si queremos escribir en un dispositivo esclavo, hemos de hacer lo siguiente:

1. Enviar la secuencia de inicio (una vez que el bus de datos está liberado).
2. Enviar la dirección de dispositivo con el bit R/W en bajo, de forma que indicamos es vamos a escribir.
3. Enviamos el número de registro interno en el que deseamos escribir.
4. Enviamos el byte de datos (aunque podemos enviar más si queremos).
5. Enviamos la secuencia de parada.

Si lo que queremos hacer es leer desde un dispositivo esclavo, hemos de realizar el siguiente proceso:

1. Primero debemos informar al dispositivo esclavo desde cuál de sus direcciones internas vamos a leer. La manera de hacerlo es igual que la de escritura: enviamos la secuencia de inicio, enviamos la dirección del dispositivo con el bit R/W en bajo, y enviamos el registro interno desde el que deseamos leer.
2. Enviamos otra secuencia de inicio con la dirección del dispositivo, pero con el bit R/W en alto, para la lectura.

3. Leemos todos los bytes necesarios.
4. Enviamos la secuencia de parada para terminar la conexión.

Podemos ver la secuencia completa en el siguiente dibujo:

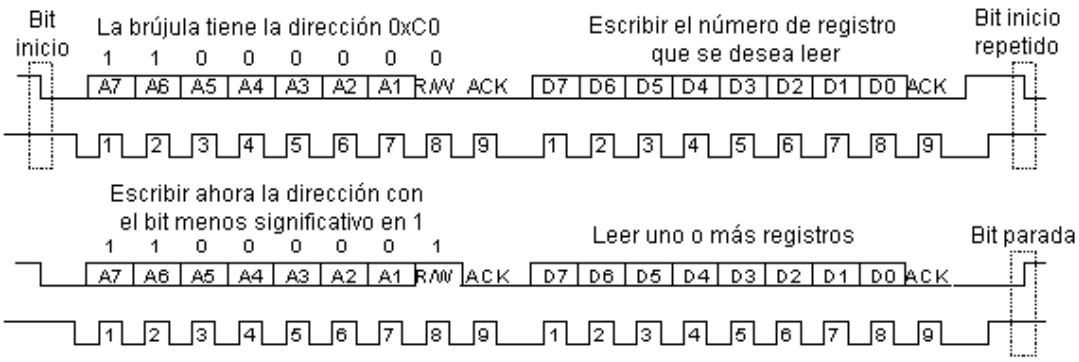

Ilustración 8 Secuencia completa I2C

Existe un pequeño problema en el caso de que el dispositivo esclavo sea un microprocesador y está realizando otras tareas. En ese caso, el microprocesador pasará a su rutina de interrupción, guardará los registros de trabajo, determinará la dirección que desea leer el dispositivo maestro, obtendrá los datos y los pondrá en su registro de transmisión. Todo este proceso conllevará varios microsegundos, lo que hará que el maestro ya haya enviado varios impulsos de reloj sin que el esclavo haya podido enviar nada.

La solución que tenemos es que el esclavo mantenga la línea del reloj en bajo (se le llama estiramiento del reloj). Al recibir el comando de lectura, el esclavo pone la línea en bajo, hace todo lo mencionado anteriormente y después libera la línea del reloj.

4.1.1 Potenciómetro digital.

Para configurar el potenciómetro digital, hemos de saber que vamos a interactuar con él mediante un I2C. Es decir, mediante el protocolo descrito anteriormente, vamos a realizar la comunicación con el dispositivo MCP446 (el potenciómetro digital).

Este potenciómetro tiene las siguientes características: una dirección de 7 bits de esclavo, tiene tres modos de reloj: el standard mode (reloj hasta 100 kHz), fast mode (reloj hasta 400 kHz) y high-speed mode (reloj hasta 3.4 MHz). El reloj que vamos a utilizar nosotros es de 100 kHz, por lo que usaremos el standard mode. Para ello, hemos conectado el pin SCL del potenciómetro al pin SCL del microcontrolador. Igualmente hemos conectado el pin SDA del potenciómetro al del microcontrolador. Por último, hemos conectado los pines A0 y A1 (que son los que marcan las direcciones) a tierra.

Los bits característicos los hemos explicado con anterioridad pero vamos a mostrar las figuras concretas para el MCP446 (el bit de inicio, los bits de datos, el bit de haber recibido el dispositivo esclavo correctamente todo, el bit de repetición de comienzo, el bit de parada, una forma típica de onda de los 8 bits de información y un resumen de la secuencia de bits).

El bit de inicio se da cuando el reloj está en estado lógico alta y el SDA se pone en estado lógico bajo.

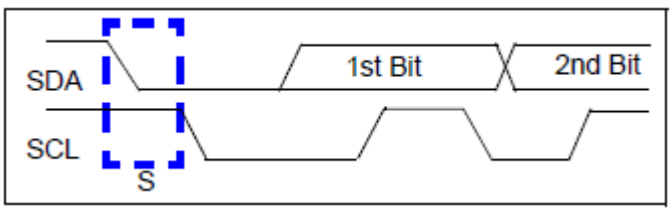

Ilustración 9 Bit inicio I2C

Los bits con los datos se producen mientras el reloj se encuentra en estado lógico alto.

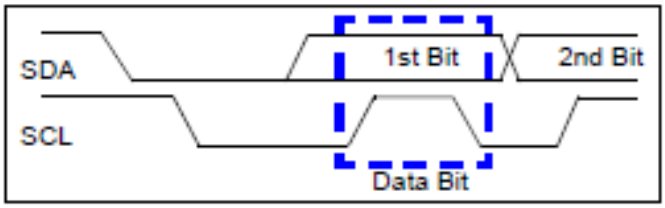

Ilustración 10 Bit de datos I2C

El esclavo, al recibir los datos, manda un bit en estado lógico bajo que significa que ha recibido todos los datos correctamente.

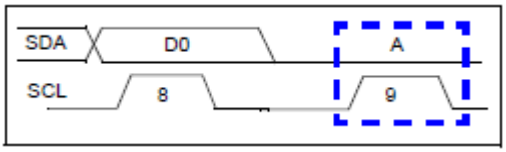

Ilustración 11 Bit de recepción I2C

El bit de repetir el comienzo se envía poniendo el SDA a estado lógico bajo mientras el reloj se encuentra en un estado lógico alto.

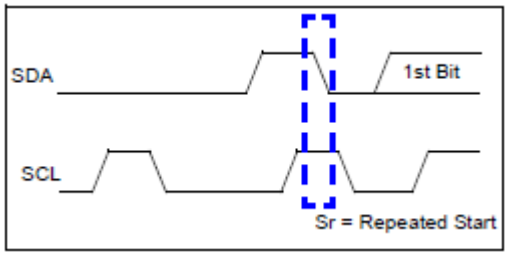

Ilustración 12 Bit repetición I2C

El bit de parada se realiza poniendo el bus de datos en estado lógico alto mientras el reloj se encuentra en el mismo estado lógico.

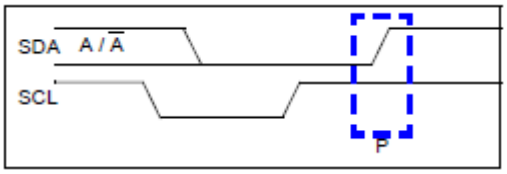

Ilustración 13 Bit parada I2C

Por último, podemos observar una forma típica de onda con los ocho bits de información y un resumen de la secuencia de bits completa.

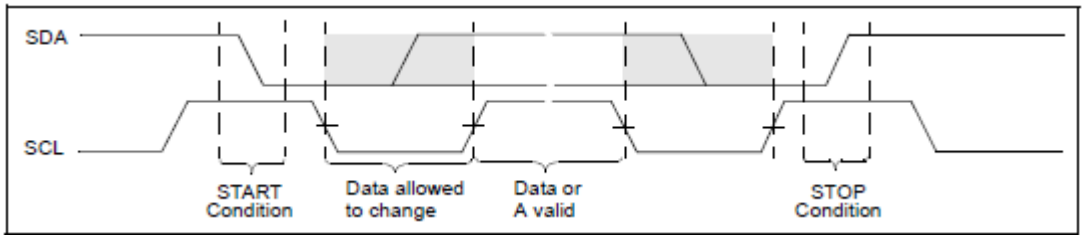

Ilustración 154 Forma típica de onda con 8 bits

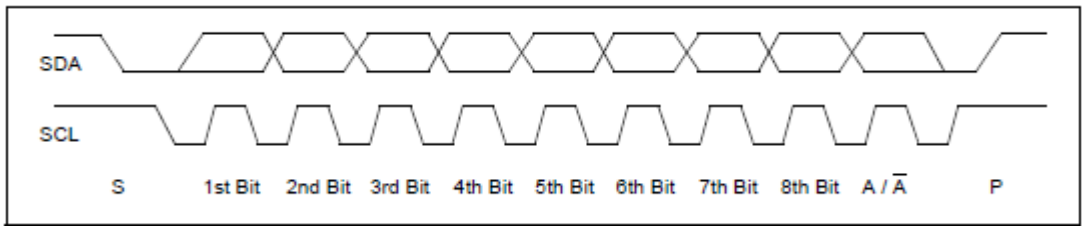

Ilustración 145 Secuencia bits completa I2C

Concretamente, para la dirección del MCP446, hemos de saber que mirando su data sheet podemos encontrar que la dirección tienen un comienzo ya determinado, que podemos ver perfectamente en esta imagen:

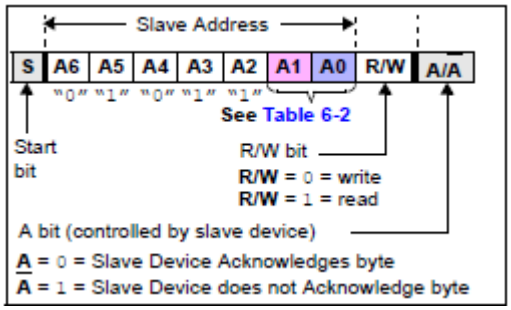

Ilustración 16 Ejemplo del potenciómetro

De forma que los bits A1 y A0 nos permiten variar entre 4 dispositivos distintos (porque podemos hacer 4 combinaciones de dichos bits para crear 4 direcciones distintas).

Los comandos que podemos enviar al potenciómetro son los siguientes:

7-bit Command (1, 2, 3)	Comment
'1000 00d'b	Write Next Byte (Third Byte) to Volatile Wiper 0 Register
'1001 00d'b	Write Next Byte (Third Byte) to Volatile Wiper 1 Register
'1100 00d'b	Write Next Byte (Third Byte) to TCON Register
'1000 010'b or '1000 011'b	Increment Wiper 0 Register
'1001 010'b or '1001 011'b	Increment Wiper 1 Register
'1000 100'b or '1000 101'b	Decrement Wiper 0 Register
'1001 100'b or '1001 101'b	Decrement Wiper 1 Register

Tabla 2 Comandos potenciómetro

Además, en el data sheet del elemento podemos encontrar que, para convertir el comando de 7 bits a 8 bits, hemos de añadir un "0". Por otro lado, la "d" es el bit D8 para escribir con 9 bits. La siguiente figura muestra concretamente qué haría el potenciómetro se recibiera las siguientes series de bits:

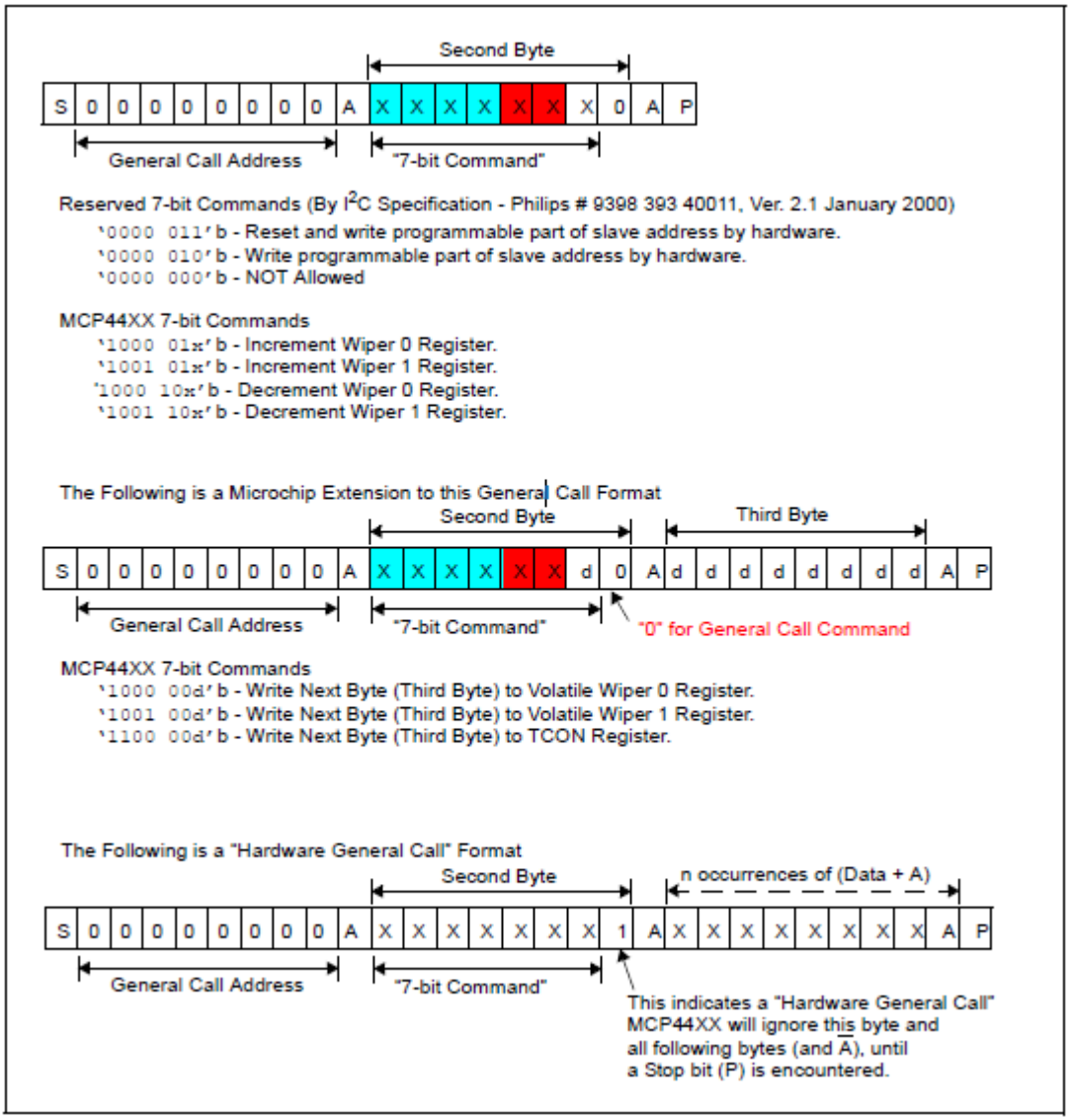

Ilustración 167 Otro ejemplo del potenciómetro

Por lo que la secuencia de escritura, que es la que vamos a necesitar nosotros, quedaría de la siguiente manera:

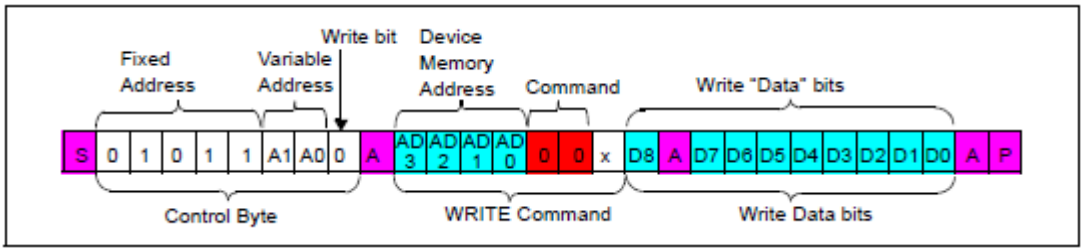

Ilustración 178 Secuencia de escritura I2C

4.2 Entorno de programación (STM32CubeIDE).

Este entorno de programación nos lo ofrece el fabricante del microcontrolador para programar y hacer debug del microcontrolador STM32 de una manera sencilla. El STM32Cube es la combinación de herramientas de software y de librerías que se encuentran integradas, de forma que soluciona cualquier necesidad que se pueda tener en un proyecto con dicho microcontrolador.

4.2.1 Configuración inicial.

En primer lugar, al abrir el STM32CubeIDE podemos hacer una configuración de manera sencilla de cualquier microcontrolador y microprocesador de STM32. Esto se realiza introduciendo la placa de desarrollo. A continuación, nos muestra los pines que tiene dicha placa y nos permite realizar la configuración sencilla tanto de los pines, como del reloj, de los periféricos y del resto de cosas. Esta configuración, una vez hecho de manera visual, la convierte a código de programación.

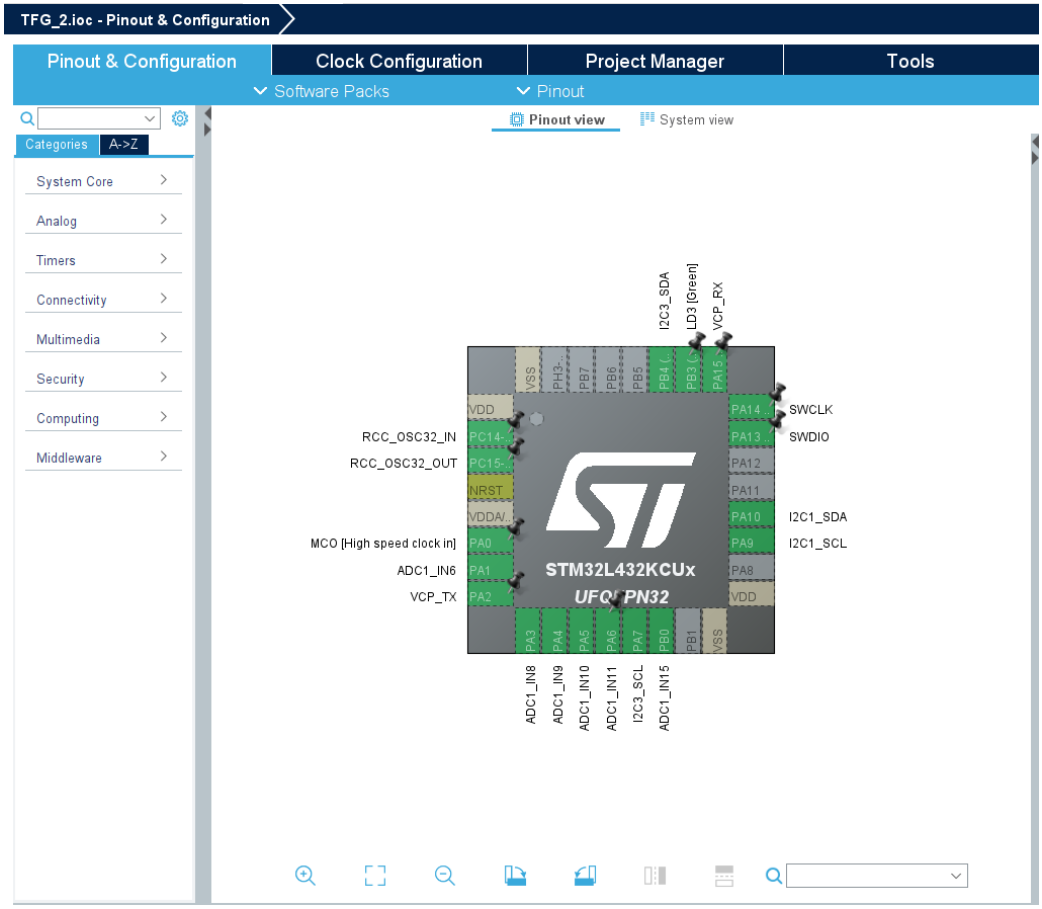

Ilustración 189 Configuración Pinout Nucleo L432KC

Como podemos observar, en nuestro caso hemos seleccionado los elementos que necesitamos en nuestro proyecto, que son: 2 I2C (que se encuentran en los pines PA9, PA10, PB4 y PA7) y seis pines ADC (que se encuentran en los pines PB0,PA6, PA5, PA4, PA3 y PA1).

A continuación, el programa nos indica que se realice la configuración del reloj. Esta configuración la hemos realizado para que el microprocesador se pueda comunicar de manera efectiva con los periféricos y demás elementos del proyecto. La configuración final se puede observar en la siguiente figura.

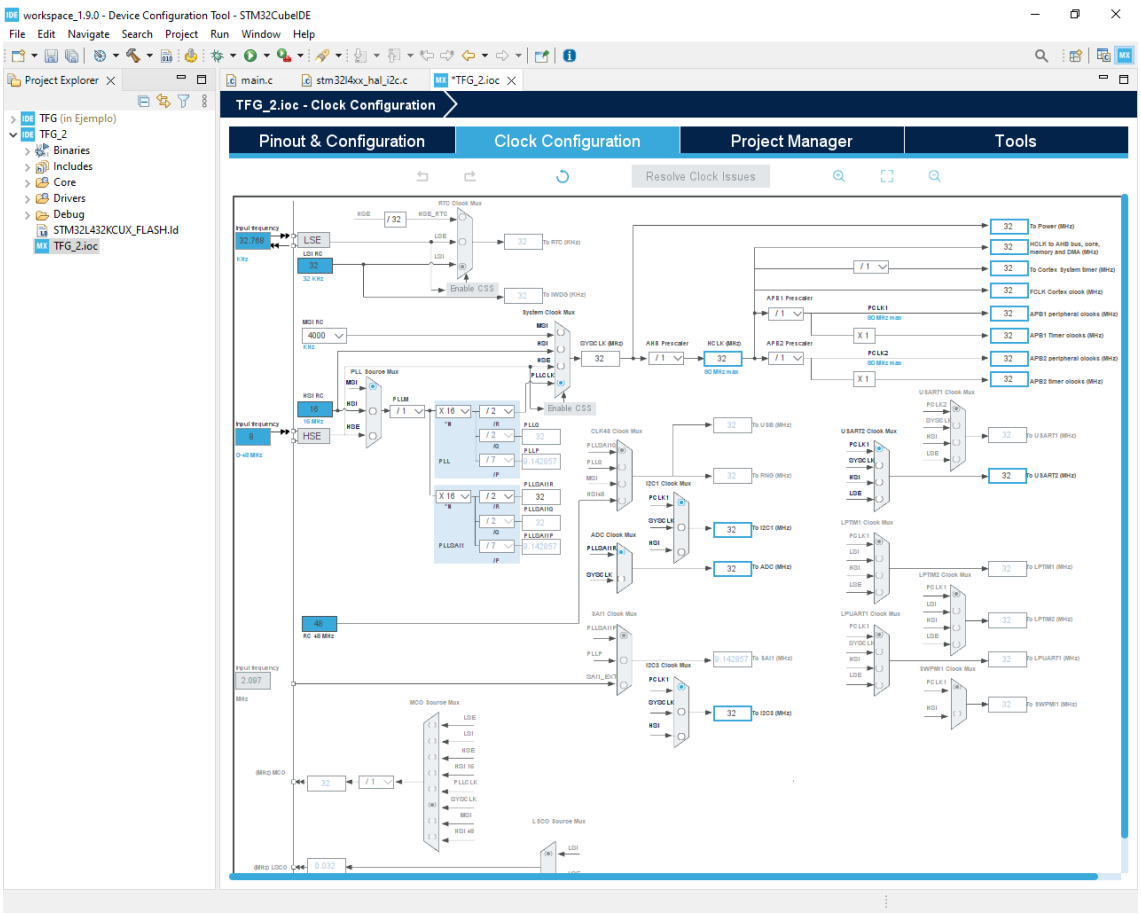

Ilustración 20 Configuración de los relojes del Núcleo L432KC

Al realizar todo lo mencionado anteriormente (tanto la configuración del reloj como la configuración de los pines y de los periféricos) el programa traduce a código C todo lo que se ha introducido de una manera más visual y nos crea un archivo main.c con el código ya generado.

4.2.2 Funciones de configuración.

En el código, aparecen varias funciones distintas. La primera función es HAL_Init(). Su función es inicializar la librería HAL. Esto significa que inicializa la Flash, resetea todos los periféricos y hace algunas cosas más. La función es la siguiente:

```
HAL_StatusTypeDef HAL_Init(void)
{
  HAL_StatusTypeDef  status = HAL_OK;

#if (INSTRUCTION_CACHE_ENABLE == 0)
    __HAL_FLASH_INSTRUCTION_CACHE_DISABLE();
#endif /* INSTRUCTION_CACHE_ENABLE */

#if (DATA_CACHE_ENABLE == 0)
    __HAL_FLASH_DATA_CACHE_DISABLE();
#endif /* DATA_CACHE_ENABLE */

#if (PREFETCH_ENABLE != 0)
  __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
#endif /* PREFETCH_ENABLE */
```

```
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);

  if (HAL_InitTick(TICK_INT_PRIORITY) != HAL_OK)
  {
    status = HAL_ERROR;
  }
  else
  {
    HAL_MspInit();
  }
  return status;
}
```

En segundo lugar, llamamos a la función SystemClock_Config(). Esta función es la que se va a encargar de configurar el reloj del sistema. Esta configuración se realiza con los datos introducidos anteriormente. La función queda de la siguiente manera:

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);

  RCC_OscInitStruct.OscillatorType=RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 16;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
```

```
    Error_Handler();
  }

  HAL_RCCEx_EnableMSIPLLMode();
}
```

También nos falta inicializar los GPIOS, lo relacionado con la inicialización de los I2C y del USART. Eso se realiza con las siguientes funciones que vienen desarrolladas a continuación:

```
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_I2C3_Init();
```

```
static void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00707CBB;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_I2C3_Init(void)
{

  hi2c3.Instance = I2C3;
  hi2c3.Init.Timing = 0x00707CBB;
  hi2c3.Init.OwnAddress1 = 0;
  hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c3.Init.OwnAddress2 = 0;
  hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);

}
```

Por último, nos falta la inicialización del ADC. Dado que vamos a usar varios canales distintos, porque tenemos que utilizar 6 canales para el análisis de todos los componentes electrónicos, la inicialización la realizamos de la siguiente manera:

```
static void MX_ADC1_Init(void);
```

```
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */
```

```
/* USER CODE END ADC1_Init 1 */

/** Common config
*/
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
hadc1.Init.LowPowerAutoWait = DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.NbrOfConversion = 6;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc1.Init.OversamplingMode = DISABLE;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_8;
sConfig.Rank = ADC_REGULAR_RANK_2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_9;
sConfig.Rank = ADC_REGULAR_RANK_3;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_10;
sConfig.Rank = ADC_REGULAR_RANK_4;
```

```
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_11;
  sConfig.Rank = ADC_REGULAR_RANK_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_12;
  sConfig.Rank = ADC_REGULAR_RANK_6;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}
```

De esta forma han quedado inicializados todos los canales del ADC que vamos a utilizar. Sin embargo, cada vez que usemos el ADC, vamos a tener que seleccionar el canal que queremos utilizar. Esto lo vamos a realizar mediante unas funciones que hemos creado. Cada canal tiene la suya propia. A continuación se muestra una función de las mencionadas:

```
void ADC_Select_CH0 (void)
{
      ADC_ChannelConfTypeDef sConfig = {0};
        /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
        */
        sConfig.Channel = ADC_CHANNEL_6;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
        sConfig.SingleDiff = ADC_SINGLE_ENDED;
        sConfig.OffsetNumber = ADC_OFFSET_NONE;
        sConfig.Offset = 0;

        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
}
```

Dado que son 6 los canales que vamos a utilizar, tendremos seis funciones distintas donde lo único que cambiará es el número del canal a utilizar.

4.2.3 Funciones relacionadas con el potenciómetro.

El primer elemento a trabajar es el potenciómetro, que es el que se encargará de hacer un barrido de 0 a 5V para comprobar la reacción de los componentes mencionados anteriormente. Para variar el valor del potenciómetro usaremos comunicación I2C.

Para ello, hemos creado dos funciones, una para aumentar el valor del potenciómetro y otra para poner el valor del potenciómetro a cero. De forma que, en el bucle infinito, vamos a ir aumentando el valor del potenciómetro hasta llegar a los cinco voltios. Entonces, lo que haremos será pasar el valor de 5V a 0V, y comenzar de nuevo.

La primera función creada es la siguiente:

```
void subir_valor_potenciometro(void)
{
	uint16_t dir_pot = 44; /*44 es 0101100 en decimal*/
	uint8_t mensaje0= 66; /*66 es 1000010 en decimal*/
	uint8_t mensaje1= 74; /*74 es 1000010 en decimal*/

	HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje0,sizeof(mensaje0),100);
/*Cambiar valor fuente 0*/
	HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje1,sizeof(mensaje1),100);
/*Cambiar valor fuente 1*/
}
```

Su misión es enviar por el I2C "1000010", lo que hará que el potenciómetro aumente en 1 su valor. Se ha calculado que tendremos que repetir este proceso 182 veces hasta llegar a los 5V. Esto se debe a que si pusiéramos el número 256 serían 7V, por lo que el valor 182 es el que nos da los 5V que buscamos.

Conforme vamos aumentando el valor, también hemos de ir midiendo los ADC y transmitiendo esos datos. Más adelante explicaremos las funciones correspondientes a estos elementos.

Cuando hemos llegado a los 5V, tendremos que bajar todo el voltaje y ponernos de nuevo a 0V. Esto lo realizamos con la siguiente función:

```
void cero_valor_potenciometro(void)
{
	int i;
	uint16_t dir_pot = 44; /*44 es 0101100 en decimal*/
	uint8_t mensaje0= 68; /*68 es 1000100 en decimal*/
	uint8_t mensaje1= 76; /*76 es 1001100 en decimal*/
	for(i=0;i<182;i++)
	{
		HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje0,sizeof(mensaje0),100);
/*Cambiar valor fuente 0*/
		HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje1,sizeof(mensaje1),100);
/*Cambiar valor fuente 1*/
	}

}
```

Su misión es ir bajando los valores del potenciómetro de manera suave, para evitar grandes cambios que puedan degenerar en problemas en el Cubesat. Se sigue el mismo proceso de antes, cambiando simplemente el mensaje que se envía a través del I2C, que en este caso el mensaje es un mensaje que indica una disminución del valor del potenciómetro.

4.2.4 Funciones relacionadas con el ADC.

Como se ha mencionado anteriormente, después de modificar el valor del potenciómetro para que cambie el voltaje de los elementos que estamos midiendo, hemos de realizar la lectura de los seis canales ADC para tomar nota de los datos.

Estos datos serán enviados a través del UART en el mismo momento de la toma de la medida para evitar posibles errores de almacenamiento. Esto lo hemos realizado de esta manera en el prototipo, sin embargo, en el caso de la realización de versiones avanzadas en el futuro, se ha de tener en cuenta que la comunicación con otro satélites se realiza con el I2C, por lo que esta comunicación a través del UART no sería posible.

Para realizar la lectura de los ADC se ha creado una función, cuyo parámetro es el número del canal que queremos leer. La función es la siguiente:

```
void lectura_adc(int);
```

Y la podemos desarrollar de la siguiente manera:

```
/*      leer_ADC;*/
void lectura_adc(int i)
      {
    uint32_t ADC_VAL=0;
      //uint8_t valor[40];
      char valor[40];
     float voltaje;
      int conve;

      if (i==0){
        ADC_Select_CH0();
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, 1000);
        ADC_VAL = HAL_ADC_GetValue(&hadc1);
        HAL_ADC_Stop(&hadc1);
      }
      if (i==1){
              ADC_Select_CH1();
              HAL_ADC_Start(&hadc1);
              HAL_ADC_PollForConversion(&hadc1, 1000);
              ADC_VAL = HAL_ADC_GetValue(&hadc1);
              HAL_ADC_Stop(&hadc1);
            }
      if (i==2){
              ADC_Select_CH2();
              HAL_ADC_Start(&hadc1);
              HAL_ADC_PollForConversion(&hadc1, 1000);
              ADC_VAL = HAL_ADC_GetValue(&hadc1);
              HAL_ADC_Stop(&hadc1);
            }
      if (i==3){
              ADC_Select_CH3();
              HAL_ADC_Start(&hadc1);
              HAL_ADC_PollForConversion(&hadc1, 1000);
              ADC_VAL = HAL_ADC_GetValue(&hadc1);
              HAL_ADC_Stop(&hadc1);
            }
      if (i==4){
              ADC_Select_CH4();
```

```
                HAL_ADC_Start(&hadc1);
                HAL_ADC_PollForConversion(&hadc1, 1000);
                ADC_VAL = HAL_ADC_GetValue(&hadc1);
                HAL_ADC_Stop(&hadc1);
              }
        if (i==5){
                ADC_Select_CH5();
                HAL_ADC_Start(&hadc1);
                HAL_ADC_PollForConversion(&hadc1, 1000);
                ADC_VAL = HAL_ADC_GetValue(&hadc1);
                HAL_ADC_Stop(&hadc1);
              }

            voltaje= (float) ADC_VAL/819;
            voltaje= (float) voltaje*1000;
            conve= (unsigned) voltaje;
            sprintf (valor,"%u",conve);

//          HAL_UART_Transmit(&huart2, valor, sizeof (valor), 10);
//          HAL_UART_Transmit(&huart2,ADC_VAL,sizeof(ADC_VAL),10);
        }
```

Como podemos observar, lo primero que realizamos es la selección del canal que hemos recibido a través de la llamada a la función. A continuación iniciamos dicho ADC, obtenemos el valor y paramos el ADC. Por último, convertimos el valor obtenido y lo transmitimos. Reitero que lo transmitimos a través del UART, cuando en versiones futuras será a través del I2C, siendo nuestro micro controlador un esclavo, no el maestro de dicha conexión.

5 OBTENCIÓN DE MEDIDAS

Tras la realización del código que hemos mencionado anteriormente, se han realizado varias pruebas de comprobación para asegurar el correcto funcionamiento del código en el microcontrolador. Las pruebas han consistido en la realización del barrido del potenciómetro de 0 a 5V, y la comprobación del valor de cada ADC en dicho barrido.

Se han ido tomando nota de los valores del ADC y se han realizado las tablas que se muestran a continuación. Para la comprobación de su correcto funcionamiento, se van a comparar las tablas con los datos tomados del microcontrolador y las tablas con la simulación de lo que debería dar.

Podemos observar que en todos los casos son muy parecidas salvo en algunos puntos, en los que suponemos que ha debido de haber algún error en la toma de medidas. Sin embargo, dado que la mayoría de los puntos coinciden o son bastante cercanos, podemos dar por válida la programación del microcontrolador.

5.1 Fuente de tensión.

En primer lugar mostramos la simulación del V(ADC0). Esta es la simulación de la fuente de tensión. Dado que los ADC no soportan más de 3V, la tensión del V(ADC0) realmente hemos tenido que disminuirla para el correcto funcionamiento del microcontrolador. Esta es la adaptación del circuito:

Ilustración 191 Adaptación para el canal ADC0

Y la simulación de dicha fuente de tensión quedaría de la siguiente manera:

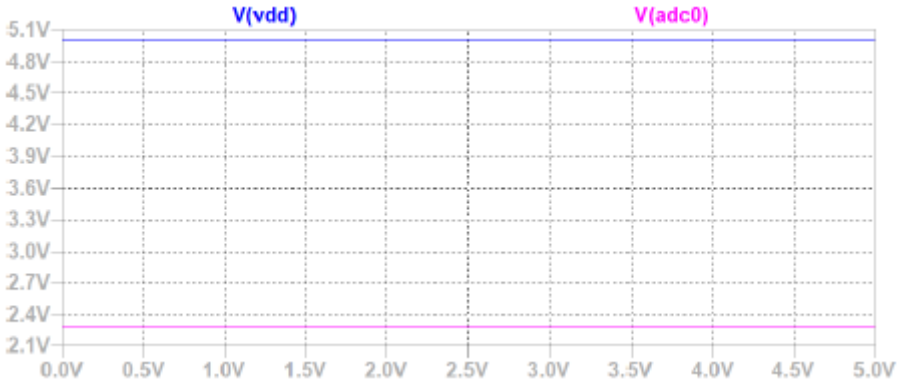

Ilustración 202 Tensión simulación ADC0

Por lo que podemos observar, el valor del V(ADC0) se encuentra cerca del 2.27 en todo momento. A continuación muestro los datos tomados por nuestro microcontrolador. Recuerdo que se han ido tomando los datos uno a uno y se ha obtenido la gráfica que se introduce a continuación:

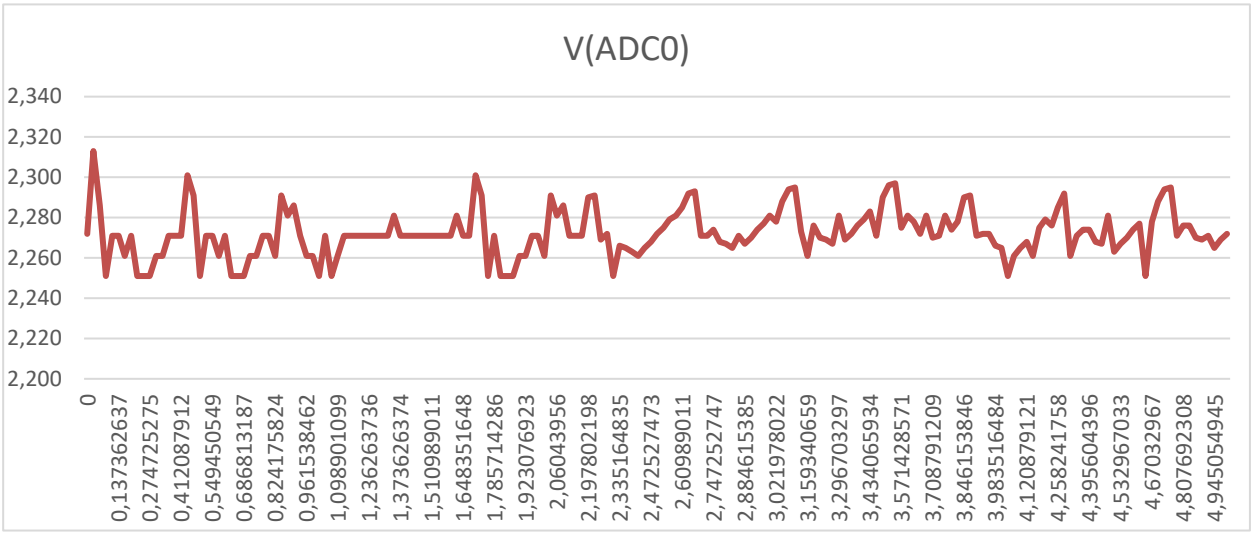

Ilustración 213 Tensión real ADC0

Como podemos comprobar, los datos nos ha dado valores muy cercanos a los calculados teóricamente, por lo que parece que el barrido de la fuente de tensión se realiza de una manera correcta.

5.2 Diodo.

5.2.1 Polarización directa.

El primer componente con el que estamos ensayando es el diodo. En primer lugar vamos a realizar el ensayo para la polarización directa. Para el diodo, tenemos los adc ADC1 y ADC2, que también se encuentran con divisores de tensión iguales a los mencionados para la fuente de tensión. Se ha añadido una resistencia de 100 kilo ohmios, para limitar la corriente que le entra al diodo por la base.

El circuito para la simulación queda de la siguiente manera:

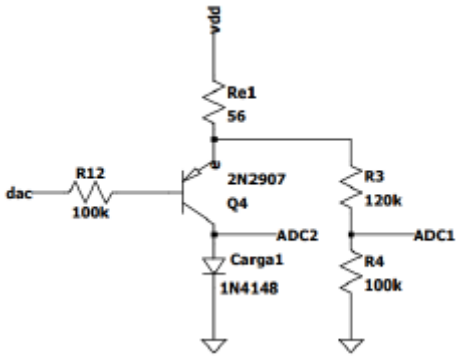

Ilustración 224 Circuito polarización directa diodo

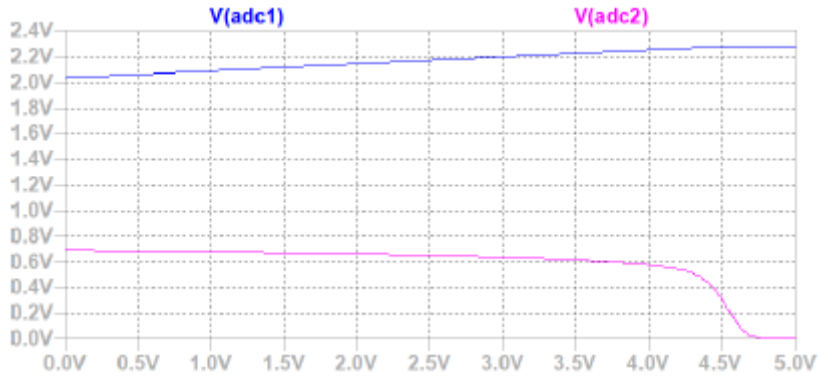

Ilustración 235 Tensiones simulación ADC1 y ADC2

Y los cálculos teóricos hacen que, en la simulación de este componente, se obtengan las siguiente señales de los ADC correspondientes:

Tras introducir los datos que vamos obteniendo en el microcontrolador para estas dos señales del ADC, podemos obtener la gráfica que vamos a mostrar a continuación donde se puede observar que, pese a que algún punto sí que difiere más que antes respecto al valor teórico, la mayor parte de los puntos corresponde con su valor teórico, por lo que nos encontramos ante una toma correcta de medidas.

Como podemos observar, casi todos los puntos de ADC1 coinciden con el valor que debería de tener, por lo que hemos obtenido la respuesta que esperábamos del componente. A continuación se muestra el valor del ADC2 que es en el que hemos obtenido algún dato un poco extraño, que se debe a algún error en la toma de medidas.

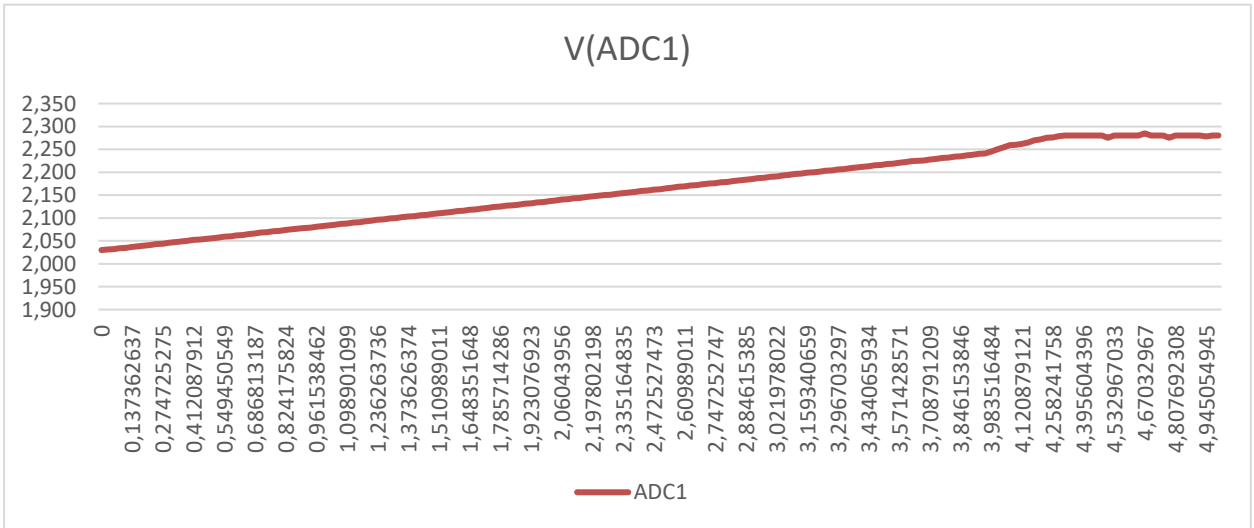

Ilustración 246 Tensión real ADC1

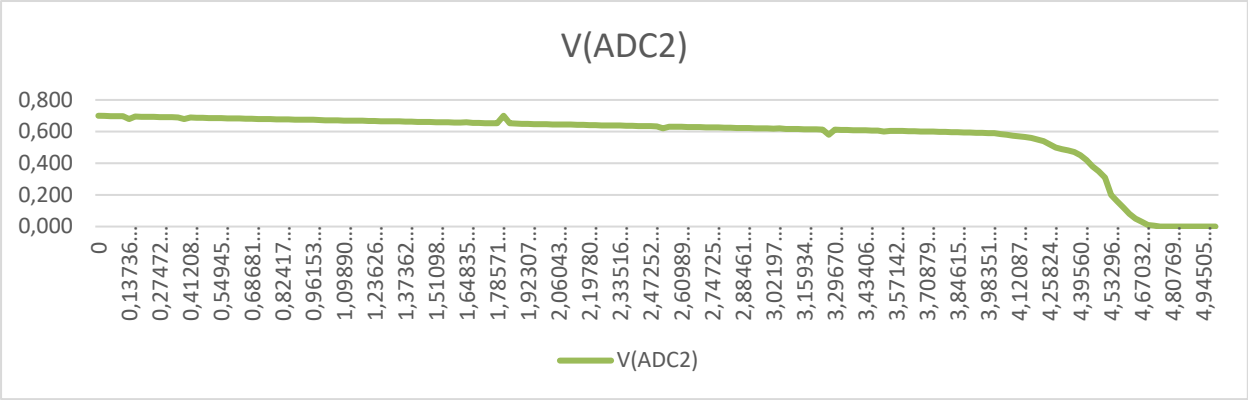

Ilustración 257 Tensión real ADC2

5.2.2 Polarización inversa.

En este punto vamos a estudiar la polarización inversa del diodo. En primer lugar se muestra el circuito utilizado para la simulación:

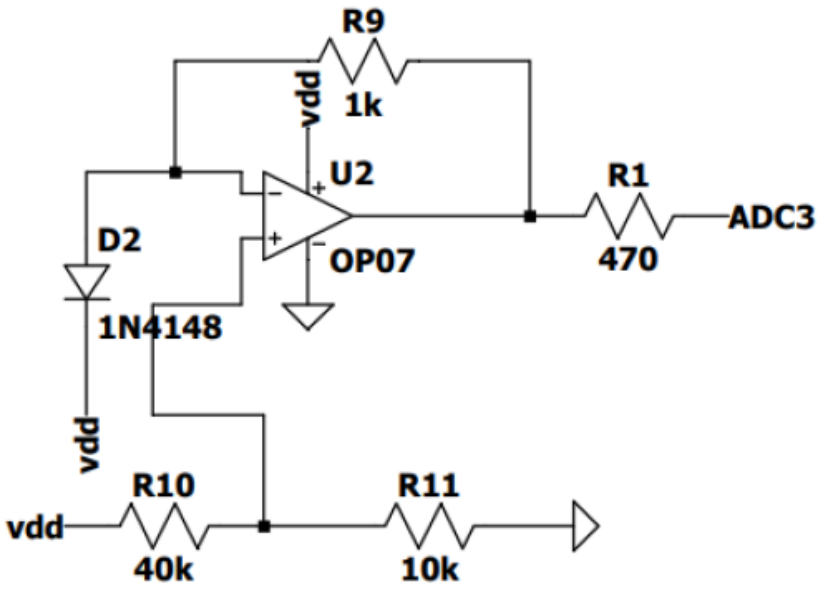

Ilustración 268 Circuito polarización inversa del diodo

Como lo que estamos buscando es asegurar que la corriente que entra en el diodo es cero, o un valor que se le asemeje, necesitamos utilizar el circuito mencionado anteriormente. Los valores obtenidos por la simulación son los siguientes:

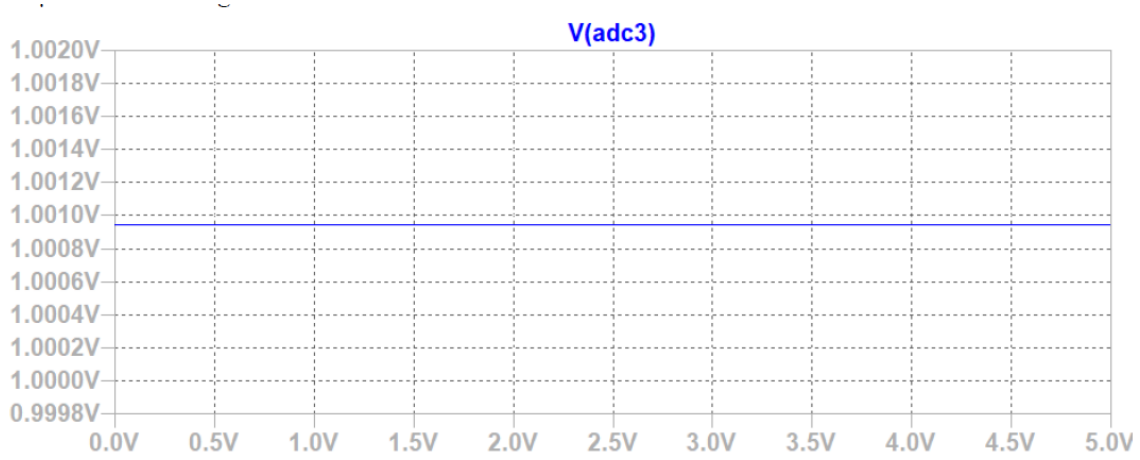

Ilustración 279 Tensión simulada ADC3

Nosotros, una vez introducido el código de programación en el microcontrolador y haber extraído los datos obtenidos por la lectura del ADC3, obtenemos la siguiente gráfica con los datos:

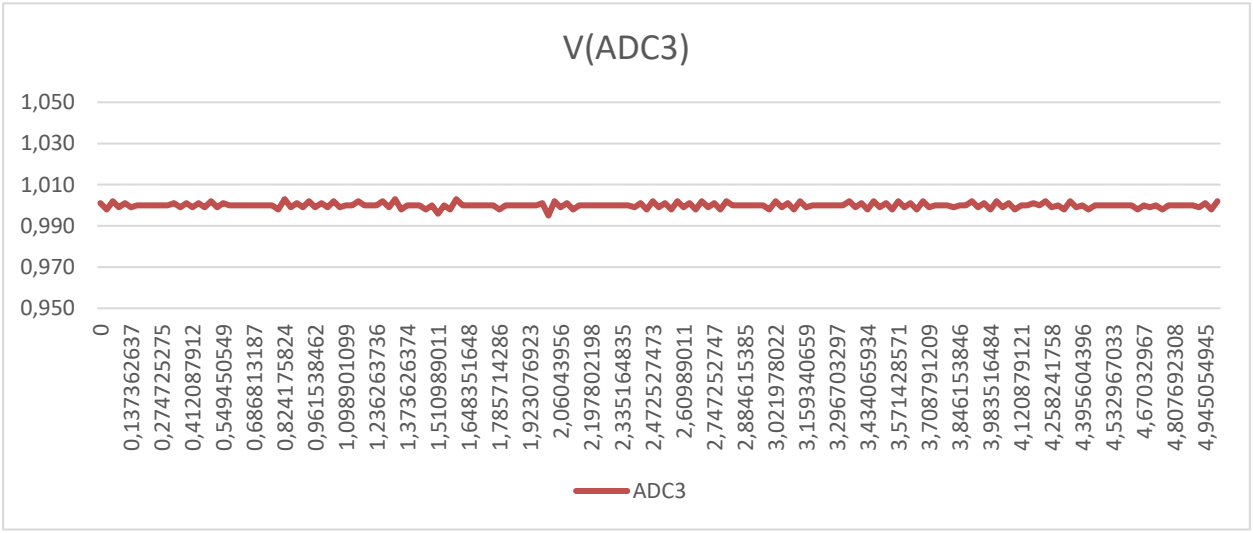

Ilustración 30 Tensión real ADC3

5.3 Transistor BJT.

Por último, tenemos el transistor BJT PNP que es el que utilizaremos para analizarlo. Igual que con los anteriores elementos, en primer lugar se va a mostrar el circuito que se ha utilizado para realizar la simulación del elemento.

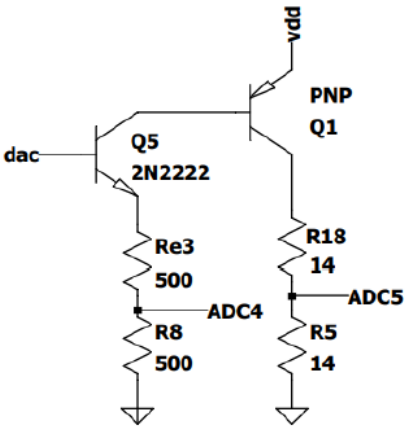

Ilustración 281 Polarización del transistor BJT

Las señales obtenidas con los canales del ADC4 y del ADC5 quedaría de la siguiente manera en la simulación realizada:

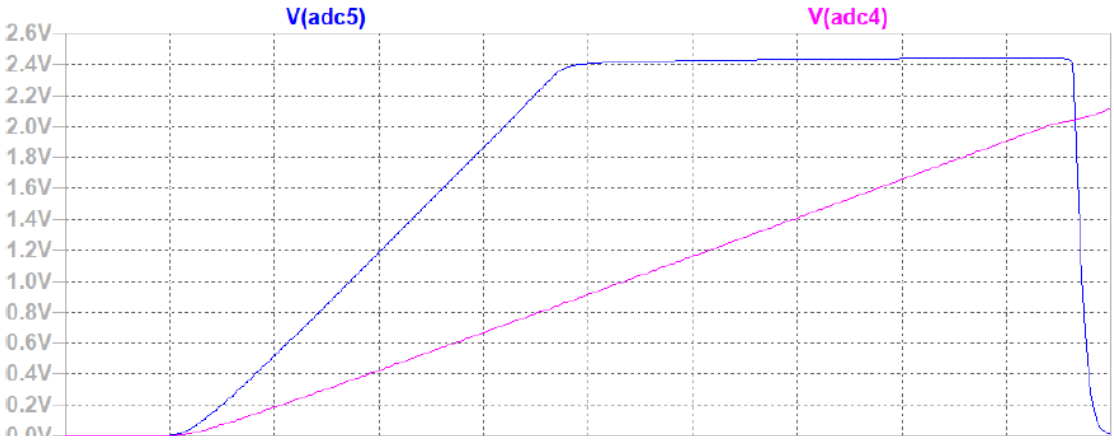

Ilustración 292 Tensión de simulación de ADC4 y ADC5

Esto se debe a que, cuando la tensión supera los 2,5V se satura el segundo transistor. A continuación se muestra la gráfica que representa la tensión en el ADC4, que podemos observar que es muy parecida a la tensión obtenida en la simulación.

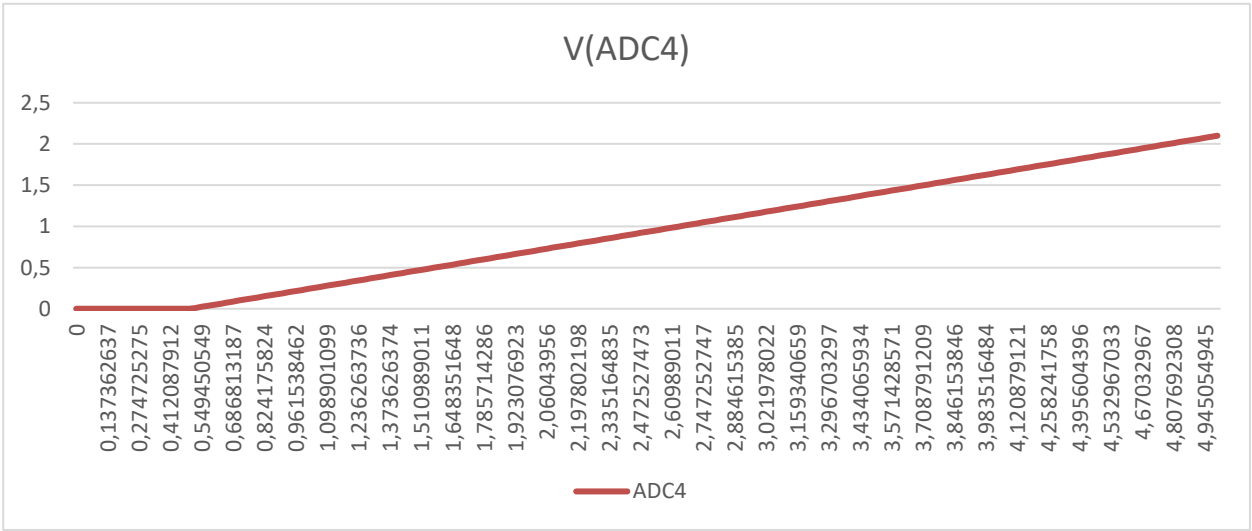

Ilustración 303 Tensión real ADC4

Por último, vamos a obtener la tensión ADC5. La obtenemos de la misma forma, con los valores que nos da el microcontrolador y los unimos para crear la siguiente gráfica:

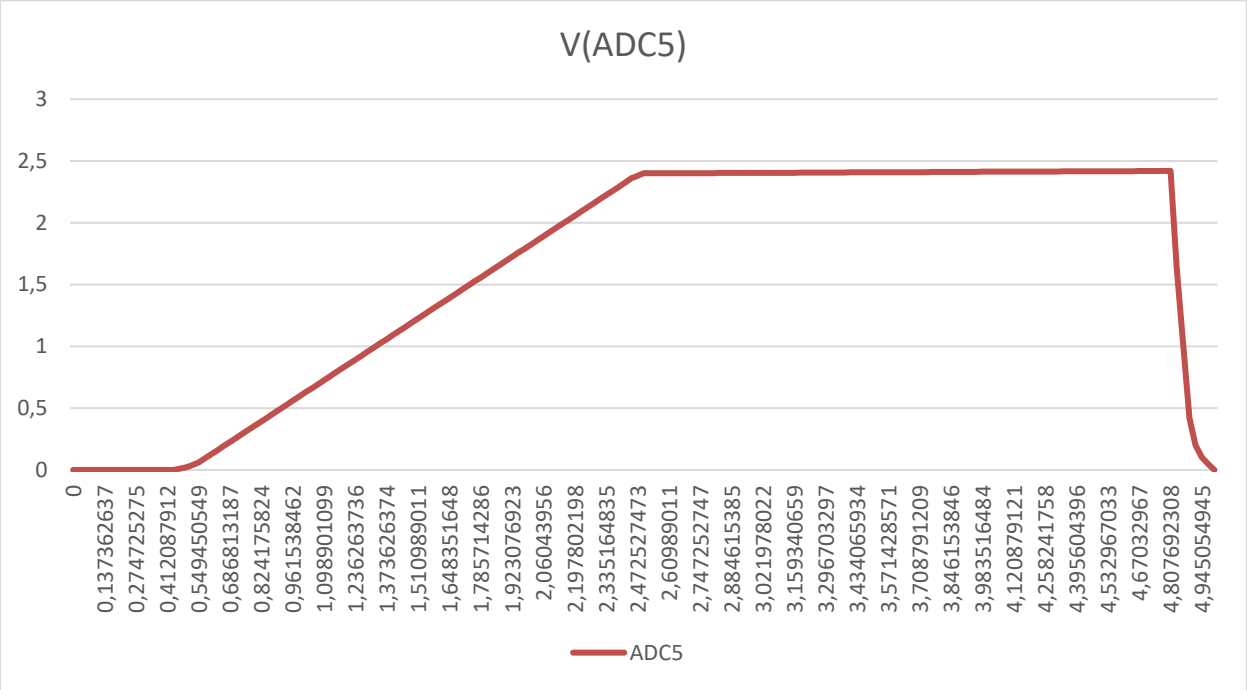

Ilustración 314 Tensión real ADC5

Podemos observar que obtenemos también valores muy similares a los obtenidos en la simulación.

5.4 Conclusiones de la obtención de medidas.

Como podemos observar en las tablas aportadas en este documento, las medidas obtenidas mediante la simulación se corresponden en gran medida con las obtenidas del ADC del microcontrolador. Por este motivo, podemos confirmar que el dispositivo construido se encuentra en disposición de comenzar a realizar las medidas a los componentes para observar la degradación.

6 CONCLUSIONES Y POSIBLES AMPLIACIONES

Este trabajo ha sido un acercamiento al sector del Cubesat, en el que se ha realizado una primera versión de un proyecto con la intención de ir ampliándolo. Pienso que el microcontrolador seleccionado, pese a tener unas características interesantes, dado que no ha sido diseñado para el espacio, no creo que se pudiera utilizar para el propósito de enviar un Cubesat al espacio.

Por ese motivo, la primera línea de investigación podría ser la de diseñar el Cubesat con un microcontrolador probado y diseñado para el espacio. Esto podría facilitar muchas cosas a la hora del diseño y de posibles problemas que se pueden encontrar en el espacio.

En segundo lugar, se le podrían añadir algunos puntos a la programación. Se podría hacer una autocalibración en tiempo real de los componentes del Cubesat con la idea de conseguir estudiar el error debido a la radiación en los componentes de estudio. Por otro lado, pienso que se podría conseguir guardar un histórico de los datos obtenidos para poder ir viendo el deterioro que produce la radiación.

Por otro lado, como posible ampliación se podrían realizar una serie de test y pruebas de resistencia para comprobar si el Hardware es capaz de mantenerse en órbita sin sufrir desperfectos.

Creo que este trabajo puede ayudar al acercamiento desde cualquier Universidad al sector espacial, sabiendo que las facilidades del Cubesat son principalmente el poco coste y el poco tiempo de dedicación y de diseño en comparación con otro tipo de satélites. Por este motivo, animo a todo el que esté interesado en investigar sobre este tema a adentrarse en la filosofía del Cubesat.

REFERENCIAS

[1] «Cubesat,» [En línea]. Available: www.cubesat.org/about.

[2] A. space. [En línea]. Available: https://alen.space/es/guia-basica-nanosatelites/. [Último acceso: 15 06 2022].

[3] «Nanosatélites,» [En línea]. Available: https://www.xataka.com/n/nanosatelites-la-democratizacion-low-cost-de-la-conquista-espacial.

[4] B. Twiggs, Origin of Cubesat, 2008.

[5] J. P.-S. A. M. S. N. R. T. H. Heidt, «CubeSat: A New Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation».

[6] A. Poghosyan, «CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions,» *Progress in Aerospace Sciences,* vol. 88, pp. 59-83, January 2017.

[7] W. A. C. T. J. Puig-Suari, «Development of the standard CubeSat deployer and a CubeSat class PicoSatellite,» *2001 IEEE Aerospace Conference Proceedings,* vol. 1, pp. 1/347-1/353, 2001.

[8] T. Diallo, *Aplicaciones de procesado digital de señal sobre plataforma NUCLEO-L432KC de bajo consumo,* 2019.

[9] California Polytechnic State University, «CubeSat Design Specification, 13th ed. California, 2014.».

[10] B. A. Ruiz-Morón, «Adquisición de datos con microcontroladores para aplicaciones en el internet de las cosas,» 2017.

[11] A. C. Pimentel, «A benchmark for assessing nanosatellite on-board computer power-efficiency and performance,» 2020.

[12] W. K. Hui-fu, «Design of the Data Acquisition System Based on STM32,» *Procedia Computer Science,* vol. 17, pp. 222-228, 2013.

[13] C. D. K. T. M. K. P. S. J. Mankar, «REVIEW OF I2C PROTOCOL,» *International Journal of Research in Advent Technology,* vol. 2, January 2014.

[14] Z. Hu, «I2C Protocol Design for Reusability,» *Third International Symposium on Information Processing,* pp. 83-86, 2010.

[15] A. P. T. Ms. Neha, «A Review on Serial Communication by UART,» *International Journal of Advanced Research in,* vol. 3, January 2013.

[16] A. E. H. A. H. D. T. Y. A.-N. a. M. S. A. N. Saeed, «CubeSat Communications: Recent Advances and Future Challenges,» *IEEE Communications Surveys & Tutorials,* vol. 22, nº 3, pp. 1839-1862, 2020.

[17] J. A. L. a. J. D. Nielsen, «Development of cubesats in an educational context,» *Proceedings of 5th International Conference on Recent Advances in Space Technologies,* pp. 777-782, 2011.

[18] C. N.-P. a. M. R. Emami, «CubeSat mission: From design to operation,» *Applied Sciences (Switzerland),* vol. 9, nº 15, 2019.

[19] D. B. Ibort, «Prototipo de un nanosatélite bajo el estándar CubeSat».

[20] S. D. S. P. C. Z. J. A. F. a. B. B. J. E. Herrara, «Proceso de diseño de una estructura nanosatélital CubeSat,» *XI Congreso Internacional sobre Innovación y Desarrollo Tecnológico,* nº 67, 2015.

[21] G. D. Martino, «Software y protocolos para Cubesat,» *Tesis de grado,* 2013.

[22] D. G. Díez, «Design of the electric power system of a Cubesat educational kit,» *Trabajo final de grado,* 2019.

[23] J. B. a. A. P. S. van der Linden, «Design and Validation of an Innovative Data Bus Architecture for CubeSats,» *Proceedings of the Reinventing Space Conference,* pp. 1-13, 2016.

ANEXO A: CÓDIGO DE PROGRAMACIÓN

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

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

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

/* USER CODE END PTD */

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

/*       cambiar_valor_potenciometro(potenciometro);
         leer_ADC_0;
         leer_ADC_1;
         leer_ADC_2;
         leer_ADC_3;
         leer_ADC_4;
         leer_ADC_5;
         enviar(valor_adc_0);
*/

/* USER CODE END PD */

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

/* USER CODE END PM */

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

I2C_HandleTypeDef hi2c1;
```

```
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

void lectura_adc(int);
void ADC_Select_CH0 (void);
void ADC_Select_CH1 (void);
void ADC_Select_CH2 (void);
void ADC_Select_CH3 (void);
void ADC_Select_CH4 (void);
void ADC_Select_CH5 (void);

void subir_valor_potenciometro(void);
void cero_valor_potenciometro(void);

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
       float potenciometro=0;

  /* USER CODE END 1 */

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

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

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

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

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_ADC1_Init();
```

```
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */

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

    /* USER CODE BEGIN 3 */
        if (potenciometro<182)  //256 son 7V. 182 son 5V
        {
              potenciometro=potenciometro+1;
              subir_valor_potenciometro();  /*Función cambiar_valor_potenciometro*/
        }
        if (potenciometro==182)
        {
              potenciometro=0;
              cero_valor_potenciometro();   /*Función cambiar_valor_potenciometro*/
        }

        lectura_adc (0);
        lectura_adc (1);
        lectura_adc (2);
        lectura_adc (3);
        lectura_adc (4);
        lectura_adc (5);
  }
  /* USER CODE END 3 */
}

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

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
```

```
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 16;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Enable MSI Auto calibration
  */
  HAL_RCCEx_EnableMSIPLLMode();
}

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

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc1.Init.LowPowerAutoWait = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
```

```
hadc1.Init.NbrOfConversion = 6;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc1.Init.OversamplingMode = DISABLE;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_11;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_10;
sConfig.Rank = ADC_REGULAR_RANK_2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_9;
sConfig.Rank = ADC_REGULAR_RANK_3;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_8;
sConfig.Rank = ADC_REGULAR_RANK_4;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = ADC_REGULAR_RANK_5;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}
```

```
  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_15;
  sConfig.Rank = ADC_REGULAR_RANK_6;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00707CBB;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */
```

```c
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

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

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

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : LD3_Pin */
  GPIO_InitStruct.Pin = LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
```

```
  HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

void ADC_Select_CH0 (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	  /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
	  */
	  sConfig.Channel = ADC_CHANNEL_11;
	  sConfig.Rank = ADC_REGULAR_RANK_1;
	  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
	  sConfig.SingleDiff = ADC_SINGLE_ENDED;
	  sConfig.OffsetNumber = ADC_OFFSET_NONE;
	  sConfig.Offset = 0;

	  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	  {
	    Error_Handler();
	  }
}
void ADC_Select_CH1 (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	  /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
	  */
	  sConfig.Channel = ADC_CHANNEL_10;
	  sConfig.Rank = ADC_REGULAR_RANK_1;
	  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
	  sConfig.SingleDiff = ADC_SINGLE_ENDED;
	  sConfig.OffsetNumber = ADC_OFFSET_NONE;
	  sConfig.Offset = 0;
	  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	  {
	    Error_Handler();
	  }
}
void ADC_Select_CH2 (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	  /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
	  */
	  sConfig.Channel = ADC_CHANNEL_9;
	  sConfig.Rank = ADC_REGULAR_RANK_1;
	  sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
	  sConfig.SingleDiff = ADC_SINGLE_ENDED;
	  sConfig.OffsetNumber = ADC_OFFSET_NONE;
	  sConfig.Offset = 0;
	  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	  {
	    Error_Handler();
	  }
}
void ADC_Select_CH3 (void)
{
```

```
      ADC_ChannelConfTypeDef sConfig = {0};
        /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
        */
        sConfig.Channel = ADC_CHANNEL_8;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
        sConfig.SingleDiff = ADC_SINGLE_ENDED;
        sConfig.OffsetNumber = ADC_OFFSET_NONE;
        sConfig.Offset = 0;
        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
}
void ADC_Select_CH4 (void)
{
      ADC_ChannelConfTypeDef sConfig = {0};
        /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
        */
        sConfig.Channel = ADC_CHANNEL_7;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
        sConfig.SingleDiff = ADC_SINGLE_ENDED;
        sConfig.OffsetNumber = ADC_OFFSET_NONE;
        sConfig.Offset = 0;
        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
}
void ADC_Select_CH5 (void)
{
      ADC_ChannelConfTypeDef sConfig = {0};
        /** Configure for the selected ADC regular channel its corresponding rank in
the sequencer and its sample time.
        */
        sConfig.Channel = ADC_CHANNEL_15;
        sConfig.Rank = ADC_REGULAR_RANK_1;
        sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
        sConfig.SingleDiff = ADC_SINGLE_ENDED;
        sConfig.OffsetNumber = ADC_OFFSET_NONE;
        sConfig.Offset = 0;
        if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
        {
          Error_Handler();
        }
}

/*************************************************************************************
**************/
/*      leer_ADC;*/
void lectura_adc(int i)
      {
    uint32_t ADC_VAL=0;
      //uint8_t valor[40];
      char valor[40];
    float voltaje;
      int conve;
```

```
	if (i==0){
		ADC_Select_CH0();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	}
	if (i==1){
		ADC_Select_CH1();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	      }
	if (i==2){
		ADC_Select_CH2();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	      }
	if (i==3){
		ADC_Select_CH3();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	      }
	if (i==4){
		ADC_Select_CH4();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	      }
	if (i==5){
		ADC_Select_CH5();
		HAL_ADC_Start(&hadc1);
		HAL_ADC_PollForConversion(&hadc1, 1000);
		ADC_VAL = HAL_ADC_GetValue(&hadc1);
		HAL_ADC_Stop(&hadc1);
	      }

	    voltaje= (float) ADC_VAL/4096;
	    voltaje= (float) voltaje*3300;
	    conve= (unsigned) voltaje;
	    sprintf (valor,"%u",conve);

	HAL_UART_Transmit(&huart2, valor, sizeof (valor), 10);
	HAL_UART_Transmit(&huart2,ADC_VAL, sizeof (ADC_VAL),10);
	}

/*****************************************************************************
**************/
/*Cambiar potenciometro*/

void subir_valor_potenciometro(void)
{
	uint16_t dir_pot = 44; /*44 es 0101100 en decimal*/
```

```
	uint8_t mensaje0= 66; /*66 es 1001010 en decimal*/
	uint8_t mensaje1= 74; /*74 es 1000010 en decimal*/

	HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje0,sizeof(mensaje0),100);
/*Cambiar valor fuente 0*/
	HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje1,sizeof(mensaje1),100);
/*Cambiar valor fuente 1*/
}

void cero_valor_potenciometro(void)
{
	int i;
	uint16_t dir_pot = 44; /*44 es 0101100 en decimal*/
	uint8_t mensaje0= 68; /*68 es 1000100 en decimal*/
	uint8_t mensaje1= 76; /*76 es 1001100 en decimal*/
	for(i=0;i<182;i++)
	{
		HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje0,sizeof(mensaje0),100);
/*Cambiar valor fuente 0*/
		HAL_I2C_Master_Transmit(&hi2c1,dir_pot,&mensaje1,sizeof(mensaje1),100);
/*Cambiar valor fuente 1*/
	}

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

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