UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TRABAJO DE FINAL DE CARRERA

TÍTULO DEL TFG: Actualización del firmware del MSP430 mediante el Bootloader

TITULACIÓN: Grado en Ingeniería de Sistemas Aeroespaciales

AUTOR: Cristian Rogel Sanchez

DIRECTOR: Oscar López Lapeña

FECHA: 3 de septiembre de 2019

Título: Actualización del firmware del MSP430 mediante el Bootloader
Autor: Cristian Rogel Sanchez

Director: Oscar López Lapeña

Fecha: 3 de septiembre de 2019

Resumen

A día de hoy, la gran mayoría de productos que podemos encontrar en el mercado están basados en dispositivos que tienen la capacidad de ser reprogramados según las necesidades, siendo el firmware un factor a tener en cuenta en el diseño de un sistema. Es por ello, que no solo el firmware podrá ser programado en producción por el fabricante sino que el usuario deberá de ser capaz de actualizar el software de dicho dispositivo ya sea para resolver posibles problemas o bien para añadir o mejorar prestaciones. Para ello, se necesitará de un programa residente en la memoria del dispositivo, conocido como Bootloader, el cual se ejecuta cuando sea necesario actualizar el firmware. Este programa recibe a través de un cierto protocolo de comunicaciones los datos sobre el nuevo firmware y lo deberá de interpretar y grabar en la memoria de programa del dispositivo.

En este proyecto, se estudiará el uso del Bootloader que incorpora ya de fábrica el microcontrolador MSP430FR6989 para poder realizar las actualizaciones a través de conexión USB o Bluetooth. Para ello, se ha diseñado el hardware necesario escogiendo los dispositivos más convenientes que permitan establecer conexión entre el microcontrolador y un ordenador personal, y gracias a LabVIEW se ha creado una aplicación que sirve como interfaz de usuario y de control del Bootloader.

Title : MSP430 firmware update via Bootloader
Author: Cristian Rogel Sanchez

Advisor: Oscar López Lapeña

Date: September 3, 2019

Overview

Nowadays, most of the products that we can find in the market are based on devices that can be reprogrammed, being firmware a major factor to consider when designing a system. It means that the firmware will not be only programmed by the manufacturer during production as the user will also be able to update it in order to solve possible issues or improve the device's performance. To do this, a program resident in the device's memory, known as Bootloader, will be executed, allowing the firmware getting updated. This program receives the firmware data through an specific communications protocol and must interpret and write the data in the program memory of the device.

In this final degree project, is going to be explained how the Bootloader works and how it is going to be used in the MSP430FR6989 in order to be able to update the firmware via USB or Bluetooth.For this, the hardware has been designed by choosing the most convenient devices that allow establishing communication between the microcontroller and a personal computer, and by using LabVIEW it has been created a user interface which allows to control the Bootloader.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

CAPÍTULO 1. INTRODUCCIÓN

A día de hoy, el firmware ha tomado prácticamente la misma importancia que el hardware a la hora de desarrollar un sistema. Es por ello, que la gran mayoría de dispositivos que encontramos en el mercado tienen la capacidad de ser reprogramados, es decir, que no solamente podrá ser programado durante la fase de fabricación sino que el usuario también tendrá dicha posibilidad, pudiendo actualizar el firmware del dispositivo para así añadir nuevas prestaciones o corregir errores existentes. Esto será posible gracias al Bootloader, un programa ubicado en la memoria del microcontrolador del dispositivo que es capaz de recibir a través del protocolo UART los datos relacionados con el nuevo firmware, para interpretarlos y grabarlos en la memoria del dispositivo. Cada vez que el usuario quiera realizar una actualización, es importante que se le garantice la integridad y protección de los datos escritos. Para ello, se debe de informar al usuario sobre si se ha realizado correctamente la actualización, además de garantizar que los datos transmitidos han sido protegidos ante posibles plagios o amenazas.

A día de hoy, ya existen aplicaciones como BSL Scripter de Texas Instruments que permiten actualizar el firmware de un microcontrolador. Esta aplicación funciona gracias a un dispositivo que actúa como intermediario entre el ordenador personal y el microcontrolador, que se encarga de acceder al Bootloader y de transmitir los datos del nuevo firmware, convirtiéndolos previamente en el formato adecuado para su interpretación. Desafortunadamente, esta aplicación está pensada para funcionar solo con algunos de estos dispositivos puente que casualmente pertenecen a la misma compañía. Estos dispositivos son el MSP-FET y el BSL-Rocket, que presentan un principal inconveniente, su elevado precio en comparación a otros dispositivos. Tener un precio alto puede resultar un problema para el usuario que desee actualizar su dispositivo. Sin embargo, haciendo uso de módulos de conexión serie como el FT232R o de conexión Bluetooth como el módulo RN41, el impacto económico ya sería mucho menor. Principalmente por esta razón hemos encontrado necesaria la creación de una nueva aplicación que permita usar estos dispositivos de comunicación más baratos o similares, sin por ello tener que perder las principales funcionalidades que la aplicación BSL Scripter ofrece al usuario.

El objetivo principal de este proyecto de final de carrera es el de crear una aplicación capaz de acceder al Bootloader de un microcontrolador, en concreto el MSP430FR6989, a través de dispositivos de comunicación, los cuales ofrecerán conexión serie o Bluetooth. La aplicación permitirá la comunicación entre un ordenador y el microcontrolador para así poder realizar actualizaciones de firmware, para configurar parametros internos relacionados con la comunicación o para ejecutar ciertas órdenes que incorpora el Bootloader. Dicha interfaz debe de ser fácil e intuitiva, permitiendo un uso sencillo y eficaz de cara al usuario.

Crear una interfaz propia nos permitirá configurar y diseñar infinidad de funcionalidades. Por ejemplo, podremos acceder al Bootloader de la placa a través de los componentes que creamos más adecuados, crear una aplicación que actualice el firmware a través de la interpretación de un texto, limpiar la memoria del microcontrolador y mucho más.

Para ello haremos uso de LabVIEW, una aplicación que hace uso de un lenguaje de programación gráfico orientado al control de sistemas y que permite realizar pruebas a tiempo real o bien simulaciones. Con este programa se creará la aplicación a la que hemos llama-

do Bootloader Scripter que permitirá acceder al Bootloader y realizar las comunicaciones entre el usuario y el microcontrolador. Para desarrollar dicho programa, se ha hecho uso de una gran variedad de documentación y se han realizado las investigaciones necesarias a través de Internet.

Sin tener en cuenta esta introducción, la memoria se encuentra dividida en tres capítulos. En el primero de ellos se introducirán los diferentes métodos usados a día de hoy para actualizar el firmware de un dispositivo y se detallarán los motivos por los que se ha escogido el Bootloader como método principal. Después, se procederá a explicar la arquitectura que tiene el Bootloader que incluye el MSP430FR6989 junto a sus principales características. En el segundo, se seleccionará el hardware adecuado y se diseñará el sistema que permita realizar las comunicaciones con el microcontrolador. Finalmente, en el último capítulo se observará como ha sido creado el programa de LabVIEW y se explicarán las principales funcionalidades que implementa y como estas han sido creadas. A parte de estos capítulos, se incluirá un apartado de conclusiones donde se podrán detallar las observaciones que se han realizado tras finalizar el proyecto.

CAPÍTULO 2. ESTADO DEL ARTE

2.1. Métodos para la actualización del firmware en microcontroladores

A día de hoy, la mayoría de fabricantes ofrecen al usuario dos soluciones para la interacción con el microcontrolador: Hacer uso de un dispositivo depurador y programador en circuito y el Bootloader.

El dispositivo depurador y programador en circuito es un sistema externo al microcontrolador que permite al usuario la depuración del diseño del software en el propio circuito. Gracias a esto, se puede programar y grabar firmware en el microcontrolador creando paradas que permitan monitorizar el rendimiento de dicho firmware con el resto del sistema, pudiendo también monitorizar el consumo en tiempo real. Dichos dispositivos pueden ser más o menos complejos en función de las prestaciones de las que disponen. Existen soluciones complejas como el MSP-FET de la casa *Texas Instrument*, el cual proporciona mayor capacidad de puntos parada y una frecuencia de muestreo alta a la hora de monitorizar el rendimiento. También se disponen de soluciones más sencillas y baratas, como el eZ-FET que incorpora también *T.I.* en sus placas de desarrollo, que permiten realizar lo mismo que el MSP-ET tan solo que en menor medida.

En cambio, el Bootloader o gestor de arranque es un pequeño programa que suele estar incluido en la memoria ROM del microprocesador. Este programa permite modificar o grabar datos en memoria haciendo uso de comandos a través de un puerto de comunicaciones. Para acceder a él será necesario ejecutar una cierta secuencia de invocación en unos puertos ubicados en el microcontrolador cada vez que este sea iniciado. El Bootloader está únicamente orientado a la actualización del firmware de dispositivos por parte del usuario con el mínimo coste, al no disponer de las prestaciones necesarias para monitorizar. El fragmento de código que forma el Bootloader puede encontrarse ya programado en la memoria ROM del microcontrolador o bien puede ser precargado en una parte de la memoria de programa por el diseñador del sistema. A pesar de que dicho programa puede ser complejo, llegando a incluir algoritmos de encriptación y autentificación de datos, el hardware externo necesario para su invocación es de lo más sencillo ya que tan solo debe generar la secuencia de invocación y asegurar la comunicación serie, lo que hace que implementar un sistema que haga uso del Bootloader no añada un coste significativo al producto final.

A continuación, se podrá ver una tabla (2.1) donde se podrán comparar algunos de estos sistemas que ofrecen los principales fabricantes.

Dispositivo	Fabricante	Tipo	Características	Precio
eZ-FET	Texas Instruments	In-Circuit Debugger	Incluido en placas de desarrollo. Programar, depurar y monitorizar	min. 9,25€ (según la placa)
MSP430-BSL	Olimex & Texas Instruments	Bootloader	Programar. Conexión USB. Configurable. Pequeñas dimensiones.	15€ aprox.
MSP-FET	Texas Instruments Bootloader	In-Circuit Debugger /	Programar, depurar y monitorizar con gran rendimiento	120€ aprox.
FT232R	FTDI	Bootloader	Programar. Conexión USB. Configurable	5€ aprox.
PICkit3	Microchip Technology	In-Circuit Debugger	Programar, depurar y monitorizar con mayor rendimiento. Bajo precio para In-Circuit Debugger	45€ aprox.
ATMEL-ICE	Atmel Technology	In-Circuit Debugger	Programar, depurar y monitorizar con gran rendimiento	125€ aprox.
RN41XVC-I/RM	Roving Networks	Bootloader	Programar. Conexión Bluetooth. Configurable	25€ aprox.

Cuadro 2.1: Tabla comparativa de sistemas para la actualización del firmware

Teniendo en cuenta el precio de ellos y que la principal razón para hacer uso de ellos es para la actualización del firmware, el uso de un dispositivo que emplee el Bootloader para programar en vez de un dispositivo depurador saldrá mucho mas a cuenta.

2.2. Arquitectura del Bootloader del MSP430FR6989

En los dispositivos MSP430, el Bootloader permite al usuario comunicarse e intercambiar información con el microcontrolador durante las fases de prototipo, de producción y de servicio, permitiendo modificar la memoria programable (FRAM) y la memoria de datos (RAM) según se requiera. Para poder acceder al Bootloader, será necesaria una secuencia de entrada en unos determinados pines del microcontrolador. Una vez dentro del Bootloader, ya se podrá proceder a intercambiar datos a través de diferentes comandos incluidos en el código del Bootloader, que permiten ejecutar diferentes funciones según las necesidades del usuario, como por ejemplo, sobreescribir en la memoria o bien limpiarla.

A continuación, se detallará la estructura y los parámetros más importantes que definen al Bootloader del MSP430FR6989. Para más información acerca de la arquitectura del Bootloader en dicho microcontrolador, ver el documento [1].

2.2.1. Invocación del Bootloader

Nada más iniciar el microcontrolador, el Bootloader será iniciado cuando se detecte que se ha ejecutado una cierta secuencia en los puertos TEST y RESET del microcontrolador. Este método de acceso al Bootloader es conocido como invocación por hardware. Según si ha sido realizada o no la secuencia, se procederá a ejecutar el programa ya guardado en memoria o bien se accederá al modo Bootloader. También es posible acceder por software al Bootloader desde una aplicación que sea capaz de saltar a la dirección de inicio del Bootloader. Por úlitmo, existe la posibilidad de acceder en ciertos dispositivos a través de un vector de reseteo, según si este se encuentra vacío o no.

En este proyecto se accederá a través de invocación por hardware. En las figuras 2.1 y 2.2 se podrán observar las posibles secuencias que se ejecutan normalmente en los puertos TEST y RESET:

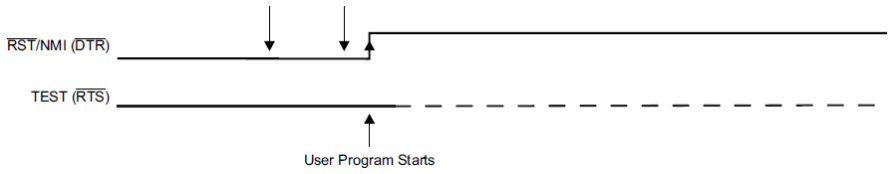

Figura 2.1: Secuencia RESET estándard

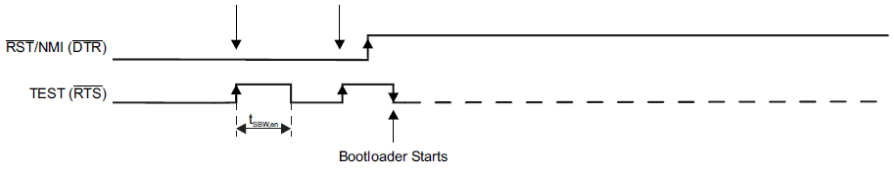

Figura 2.2: Secuencia de acceso al Bootloader

La primera secuencia es la secuencia de por defecto, que fuerza al microcontrolador a ejecutar el programa que tenga descargado en memoria. Mientras que la segunda secuencia es la que fuerza al microcontrolador a iniciar en modo Bootloader, variando las señales de ambos puertos siguiendo un patrón.

Una vez dentro del Bootloader, ya se podrá proceder a enviar los comandos necesarios para que se cumpla la función deseada (Programar el dispositivo, desbloquearlo o configurar la velocidad de transmisión de datos, por ejemplo). Finalmente, para salir del Bootloader basta con indicar la dirección de arranque donde queremos que inicie o bien reiniciar el dispositivo.

2.2.2. Protocolos de comunicación

En primer lugar, el Bootloader ofrece dos protocolos de comunicación que son la UART y la I2C. Para este proyecto, se hará uso de comunicación UART, ya que es el protocolo implementado tanto en la placa como en los componentes usados y además es el protocolo que resulta más sencillo de usar. Dicho protocolo permite la comunicación tanto serie

como inalámbrica a través de los puertos RX y TX (Ubicados en cada dispositivo) y carece de la necesidad de una señal de reloj, ya que la sincronización se efectúa mediante un bit de arranque el cual precede a cada byte.

Para poder comunicar dispositivos a través del protocolo UART, la configuración de estos debe de ser idéntica. Esta configuración incluye los siguientes parámetros: velocidad de transmisión (*Baud rate*), número de bits de datos (*Data bits*), paridad (par, impar o ninguna), número de bits de parada (*Stop bits*) y control de flujo (*Flow control*). Estos parámetros y algunos otros suelen estar configurados en el microcontrolador de inicio de la siguiente manera: *Baud rate* de 9600 bps, trabaja en modo semidúplex (envío de información bidireccional pero no simultánea), el establecimiento de comunicación (*Handshake*) entre emisor y receptor viene definido por un carácter de reconocimiento y se debe de cumplir un tiempo mínimo de retraso de 1.2 ms antes de realizar un nuevo envío de bits después haber sido recibidos. Además, los paquetes recibidos o enviados están compuestos por 1 bit de arranque, 8 bits de datos, 1 bit de paridad par y 1 bit de parada.

A la hora de enviar los datos, es necesario cumplir con un formato y un orden determinado, formando así paquetes que el Bootloader sea capaz de procesar e interpretar. Cada paquete de datos transmitido estará formado por el código PI (*Peripheral Interface*), que dependerá del protocolo UART y que permite la correcta transmisión de los datos entre dispositivos. También está formado por el comando central, si el paquete es de transmisión, o bien por la respuesta central si es de recepción. El comando o respuesta central es el fragmento del paquete que contiene toda la información que deberá de ser procesada por el Bootloader para ejecutar un comando o que contendrá la respuesta del microprocesador.

Figura 2.3: Estructura de un paquete de datos

2.2.2.1. Paquete enviado al Bootloader

El paquete enviado al Bootloader contiene los datos necesarios para mandar los bytes al microcontrolador y ejecutar una cierta orden o comando. Como bien ya se ha comentado, estará formado por el código PI y el comando central. El código PI del UART estará formado por los siguientes datos: un byte con la información relevante sobre la cabecera (Header), que es la que contiene la información necesaria para que un paquete pueda ir del emisor al receptor, también los bytes de la longitud del paquete (NL y NL) y finalmente los bytes de la suma de verificación o *Checksum* (CKL y CKH), que sirven para comprobar que los datos transmitidos y los grabados en la memoria sean los mismos. Esto se realiza a través de una función *hash*, la cual se usa para representar cadenas de datos de manera compacta. Por lo tanto, el emisor envía los datos junto al valor *hash* calculado al receptor. Una vez recibidos por el receptor, este puede volver a calcular el valor *hash* por su cuenta con los datos recibidos y compararlo con el *hash* procedente del emisor, verificando de esta forma que los datos coincidan (Esto no significa que el comando se haya ejecutado correctamente). En caso de discrepancia, los datos son rechazados o bien retransmitidos.

Header	Length	Length	BSL Core Command	CKL	CKH
0x80	NL	NH		CKL	CKH

Figura 2.4: Código de la UART en el paquete de datos enviados

A parte del código de la UART, este paquete de envío está formado también por el *Core Command* (comando central), que contiene todos los bytes relacionados con el comando que se quiere ejecutar. Dicho comando puede contener órdenes para enviar un nuevo firmware al microcontrolador, como podría ser una nueva actualización de firmware, o bien órdenes para ejecutar un comando en concreto como podría ser el de limpiar la memoria. A continuación, se adjuntará una tabla (2.2) que contiene los diferentes comandos disponibles, el byte de comando perteneciente a cada uno (CMD), si es necesario o no escribir dirección de memoria la cual esta dividida en 3 bytes: Byte de dirección de memoria alto (AH), byte de medio (AM) y byte bajo (AL), los datos que debe de contener y si recibirá o no respuesta.

Comando	**Protegido**	**CMD**	**AL**	**AM**	**AH**	**Data**	**Resp.**
RX Data Block	Sí	0x10	(AL)	(AM)	(AH)	D1...Dn	Sí
RX Password	No	0x11	-	-	-	D1...D32	Sí
Mass Erase	No	0x15	-	-	-	-	No
CRC Check	Sí	0x16	(AL)	(AM)	(AH)	Longitud (byte bajo y alto)	Sí
Load PC	Sí	0x17	(AL)	(AM)	(AH)	-	No
TX Data Block	Sí	0x18	(AL)	(AM)	(AH)	Longitud (byte bajo y alto)	Sí
TX BSL Version	Sí	0x19	-	-	-	-	Sí
RX Data Block Fast	Sí	0x1B	(AL)	(AM)	(AH)	D1...Dn	No
Change Baud Rate	No	0x52	-	-	-	-	No

Cuadro 2.2: Tabla de comandos del Bootloader

Como se puede observar, hay ciertos comandos que están protegidos por contraseña, es decir, que sin haber introducido previamente la contraseña correcta a través del comando *RX Password* estos comandos no podrán ser ejecutados. En caso de que la contraseña no coincida, el Bootloader ejecutará la orden *Mass Erase*, eliminando así todos los datos de la memoria FRAM del microcontrolador. Además de lo mencionado, cada comando tiene asociado su propio byte de comando, para así distinguirse del resto y pueden tener o no respuesta por parte del microcontrolador.

2.2.2.2. Paquete de respuesta del Bootloader

El paquete de respuesta es transmitido por el microcontrolador en respuesta al comando ejecutado. Contiene exactamente lo mismo que el paquete de envío, excepto por el bit de reconocimiento (*Acknowledgment* o ACK) y el *Core Response* (Respuesta del Bootloader). El bit de *Acknowledgment* informa de si el paquete ha sido recibido correctamente o no, y en caso de error nos da la razón de este. Mientras que el *Core Response*, en caso

de haber sido recibido correctamente, nos indica si el paquete se ha interpretado debidamente y puede incluir bytes relacionados con la respuesta al comando. El *Core Response* esta dividido en dos partes: El CMD byte y el byte de mensaje. Si el microcontrolador nos da una respuesta correcta, el byte del CMD será siempre 3B, mientras que el byte o los bytes de mensaje variarán en función de cómo el Bootloader ha interpretado los datos y también del comando ejecutado. A pesar de lo anteriormente comentado, hay ciertos comandos que no darán respuesta al usuario. Justo debajo hay adjuntada una tabla 2.3 con las posibles respuestas por parte del Bootloader, en caso de contener un solo byte de mensaje como respuesta.

Acknowledgment	Header	Length	Length	BSL Core Response[1]	CKL	CKH
ACK from BSL	0x80	NL	NH		CKL	CKH

(1) BSL Core Response is not always included.

Figura 2.5: Código de la UART en el paquete de datos recibidos

CMD Byte	BSL Core Message
0x3B	Message

Figura 2.6: Estructura de *Core Response*

Mensaje	Significado
0x00	Operación realizada con éxito.
0x01	Fallo durante la escritura, tras realizar comprobación del CRC.
0x04	Bootloader bloqueado. Falta la contraseña.
0x05	Contraseña errónea
0x07	Comando no identificado

Cuadro 2.3: Bytes de mensaje y su significado

2.2.3. Comandos de control

En este apartado, se mostrarán con más detalle los comandos que pueden ser ejecutados a través del Bootloader, explicando cómo funcionan, su estructura y cuál es la respuesta esperada por parte del Bootloader. En el apartado *4.1.5 BSL Core Commands* del documento [1], se puede encontrar una explicación más detallada para cada uno de ellos.

***RX Data Block* (CMD 10):**

Este comando se ejecuta para escribir un máximo de 256 bytes en una dirección especificada de la memoria del microcontrolador. Una vez ejecutado el comando, el usuario recibe respuesta por parte del Bootloader sobre el estado de la operación. Para ejecutarlo, se requerirá escribir la contraseña previamente. A continuación se podrá observar un ejemplo con el paquete de envio y otro con el recibido.

Header	Length	Length	CMD	AL	AM	AH	D1	D2	D3	D4	CKL	CKH
0x80	0x08	0x00	0x10	0x00	0x00	0x01	0x10	0x32	0x54	0x76	0x93	0xCA

Figura 2.7: Ejemplo de comando *RX Data Block* enviado

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

Figura 2.8: Ejemplo de comando *RX Data Block* recibido

***RX Password* (CMD 11):**

Con este comando se podrá enviar al Bootloader la contraseña para así poder acceder al resto de comandos o bien ejecutar una limpieza masiva. Esta limpieza se producirá en caso de que la contraseña no sea correcta y eliminará todo el código ubicado en la FRAM del microcontrolador. La contraseña originalmente estará formada por un vector de 32 bytes, todos ellos con valor FF (Hexadecimal), siempre y todo que la memoria se encuentre vacía.

Header	Length	Length	CMD	D1	D2	D3	D4	D5	D6
0x80	0x21	0x00	0x11	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

D7	D8	D9	D10	D11	D12	D13	D14	D15	D16
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

D17	D18	D19	D20	D21	D22	D23	D24	D25	D26
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

D27	D28	D29	D30	D31	D32	CKL	CKH
0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0x9E	0xE6

Figura 2.9: Ejemplo de comando *RX Password* enviado

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

Figura 2.10: Ejemplo de comando *RX Password* recibido

***Mass Erase* (CMD 15):**

Al ejecutar este comando se podrá realizar una limpieza masiva en la memoria FRAM del dispositivo, sin afectar a la memoria de información ni a la RAM. Este no dependerá

de la contraseña para ejecutarse. En función de la familia a la que pertenezca nuestro microcontrolador, se recibirá respuesta o no. En nuestro caso, como pertenece a la familia FR6XXX no nos devuelve respuesta, por lo que no se contará con un paquete de recepción sino tan solo con uno de transmisión.

Header	Length	Length	CMD	CKL	CKH
0x80	0x01	0x00	0x15	0x64	0xA3

Figura 2.11: Ejemplo de comando *Mass Erase* enviado

ACK	Header	Length	Length	CMD	MSG	CKL	CKH
0x00	0x80	0x02	0x00	0x3B	0x00	0x60	0xC4

Figura 2.12: Ejemplo de comando *RX Password* recibido

***CRC Check* (CMD 16):**

A través de éste comando se podrá ejecutar una suma de verificación para una cierta dirección de la memoria y para una longitud de bytes especificados. Para ello, será necesaria previamente la contraseña. Una vez ejecutada la comprobación, se recibirá respuesta del Bootloader que contendrá la suma de verificación.

Header	Length	Length	CMD	AL	AM	AH	D1	D2	CKL	CKH
0x80	0x06	0x00	0x16	0x00	0x44	0x00	0x00	0x04	0x9C	0x7D

Figura 2.13: Ejemplo de comando *CRC Check* enviado

ACK	Header	Length	Length	CMD	D1	D2	CKL	CKH
0x00	0x80	0x03	0x00	0x3A	0x55	0xAA	0x12	0x2B

Figura 2.14: Ejemplo de comando *CRC Check* recibido

***Load PC* (CMD 17):**

Éste comando permite escoger en qué dirección de la memoria queremos que inicie el microcontrolador. Para ejecutar el comando se requerirá de contraseña previa. Hay que tener cuidado con este comando ya que al no ejecutar un reset, toda la configuración empleada para iniciar el Bootloader se mantiene, pudiendo provocar errores inesperados, como por ejemplo en el uso de los relojes. Este comando no recibirá un paquete de respuesta por parte del microcontrolador.

Header	Length	Length	CMD	AL	AM	AH	CKL	CKH
0x80	0x04	0x00	0x17	0x00	0x44	0x00	0x42	0x0F

Figura 2.15: Ejemplo de comando *Load PC* enviado

***TX Data Block* (CMD 18):**

Ejecutando este comando el usuario podrá solicitar que el Bootloader le transmite ciertos bytes de la memoria que desee, tras indicarle la dirección en la que se encuentra el primer byte y la longitud de la trama. Para ello, será necesaria la contraseña. Como la cantidad de datos a recibir dependerá de la longitud especificada por el usuario y del buffer del dispositivo puente, es posible que se reciba más de un paquete de respuesta si el número de bytes solicitados supera la capacidad del buffer.

Header	Length	Length	CMD	AL	AM	AH	D1	D2	CKL	CKH
0x80	0x06	0x00	0x18	0x00	0x1C	0x00	0x04	0x00	0x87	0x81

Figura 2.16: Ejemplo de comando *TX Data Block* enviado

ACK	Header	Length	Length	CMD	D1	D2	D3	D4	CKL	CKH
0x00	0x80	0x05	0x00	0x3A	0x11	0x33	0x55	0x77	0x90	0x55

Figura 2.17: Ejemplo de comando *TX Data Block* recibido

***TX BSL Version* (CMD 19):**

Con este comando se puede comprobar la versión del Bootloader del dispositivo. Al ejecutarlo, se recibirá respuesta del Bootloader con la información del vendedor, la versión del interpretador de comandos, la versión de la API (sección de código encargada de leer y escribir en la memoria) y la versión de la interfaz periférica (sección de código que se encarga de la comunicación). Este comando requerirá de contraseña.

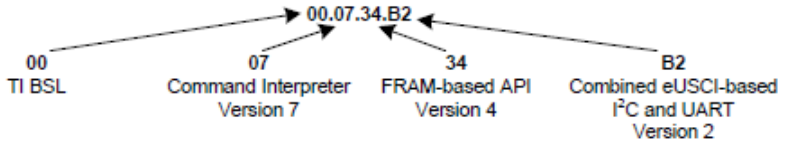

Figura 2.18: Ejemplo de versión de Bootloader

Header	Length	Length	CMD	CKL	CKH
0x80	0x01	0x00	0x19	0xE8	0x62

Figura 2.19: Ejemplo de comando *TX BSL Version* enviado

ACK	Header	Length	Length	CMD	D1	D2	D3	D4	CKL	CKH
0x00	0x80	0x05	0x00	0x3A	0x00	0x07	0x34	0xB2	0x14	0x90

Figura 2.20: Ejemplo de comando *TX BSL Version* recibido

***RX Data Block Fast* (CMD 1B):**

Este comando es prácticamente idéntico al *RX Data Block*, sin embargo no se recibirá respuesta por parte del microcontrolador, lo cual permitirá aumentar la velocidad de transmisión. Será necesaria la contraseña previamente y el paquete de envío será idéntico al del *RX Data block*, exceptuando el byte de CMD.

Header	Length	Length	CMD	AL	AM	AH	D1	D2	D3	D4	CKL	CKH
0x80	0x08	0x00	0x1B	0x00	0x00	0x01	0x10	0x32	0x54	0x76	0x3C	0x1C

Figura 2.21: Ejemplo de comando *RX Data Block Fast* enviado

***Change Baud Rate* (CMD 52):**

Finalmente, con éste último comando se podrá cambiar la velocidad de transmisión para el envío de paquetes reduciendo los tiempos de carga. Para cambiarlo, se deberá especificar la velocidad que queremos a través de un byte en concreto. Un único byte se recibirá como respuesta, el de ACK, para indicar si se ha cambiado correctamente. A continuación, se adjuntará una tabla (2.4) con los bytes específicos para cada velocidad y un ejemplo de paquete de envío y de recepción.

D1	**Baud Rate**
0x02	9600
0x03	19200
0x04	38400
0x05	57600
0x06	115200

Cuadro 2.4: Bytes de configuración de la velocidad de transmisión

Header	Length	Length	CMD	D1	CKL	CKH
0x80	0x02	0x00	0x52	0x06	0x14	0x15

Figura 2.22: Ejemplo de comando *Change Baud Rate* enviado

2.2.4. Secuencia de actualización del firmware

El Bootloader, a pesar de tener más posibles usos, se emplea principalmente para grabar firmware a un dispositivo. A continuación, se detallará paso a paso como esto se suele realizar y garantizar así un correcto uso.

Lo primero que se deberá hacer es establecer conexión con el Bootloader. Para ello, se deberá de acceder al Bootloader a través de la secuencia generada por los pines RST y TEST.

Una vez hayamos accedido al Bootloader, se deberá de borrar la imagen previa que contenía el dispositivo. Esto se puede realizar o bien a través del comando *Mass Erase* o enviando una contraseña incorrecta. Se recomienda borrar la memoria a través de la contraseña al ser de utilidad en caso de que el dispositivo ya esté vacío, ya que accedería directamente al resto de comandos protegidos al tratarse de la clave correcta. En caso de contener ya una imagen guardada en memoria, se deberá de ejecutar el comando de contraseña dos veces, una para limpiar el dispositivo y otra para acceder a los comandos protegidos.

Cuando el dispositivo ya se encuentre vacío y se haya introducido la contraseña correcta, se procederá a escribir la nueva imagen en el microcontrolador. Para ello se hará uso del comando *RX Data Block* o bien *RX Data Block Fast*. Dicha imagen puede ser enviada de distintas formas, pero en nuestro caso descargaremos la imagen a través de un archivo de texto en formato *TI-TXT*, el cual será interpretado por un pequeño programa que se encargará de interpretarlo y almacenarlo adecuadamente. Este formato puede ser extraído directamente del programa con el que hayamos programado la imagen, como puede ser *Code Composer* o bien *IAR*.

Teniendo ya la imagen descargada en el dispositivo, se deberá de comprobar que el código enviado ha sido almacenado e interpretado correctamente por el microcontrolador. Para ello, se puede ejecutar un *CRC Check* o bien usar el comando *TX Data Block* para que devuelva el código completo y así comprobarlo manualmente. Es más conveniente hacer uso del *CRC Check*, ya que resulta más cómodo al no tener que comparar byte a byte el código enviado con el recibido. El anterior comando *RX Data Block Fast* ya ejecutará la función *CRC Check* una vez grabe los datos y nos mostrará el resultado en el paquete de respuesta.

Finalmente, una vez verificada la imagen se deberá de cerrar el modo Bootloader para así poder ejecutarla. Esto se puede hacer de dos formas: Usando el comando Load PC para así escoger la dirección donde empezar a ejecutar o reseteando el dispositivo a través del puerto RST. Es recomendable usar el segundo método, ya que así se pueden evitar errores debido a la configuración establecida durante el modo Bootloader, pudiendo afectar al rendimiento.

CAPÍTULO 3. SELECCIÓN Y DISEÑO DEL HARDWARE

3.1. Arquitectura

En función de como el usuario quiera comunicarse con el microcontrolador, si a través de USB o bien por Bluetooth, se tendrá que realizar una variación en el diseño del sistema en cuanto a hardware se refiere. Es por ello, que para obtener ámbas conexiones ha sido necesario diseñar dos sistemas individuales que permitan establecer el tipo de conexión deseada entre un ordenador y el microcontrolador.

El primer sistema que se describirá es el formado por la placa y el dispositivo puente USB, que permitirá intercambiar datos haciendo uso del protocolo UART a través de conexión serie con el ordenador. El dispositivo puente escogido ha sido el módulo FT232R, un chip que permitirá trasmitir los datos entre el ordenador y el microcontrolador, habiendo realizado la conversión de los datos para que estos puedan ser interpretados debidamente. A través de dicho dispositivo también se podrá alimentar al microcontrolador y se podrán gestionar los diferentes pines digitales de los que dispone el módulo y así realizar la secuencia de acceso al Bootloader.

Figura 3.1: Diagrama de bloques para el sistema de conexión serie

El otro sistema a diseñar es el formado por la placa y el dispositivo o módulo Bluetooth RN41-I/RM. Este sistema garantiza al usuario y al microcontrolador el intercambio de información de forma inalámbrica al disponer de una antena Bluetooth de clase 1. Para que esto sea posible, será necesario emparejar previamente el ordenador con el módulo Bluetooth. Se trata de un dispositivo configurable que permite modificar sus prestaciones y su rendimiento según lo requiera el usuario. Igual que ocurre con el FT232R, dispone de pines digitales para realizar la secuencia de invocación.

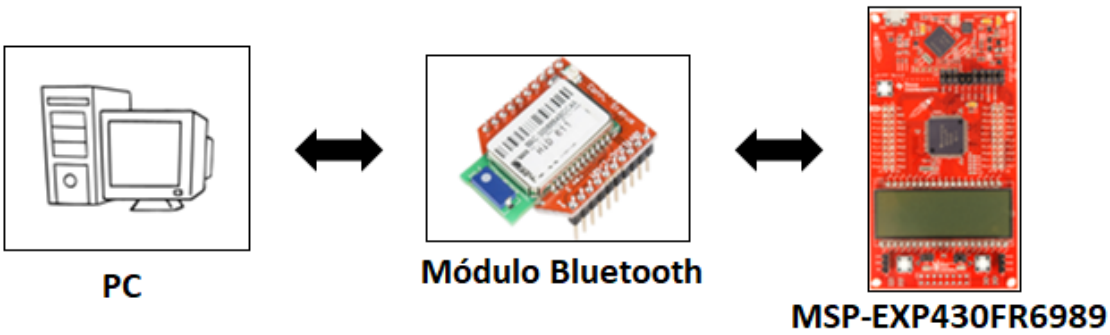

Figura 3.2: Diagrama de bloques para el sistema de conexión Bluetooth

En los siguientes apartados, se procederá a explicar en detalle las características principales de los dispositivos que forman los anteriores bloques (figuras 3.1 y 3.2).

3.2. Kit de desarrollo MSP-EXP430FR6989

El kit de desarrollo MSP-EXP430FR6989 [2] es un módulo fabricado por la casa Texas Instrument que contiene el chip MSP430FR6989 junto a multiples módulos. Dichos módulos complementan y amplían las posibilidades a la hora de trabajar con el microcontrolador, como son una pantalla LCD, un conjunto de botones, LEDs, etc.

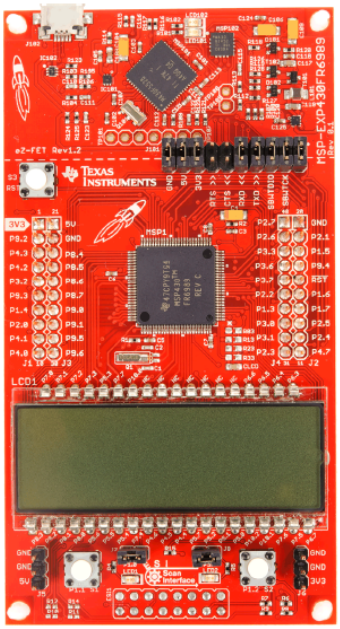

Figura 3.3: Kit de desarrollo MSP-EXP430FR6989

El chip MSP430FR6989 es el núcleo del módulo, donde se interpretan y almacenan todos los datos. Es un microcontrolador de última tecnología que se caracteriza por su bajo consumo, por mostrar un alto rendimiento y por disponer de una memoria RAM ferroeléctrica (FRAM) de 128 Kb. Incluye además un total de 40 pines, que pueden ser programados

para tener multitud de funcionalidades, como la posibilidad de añadir módulos externos ya sea para realizar conexiones inalámbricas o por cable, recoger medidas, etc. Otras de las características de este microcontrolador son las siguientes: opera con un voltaje que oscila entre los 1.8 V y 3.6 V, dispone de una arquitectura de 16 bits con un reloj del sistema a 16 MHz y otro de acceso a la FRAM de 8 MHz, FRAM de 128 Kb de almacenamiento, posibilidad de encriptado, un comparador, cinco contadores, y muchas más prestaciones. Además, dispone de 83 puertos que pueden ser modificados en función de las necesidades de nuestra aplicación. Se recomienda consultar la Datasheet del chip [3] para conocer mejor sus especificaciones.

A continuación, se adjuntará una figura (3.4) donde se podrá observar la distribución de los diferentes componentes de la placa, además de una figura (3.5) donde se mostrará las dos zonas en las que esta se divide.

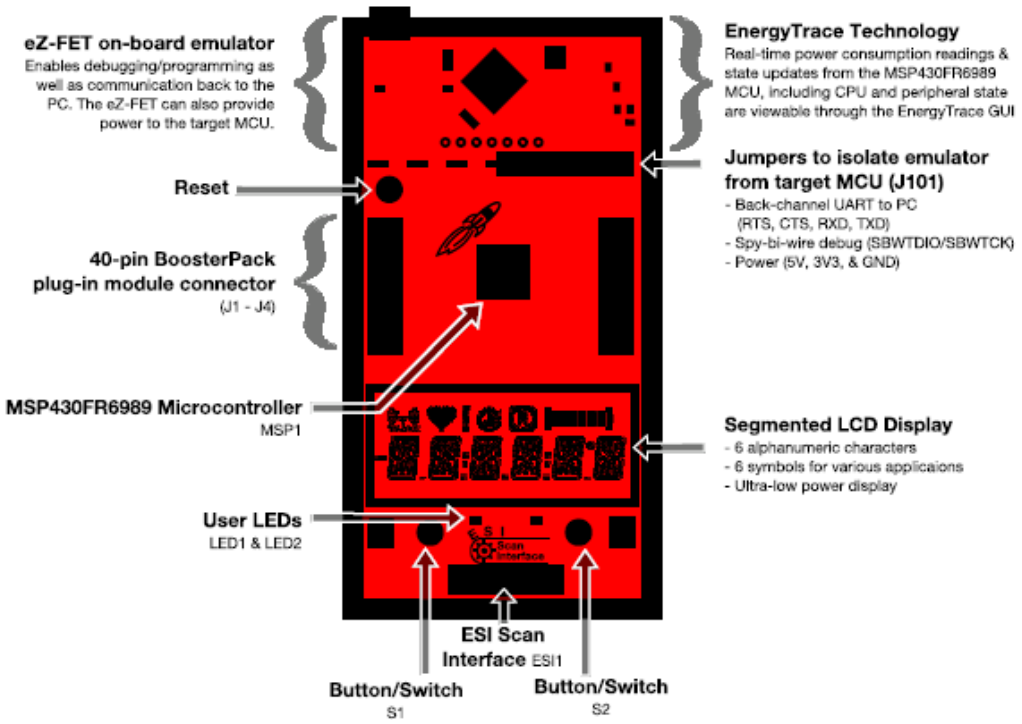

Figura 3.4: Arquitectura de la placa MSP-EXP430FR6989

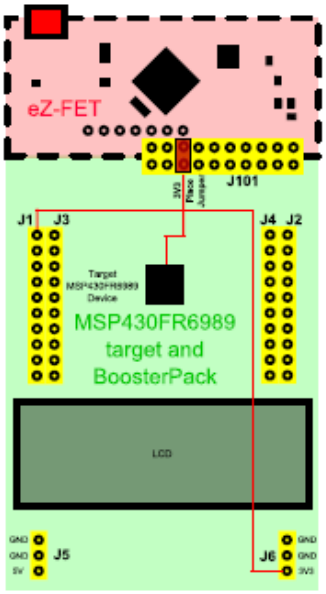

Figura 3.5: Arquitectura de la placa MSP-EXP430FR6989

Tal y como se puede observar en las anteriores figuras (3.5 y 3.4), se dispone de unos jumpers que se encargan de conectar las dos partes que forman la placa. La primera zona o sección es la formada por el módulo eZ-FET (zona roja de la figura 3.5). Se trata de un hardware tipo JTAG de bajo rendimiento que permite actualizar el firmware del dispositivo y depurar el programa durante la fase de desarrollo, a la vez que también alimenta a la placa. Por último, se encuentra la zona que está formada por el MSP430FR6989 junto a los distintos módulos que lo complementan (zona verde de la figura 3.5), como una pantalla LCD, conexiones de acceso a los puertos, dos LEDs, etc.

En caso de que se desconecten los jumpers, se puede aislar el microcontrolador del eZ-FET. Esto permite actualizar el firmware o configurar el chip a través del Bootloader que lleva incorporado y haciendo uso de los dispositivos puente. En dicho caso se hace uso de los puertos RX y TX ubicados en la placa que permiten transmitir y recibir datos y los puertos TEST y RST negada para realizar la secuencia de invocación.

3.3. Pasarela de comunicaciones serie/USB FT232R

3.3.1. Características

El FT232R [4] es un chip desarrollado por *Future Technology Devices International Ltd* (FTDI) que permite crear una conexión UART USB a serie para el intercambio de datos entre dispositivos. Este dispositivo se emplea debido a la falta de un puerto serial en el PC. Se trata de un módulo que simplifica el diseño de un dispositivo USB a serie al reducir la cantidad de componentes externos necesarios, ya que incorpora de serie una memoria no volátil EEPROM (*Electrically Erasable Programmable Read-Only Memory*) de 1024 bits que permite ser reprogramada eléctricamente, resistencias de terminación USB y un circuito con reloj integrado que no requiere de cristal externo. Además ha sido diseñado para poder rendir eficientemente sin ocupar todo el ancho de banda USB disponible.

Figura 3.6: Módulo serie FT232R

El chip cuenta con un ratio de transferencia de datos que oscila entre los 300 baudios y los 3 Megabaudios y con un buffer capaz de almacenar 128 bytes de recepción y 256 bytes de transmisión. También dispone de pines digitales configurables, LEDs para indicar el estado de transmisión y recepción de datos, modo UART con transmisión de 7 u 8 bits, 1 o 2 bits de parada y bit de paridad en caso de ser necesario, entre muchas otras características.

Finalmente, añadir que el FT232R incorpora la tecnologia FTDI-Chip-ID que permite aumentar la seguridad del dispositivo. Cada uno de los dispositivos fabricados cuenta con un número de identidad único y no programable, que evita que cualquier aplicación de un usuario pueda ser corrompida o copiada por un agente externo.

3.3.2. Configuración a través de FT_PROG

El FT232R puede ser configurado a través de una aplicación de la propia compañía llamada FT_PROG. Esta aplicación permite programar o borrar la memoria EEPROM de diversos dispositivos FTDI o bien programar el firmware ubicado en la memoria flash. A través de esta aplicación se puede configurar una gran variedad de parámetros: Alimentar al chip a través de USB o mediante una fuente externa, seleccionar la máxima corriente que circulará a través del USB (500 mA de corriente como máximo), cambiar el nombre del fabricante o la descripción del producto, regular la intensidad que puede circular por los pines de salida digital (de 4mA a 12mA), permitir el uso de un oscilador externo, invertir señales en ciertos pines o bien darle una función a cada uno de estos. Además, mediante esta aplicación podremos guardar diferentes configuraciones que más tarde podrán ser cargadas en el chip cuando sea necesario.

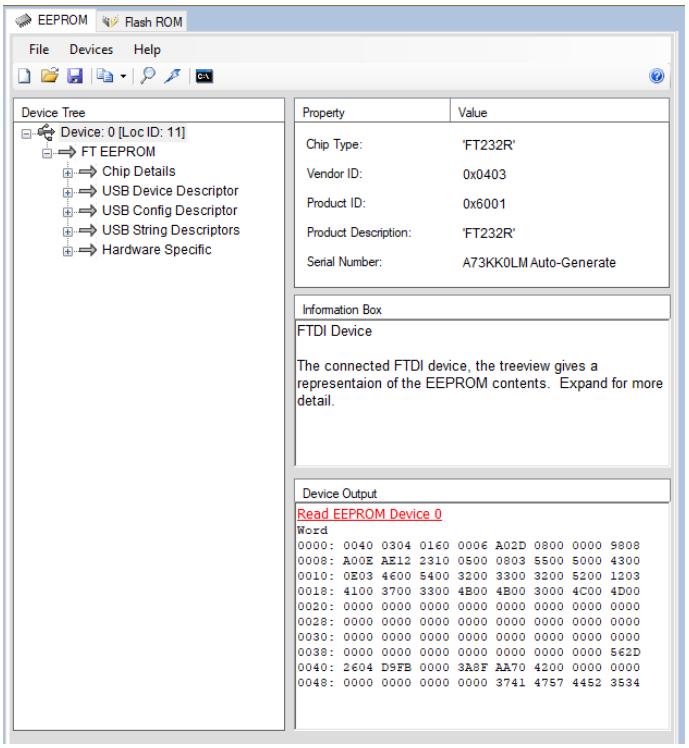

Figura 3.7: Menú principal FT_PROG

A la hora de configurar este dispositivo, tan solo se ha tenido en cuenta que la señal RTS estuviese invertida para así poder acceder correctamente al Bootloader.

3.3.3. Montaje del FT232R con el kit MSP-EXP430FR6989

Una vez esté el dispositivo FT232R configurado y listo para su uso, se conectará al kit de desarrollo MSP-430 para así hacer de puente entre el PC y el microcontrolador. Para ello, se deberán de conectar un conjunto de pines del FT232R a otros situados en el MSP-EXP430FR6989, siendo también necesario desconectar los Jumpers que conectaban con el eZ-FET y el kit de desarrollo. A continuación, se mostrará una tabla (3.1) que contendrá las conexiones necesarias entre dispositivos y una imagen (3.8) mostrando el resultado final.

FT232R	**MSP-EXP430FR6989**	**Color Cable**
GND	GND	Rojo
3.3V	3V3	Negro
RTS	SBWTCK (TEST)	Verde
DTR	RST negada	Lila
DOUT	P2.0 (BSLTX)	Azul
DIN	P2.1 (BSLRX)	Amarillo

Cuadro 3.1: Conexiones entre FT232R y la placa MSP-EXP430FR6989.

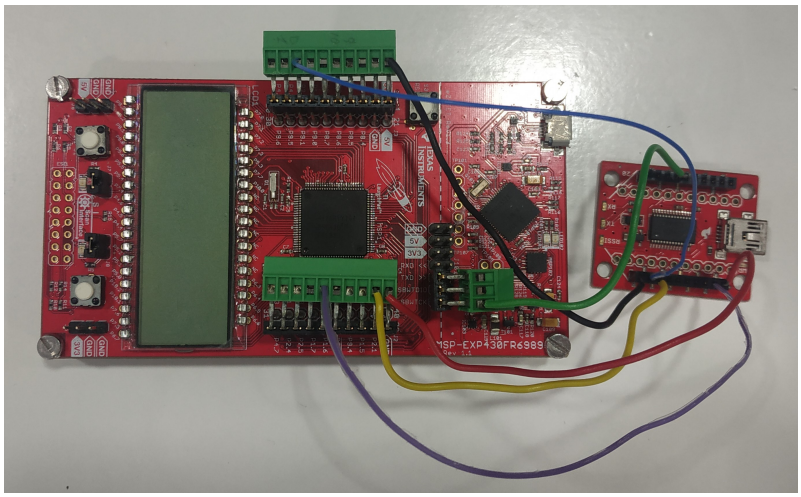

Figura 3.8: Conexión entre FT232R y placa MSP-430

Con el sistema ya montado, se puede proceder a conectarlo con el PC. Una vez conectado, lo primero que se deberá de comprobar es que se haya creado un puerto COM en la aplicación de administrador de dispositivos del ordenador. Se puede comprobar que la COM pertenece a este dispositivo si se accede sus propiedades al hacer clic derecho en esta. En caso de no aparecer ningún puerto para este dispositivo, es muy posible que se tengan que descargar e instalar los drivers desde la página oficial del fabricante.

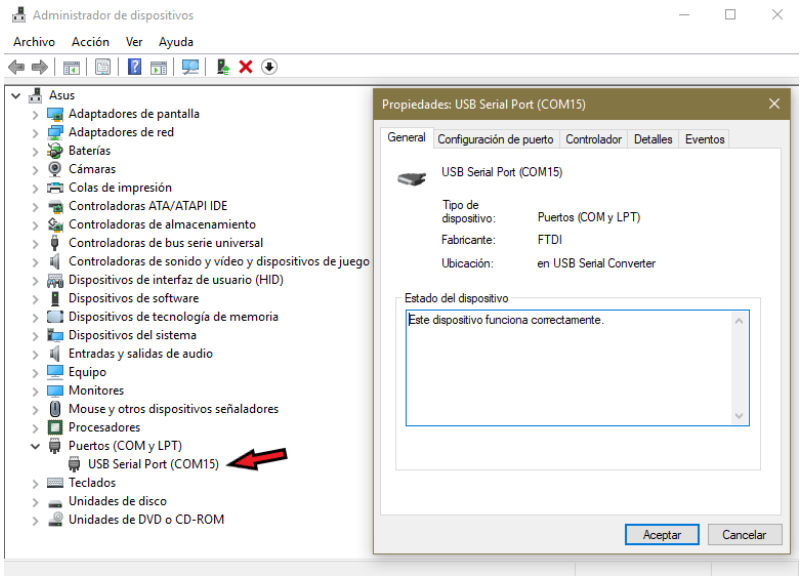

Figura 3.9: Puerto COM en Administrador de Dispositivos

A continuación, se deberá de acceder a los ajustes en el puerto COM y comprobar que la velocidad de transmisión del FT232R concuerde con la del microcontrolador, que en un principio están ambas ya configuradas a 9600 bits por segundo. Además, en el mismo apartado donde podemos cambiar la velocidad también se pueden cambiar parámetros como el número de bits por paquete, la paridad y el bit de parada.

Por último, para hacer uso de las principales funcionalidades que ofrece el FT232 en nues-

tro programa de LabVIEW, se ha descargado una librería de proyectos VI (Aplicación de *LabVIEW*) desde la página del fabricante, donde cada proyecto o VI ejecutará funciones concretas que nos servirán para construir la aplicación principal. Ejemplos de funciones que ofrecen estas VI son: una aplicación para limpiar el buffer del FT232R, otra para variar las señales de los pines digitales, una para leer o enviar datos, para abrir o cerrar conexión, etc. Además, todas ellas incluyen estados para informar al usuario de que se las aplicaciones se han ejecutado correctamente.

3.4. Módulo Bluetooth RN41-I/RM

3.4.1. Características

El módulo Bluetooth RN41-I/RM [5] es un dispositivo fabricado por *Roving Networks* que permite establecer conexión Bluetooth entre dispositivos de forma eficiente gracias a su bajo consumo. Es un dispositivo Bluetooth de clase 1, capaz de alcanzar un rango de conectividad de 100 metros y velocidades de transmisión de 3 Mbps. Resulta una solución ideal para aquellos usuarios que deseen añadir prestaciones inalámbricas a sus dispositivos sin realizar una gran inversión económica, ya que además soporta una gran variedad de protocolos de comunicación. También destaca por su fácil uso e implementación y sus certificaciones que lo catalogan como una solución segura. Este dispositivo viene montado encima de una placa que contiene los pines necesarios para conectarse con un dispositivo externo o para alimentarse. El conjunto formado por el módulo y la placa se denomina RN41XVC-I/RM y ayudará a conectar el módulo con el MSP-EXP430FR6989.

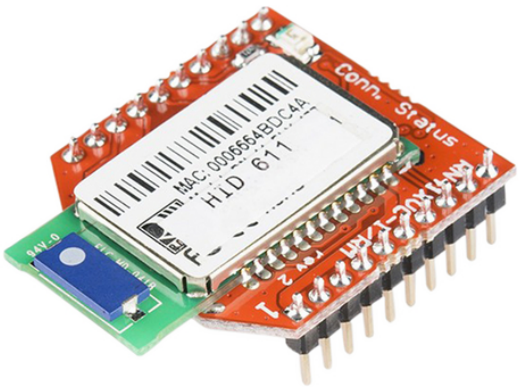

Figura 3.10: Módulo Bluetooth RN41XVC-I/RM

El dispositivo es controlado a través de comandos ASCII (*American Standard Code for Information Interchange*) transmitidos vía UART o bien a través de señales digitales provenientes de sus pines. El módulo incluye dos modos de trabajo distintos: el modo de datos y el modo comandos. El modo datos es la opción que inicia por defecto el dispositivo y que hace trabajar al dispositivo como puente entre el ordenador y el microcontrolador, gestionando y convirtiendo los datos transmitidos sin que estos datos afecten a la propia configuración del módulo. Mientras que en el modo comandos, se pueden configurar parámetros internos del propio RN-41, tales como la velocidad de transmisión, configurar

pines digitales, escoger entre los diferentes modos de trabajo de los que dispone, configurar las conexiones disponibles, etc.

3.4.2. Configuración del módulo RN41-I/RM a través del modo comandos

La configuración inicial con la que viene de por defecto es la siguiente:

- Modo Bluetooth Slave activado.
- Código PIN Bluetooth 1234.
- Conexión serie a 115,200 Kbps, 8 bits, sin paridad y 1 bit de parada.
- Control de flujo de puerto serie deshabilitado.
- Modo de baja potencia desactivado.

De inicio, alguno de los parámetros anteriores deberán de ser cambiados para poder ser compatibles con el MSP-430, como por ejemplo cambiar la velocidad de transmisión a 9600 baudios. Para configurarlo, se deberá de acceder al modo comandos del módulo a través de conexión serial o Bluetooth. Modificar este módulo permitirá configurar los pines del dispositivo Bluetooth para así generar la secuencia de invocación y acceder al Bootloader del microcontrolador.

Hay distintas formas de enviar los comandos al módulo, pero para modificar la configuración inicial es recomendable usar aplicaciones como *TeraTerm*, donde se establece conexión serie UART con el módulo y permite configurar sus propiedades. Para este proyecto, el RN41 ha sido configurado gracias a una placa Arduino Uno que cuenta con un hardware que se adapta a los pines del dispositivo Bluetooth, permitiendo conectarlo al PC y modificarlo a través de *TeraTerm*.

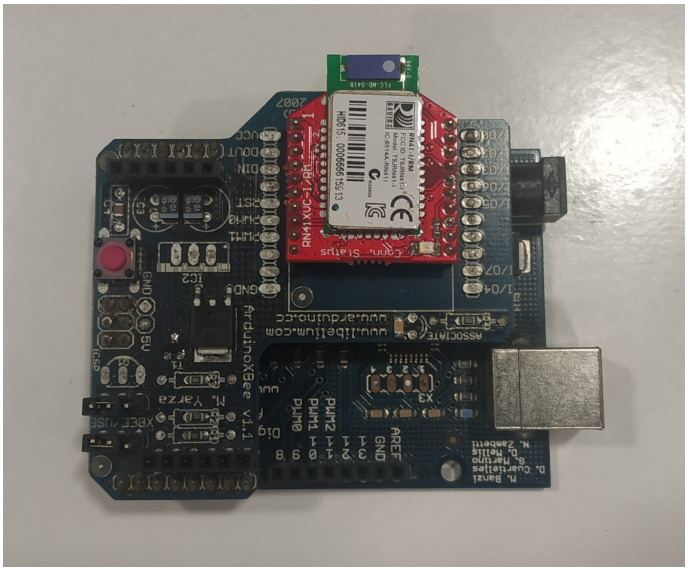

Figura 3.11: Placa Arduino Uno adaptada al módulo RN41XVC-I/RM

El emparejamiento entre dispositivos Bluetooth se realizará como se suele efectuar en la mayoría de dispositivos, esperando en primer lugar a que los dispositivos se encuentren, después emparejándolos con la clave PIN y finalmente estableciendo conexión.

Cada vez que se realice una conexión, se podrá acceder al modo comandos siempre y todo que no haya pasado el tiempo de expiración de 60 segundos. Este tiempo puede ser cambiado a través de comandos, pero esto ya se comentará más adelante.

Para acceder al modo comandos se deberá de escribir “$$$” en la terminal del emulador. Se puede distinguir el modo en el que se encuentra trabajando ya sea porque nos devuelve el mensaje CMD en pantalla o bien por la velocidad de parpadeo del LED verde. En caso de haber accedido al modo comandos, la frecuencia con la que parpadeará el LED será de 10 Hz, mientras que en modo datos el LED se quedará fijo. Además, si el módulo se queda esperando señal o buscando dispositivos, la frecuencia con la que el LED parpadeará será de 1 Hz. También puede ser útil escribir el comando “x” en la terminal para comprobar si estamos dentro o no, ya que en caso de sí haber accedido se nos devolverá un resumen de la configuración actual.

Una vez dentro del modo comandos, cada vez que se escriba y ejecute un comando que sea válido el módulo devolverá el mensaje “AOK”. Si el comando no ha sido bien interpretado la respuesta será “?” y si es incorrecta “ERR”.

Si se desea salir del modo comandos bastará con escribir “- - -” (sin dejar espacios) o bien resetear el módulo y volver a conectarlo con el dispositivo.

A continuación, se mostrará una imagen (3.12) con un ejemplo realizado en *TeraTerm*, donde primero se abrirá el modo comandos, a continuación se le solicitará que nos muestre la configuración actual con el comando “x” y finalmente se cerrará.

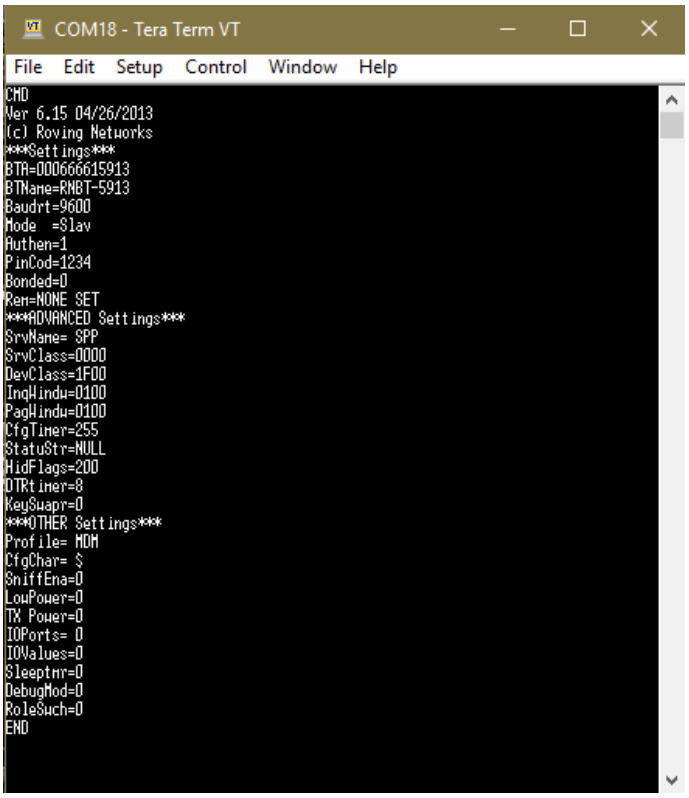

Figura 3.12: Ejemplo comandos TeraTerm

En caso de querer enviar uno de estos comandos, se deben de tener en cuenta los siguientes aspectos: Todo comando está formado por uno o dos caracteres que pueden ser mayúsculas o minúsculas, deben de estar delimitados por una coma, por lo general tienen entrada decimal y prácticamente todo comando es efectivo después de reiniciar el dispositivo.

En total, los comandos se clasifican en 5 categorías distintas: *Set commands*, *Get commands*, *Change commands*, *Action commands* y *GPIO commands*. A continuación, se explicará cada una de estas categorías y se mostrarán algunos de los comandos más útiles. Para ver el resto de comandos, consultar el capítulo *2. Command Reference* del documento [6].

3.4.2.1. Set commands

Esta categoría está formada por aquellos comandos que son útiles a la hora de guardar información en la memoria flash. Todo cambio realizado será efectivo después de un reinicio.

Algunos de los comandos más destacados son:

S7,≪*flag*≫: A través de este comando, se puede configurar el dispositivo para que trabaje con paquetes de de datos 7 u 8 bits. El valor *flag* será 0 para desactivarlo o 1 para activarlo.

SF,*1* Este comando permitirá restablecer el dispositivo a sus valores de fábrica.

SL,≪*char*≫ Gracias a este comando se podrá cambiar la paridad UART. El valor *char* vendrá determinado por un carácter el cual cambiará dicha propiedad. Si escribimos el valor E la paridad será par, con el valor O impar y con el valor N no tendrá paridad.

ST,≪*value*≫ Con este comando se podrá cambiar el tiempo de configuración escribiendo en *value* uno de los siguientes números decimales: Si escribimos el valor 0 no se podrá configurar remotamente ni de manera local. En caso de escribir un valor entre el 1 y el 252, este valor será el tiempo en segundos que habrá para configurar cada vez que se inicie el dispositivo. Si escribimos 253, se podrá configurar en modo local durante el tiempo que queramos. Si se quiere realizar lo anterior pero tan solo en modo remoto, se deberá de escribir 254. Finalmente, escribiendo 255 se podrá configurar en modo local o remoto durante el tiempo que queramos.

S˜,≪*value*≫ Este comando permite cambiar entre los diferentes perfiles de trabajo de los que dispone el módulo. De por defecto, el módulo está configurado con el perfil SPP, que no dispone de control de módem. Esto significa que no se podrán modificar las señales de los puertos digitales GPIO. Esto no interesa ya que sino no se pueden configurar los pines para realizar la secuencia de invocación. Para ello, se deberá de cambiar este perfil a MDM SPP, ya que este perfil permitie tener el control de dichas señales. Los diferentes perfiles que encontramos son: Valor 0 para perfil SPP, 1 para DUN-DCE (*Slave* o *Gateaway*), 2 para DUN-DTE (Mater o cliente), 3 para MDM SPP, 4 para SPP y DUN-DCE, 5 para APL (Perfil para dispositivos Apple) y 6 para el perfil HID.

3.4.2.2. Get commands

En esta segunda categoría se encuentran aquellos comandos que se encargan de coger información almacenada y mostrársela al usuario. Algunos de los comandos más destacados son los siguientes:

D Permite ver el estado de la configuración principal del dispositivo como son la dirección, el nombre, la configuración del UART, etc.

E Muestra la configuración del dispositivo de forma más extendida que el anterior comando.

3.4.2.3. Change & action commands

Gracias a los *Change comands* se puede cambiar el valor temporal de diferentes parámetros, como pueden ser la paridad, la velocidad de transmisión, etc. Mientras que los *Action comands* ejecutan acciones como la de establecer conexión, entrar o salir del modo comandos, por ejemplo.

$$$ Este comando permite acceder al modo comandos del módulo Bluetooth.

- - - Con este comando se puede salir del modo comandos. Los tres guiones se escriben seguidos, no separados.

C,≪*address*≫ Causa que el módulo Bluetooth se conecte a un dispositivo a través de su dirección. Además, la dirección es almacenada en la memoria.

R,*1* Se ejecuta un reinicio total del dispositivo.

U,≪*value1*≫,≪*value2*≫ Gracias a este comando se podrá cambiar temporalmente la velocidad de transmisión *value1* y la paridad *value2*, ya que el dispositivo cambia los valores pero no los almacena en memoria.

3.4.2.4. GPIO commands

Por último, en esta categoría encontramos comandos con los que poder configurar y manipular las señales de los puertos digitales GPIO. Cada comando estará formado por 2 bytes de 16 bits cada uno, donde el primer byte especifica el número del puerto GPIO y el segundo byte recoge el valor de datos que tendrá el comando.

A través de la configuración y control de estos puertos, se podrá acceder al modo Bootloader mediante la secuencia de invocación.

Los principales comandos son los siguientes:

S@,≪*hex value*≫ Con este comando se puede configurar la dirección hacia donde apuntará un cierto puerto GPIO, en caso de si se quiere que sea de entrada o de salida.

S&,≪*hex value*≫) A través de este comando se podrá variar el valor de salida de los pines GPIO.

S%,≪*hex value*≫ Gracias a este se asignará el valor que queramos que tenga un puerto GPIO cada vez que arranque el dispositivo.

S*,≪*hex value*≫ Este comando permite dar valor a los pines GPIO8, GPIO9, GPIO10 y GPIO11. Estos pines se controlan desde un comando distinto, debido a que hay algunos módulos que no disponen de ellos.

3.4.3. Montaje del módulo RN41-I/RM con el kit MSP-EXP430FR6989

Para poder conectar el módulo Bluetooth al MSP-430, se deberán de realizar las conexiones tal y como se muestran en la siguiente tabla (3.2) e imagen (3.13):

RN41-I/RM	**MSP-EXP430FR6989**	**Color Cable**
GND	GND	Rojo
VDD_3V3	3V3	Negro
GPIO10 (RTS)	SBWTCK (TEST)	Verde
GPIO11 (DTR)	RST negada	Lila
RX	P2.0 (BSLTX)	Azul
TX	P2.1 (BSLRX)	Amarillo

Cuadro 3.2: Conexiones entre RN41-I/RM y la placa MSP-EXP430FR6989.

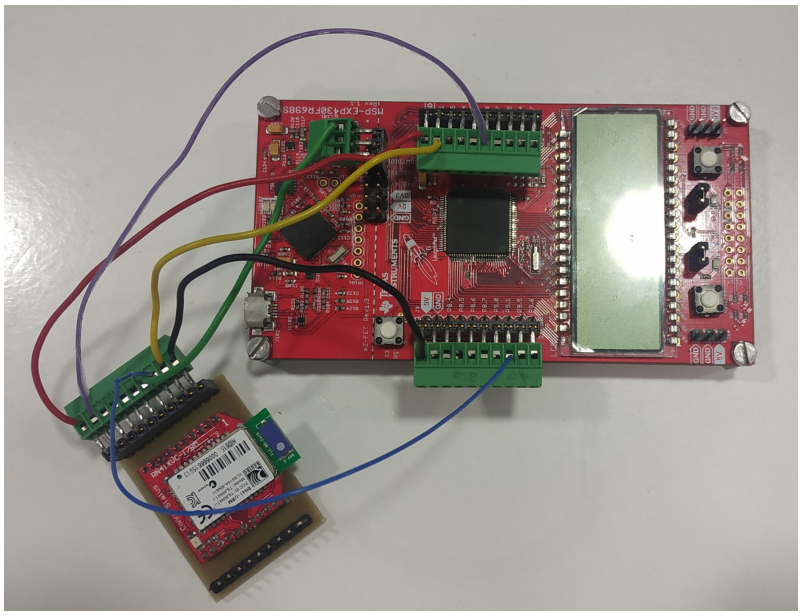

Figura 3.13: Conexión entre RN41-I/RM y placa MSP-430

Con el montaje ya listo, tan solo se debe de comprobar de que el PC y el módulo RN41 se hayan emparejado mediante Bluetooth para poder empezar a mandar comandos a través del Bootloader Scripter.

CAPÍTULO 4. BOOTLOADER SCRIPTER

4.1. Introducción

Bootloader Scripter es la aplicación que se ha desarrollado para acceder al Bootloader del MSP430FR6989 a través de conexión serie (FT232R) o Bluetooth (Módulo Bluetooth RN41-I/RM). Gracias a esta aplicación se pueden modificar prestaciones del microcontrolador, ejecutar ordenes específicas o bien actualizar el firmware. El documento *SLAU655F. Bootloader (BSL) Scripter* [7] servirá como guía a la hora de diseñar la aplicación.

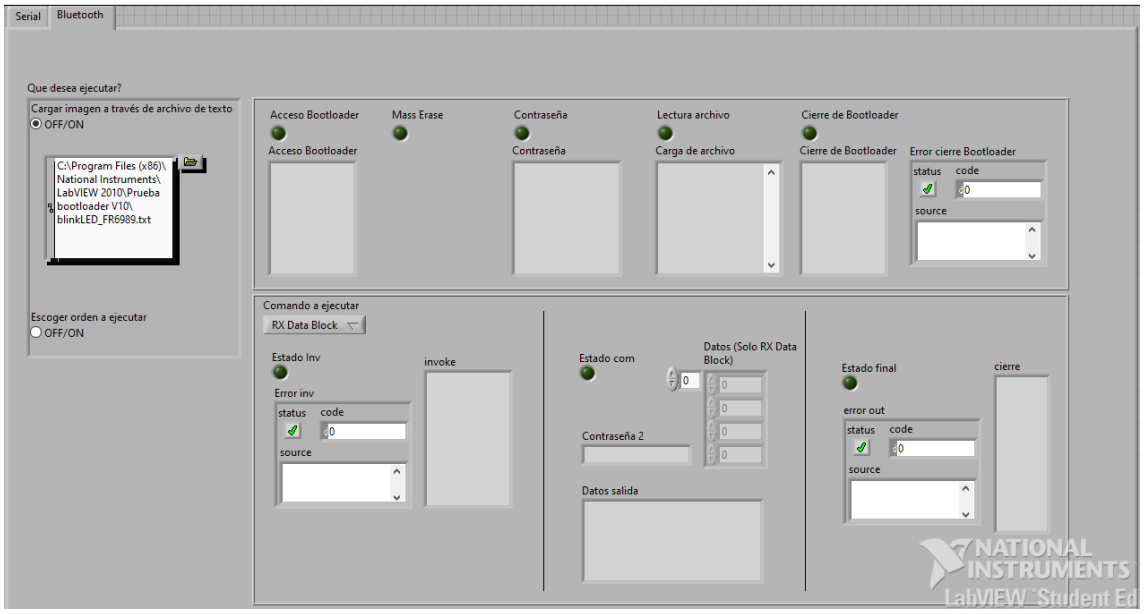

Figura 4.1: Menú principal del Bootloader Scripter

Esta aplicación se encarga de actualizar el firmware del microprocesador a través de la interpretación de un fichero de texto o bien a través de la introducción de estos bytes de forma manual. También dará la opción al usuario de ejecutar ciertos comandos incluidos en el Bootloader. Todos los comandos serán transmitidos a través de los dispositivos puente (FT232R o RN41) para después ser grabados en la memoria del chip. Además, esta interfaz también se encarga de recibir respuesta por parte del microcontrolador en caso de que la haya y mostrársela al usuario.

La aplicación ha sido creada a través de la plataforma LabVIEW, un entorno desarrollado para diseñar sistemas a través de un lenguaje de programación visual. Gracias a esta se podrá diseñar la aplicación principal que estará formada por un conjunto de programas o aplicaciones llamadas VIs o Instrumentos Visuales, que cumplirán con una función específica dentro del programa principal, como por ejemplo, iniciar conexión entre PC y el disipositivo puente o bien cerrarla, entre otros.

Para dicho programa, se ha realizado una programación jerárquica a través de los diferentes bloques que proporciona el fabricante para controlar el FT232R y a través de la librería de LabVIEW para controlar el módulo Bluetooth. Haciendo uso de dichos bloques

se creará una librería para la conexión USB y otra para la Bluetooth, que contendrán las funcionalidades básicas con las que poder controlar estos dispositivos y que serán implementadas en el programa principal. Conceptualmente ambas librerías son muy parecidas, ya que disponen de las mismas funcionalidades. Tan solo se diferenciarán por algunos bloques que serán distintos en función del dispositivo que vaya a ser usado.

La interfaz de usuario implementada permite escoger entre dos modos de trabajo según el tipo de conexión. El primer modo de trabajo se trata del modo lectura de fichero. Si ejecutamos esta opción, tan solo hará falta cargar la imagen en formato TI-TXT. Una vez cargada la imagen en la aplicación, el programa se encargará de ejecutar las ordenes y comandos necesarios para grabar los bytes del fichero en la memoria del MSP-430. El segundo modo permitirá seleccionar tres comandos a ejecutar de manera libre: mostrar la versión del Bootloader, realizar una limpieza masiva o enviar bytes a una cierta dirección de la memoria.

En el siguiente apartado se describirán las funcionalidades de ambas librerías y se realizará una breve explicación de la estructura de cada una de ellas y de como funcionan.

4.2. Librerías para el control del Bootloader

En este apartado se comentarán las principales VI que han sido diseñadas para las librerías de conexión serie y Bluetooth. A pesar de ser librerías con una estructura muy parecida, la principal diferencia se encuentra en las funciones que se usan dentro de las VI para ejecutar las ordenes. Debido a esta similitud, en los siguientes subapartados se explicará de manera conjunta cada una de las VI que las forman.

4.2.0.1. Abrir conexión

Esta VI permite al usuario abrir una conexión entre el ordenador y el dispositivo puente, ya sea el FT232R o bien el RN41. La figura (4.2) muestra concretamente la implementación para el FT232R.

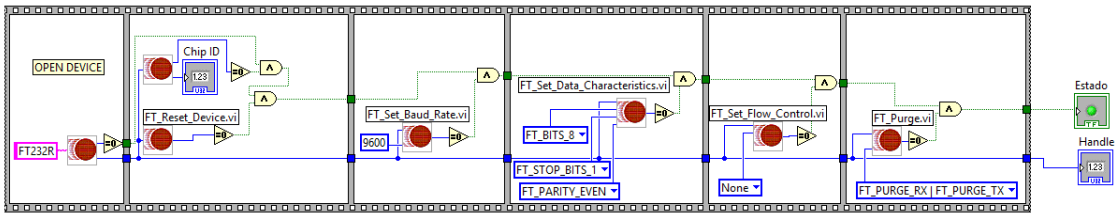

Figura 4.2: Abrir conexión para comunicación serie

En primer lugar la aplicación inicia conexión con el FT232R identificando a este con un nombre previamente asignado desde la aplicación FT_PROG. Una vez iniciada la conexión, se obtiene un Handle que será utilizado posteriormente por diferentes funciones de control del dispositivo y después se reseteará. A continuación, se configurará la velocidad de transmisión del FT232R a 9600 baudios y también el formato de los paquetes, que estarán formados por 8 bits, 1 bit de parada y 1 bit de paridad par. Este paso es necesario

para garantizar la compatibilidad de los paquetes entre dispositivos. También se considerará que no habrá control de flujo. Por último, tras realizar la configuración de los paquetes se limpiará tanto el buffer de salida como el de llegada, para que no haya posibilidad de transmitir o recibir datos de forma incorrecta.

La VI equivalente para conexión Bluetooth, la cual se muestra en la figura 4.3, utiliza la función Open Bluetooth Conection de la librería de LabVIEW. Para establecer conexión, se identifica al dispositivo a través de la dirección MAC (Media Access Control) del módulo Bluetooth y se establece un tiempo máximo de espera de 5000ms. Antes de proceder a configurar el dispositivo, se realiza una pausa de 1100ms ya que es el tiempo mínimo que se requiere para acceder al modo comandos después de iniciar conexión. Por último, se revisará el estado de la aplicación y si el estado es correcto, significará que se ha ejecutado correctamente la secuencia de acceso al Bootloader. En caso de que el estado sea erróneo, se cerrará la conexión y se volverá al inicio para abrir esta de nuevo y realizar otra vez la secuencia. Este bucle puede ser ejecutado como máximo 5 veces. Si da error para todos los intentos, se cerrará el bucle y con él finalizará la aplicación sin haber podido iniciar conexión.

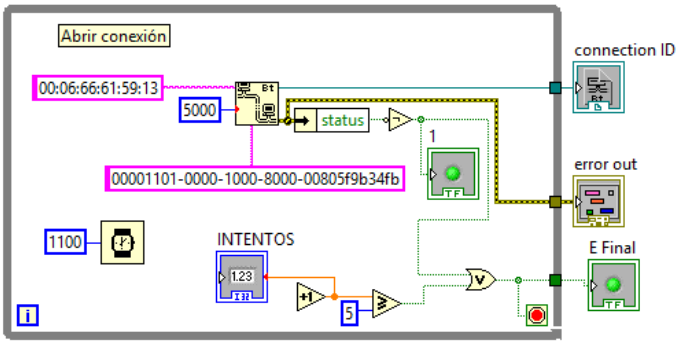

Figura 4.3: Abrir conexión para comunicación Bluetooth

4.2.0.2. Cerrar conexión

Esta VI se encarga de cerrar la conexión tanto para conexión serie como para Bluetooth. Es importante cerrar una conexión que no vaya a ser usada para así evitar posibles problemas en caso de volver a querer establecerla. A continuación, se adjuntará una figura (4.4) donde se mostrará la implementación de esta función para el FT232R.

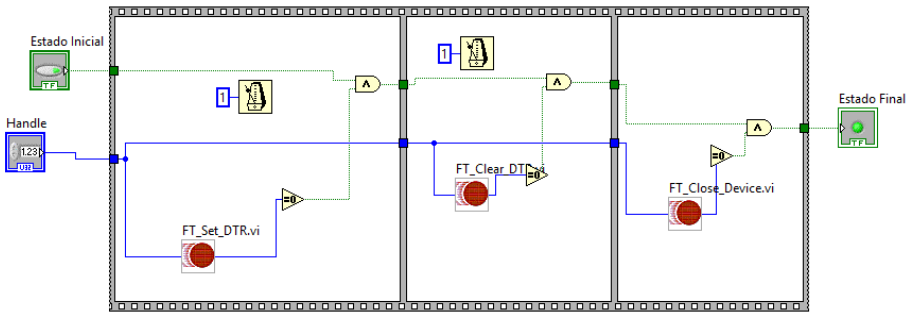

Figura 4.4: Cerrar conexión para comunicación serie

En primer lugar, antes de cerrar la conexión se debe de salir del modo Bootloader. Para ello antes se deberá de reiniciar el dispositivo MSP-430. Como el puerto RST de la placa está controlado a través del puerto DTR del FT232R, modificaremos la señal del DTR para así a reiniciar el dispositivo. Para finalizar, se puede procede a cerrar la conexión con el dispositivo puente.

En cuanto a la VI creada para la comunicación Bluetooth (figuras 4.5 y 4.6), primero se iniciará el modo comandos para así reiniciar el microcontrolador a través del puerto digital GPIO11 (DTR), después se cerrará el modo comandos y por último se cerrará la conexión con la función *Bluetooth Close Connection*.

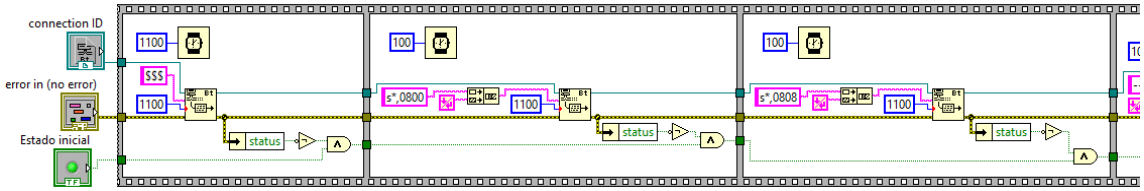

Figura 4.5: Cerrar conexión para comunicación Bluetooth (1)

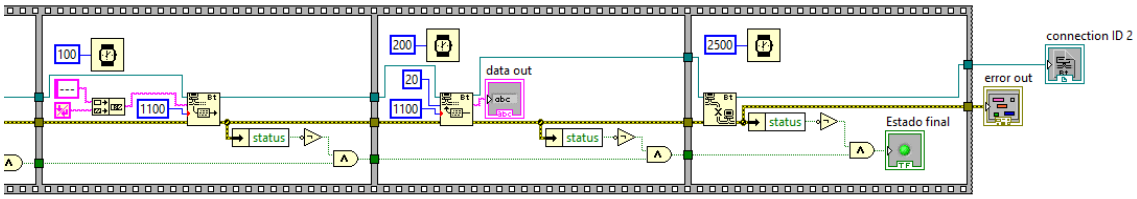

Figura 4.6: Cerrar conexión para comunicación Bluetooth (2)

4.2.0.3. Invocación del Bootloader

Esta VI se encargará de realizar la secuencia de invocación del Bootloader. A continuación, se mostrará la estructura de la aplicación para la conexión serie en la figura 4.7.

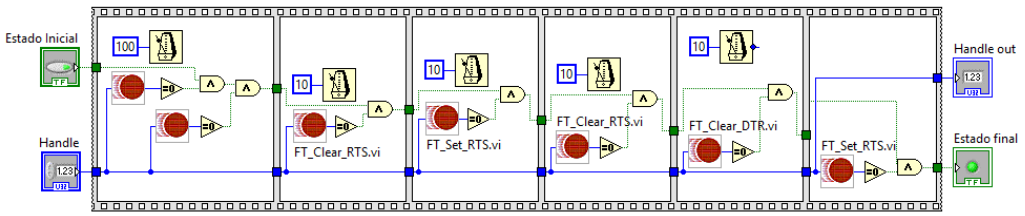

Figura 4.7: Invocación del Bootloader para comunicación serie

Primero se modifican las señales del DTR y RTS, dejando ambas señales bajas y listas para iniciar la secuencia. Después, se iniciará la secuencia modificando ambas señales y siguiendo el orden establecido en la figura 2.2, respetando un tiempo mínimo para cada cambio de señal.

En la VI para conexión Bluetooth (figuras 4.8, 4.9 y 4.10), el dispositivo accederá al modo comandos para así poder controlar los puertos digitales. Todo esto se hará a través de la función *Bluetooth Write*, donde se escribirá como entrada la orden en formato script. Un tiempo de espera de 1.1 segundos ha sido también añadido al acceder al modo comandos para garantizar que el dispositivo tenga suficiente tiempo para cambiar de modo de funcionamiento. En cuanto haya pasado ese tiempo de espera, se comenzarán a modificar los puertos a través de la función de “Write”, siguiendo la secuencia y respetando los tiempos para cada cambio de señal. Cuando ya se haya enviado la secuencia se saldrá del modo comandos y se leerá la respuesta del microcontrolador. Finalmente, se verifica que no haya habido errores durante la ejecución. En caso de error, la secuencia puede repetirse hasta 5 veces para así conseguir mayor fiabilidad de funcionamiento.

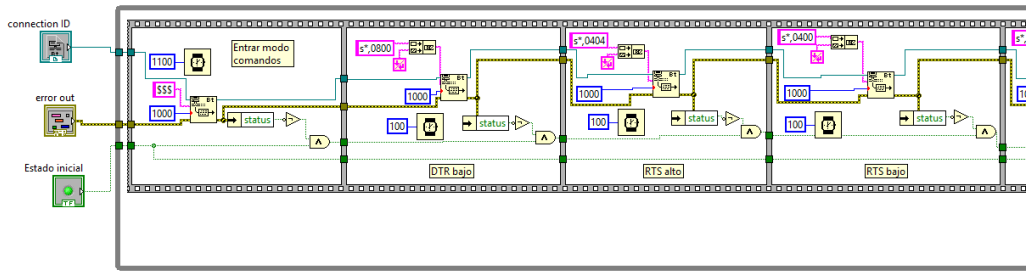

Figura 4.8: Invocación del Bootloader para comunicación Bluetooth (1)

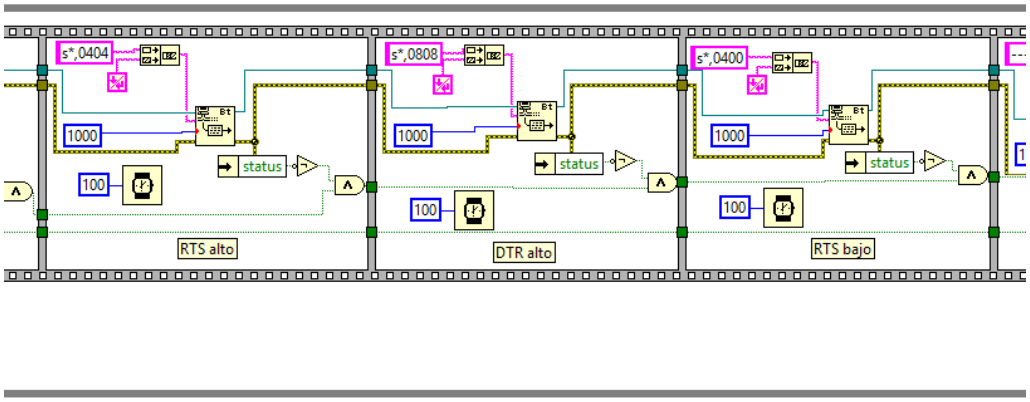

Figura 4.9: Invocación del Bootloader para comunicación Bluetooth (2)

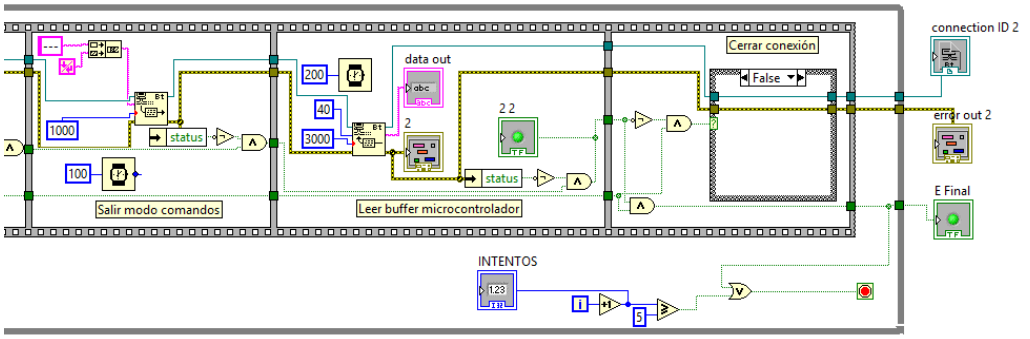

Figura 4.10: Invocación del Bootloader para comunicación Bluetooth (3)

4.2.0.4. Comandos

En esta VI se le ordenará al microcontrolador que ejecute los distintos comandos de los que dispone el Bootloader. Esta aplicación se encargará de realizar todo lo relacionado con la ejecución de comandos, desde enviar el paquete en su debido formato a recibir la respuesta por parte del microcontrolador.

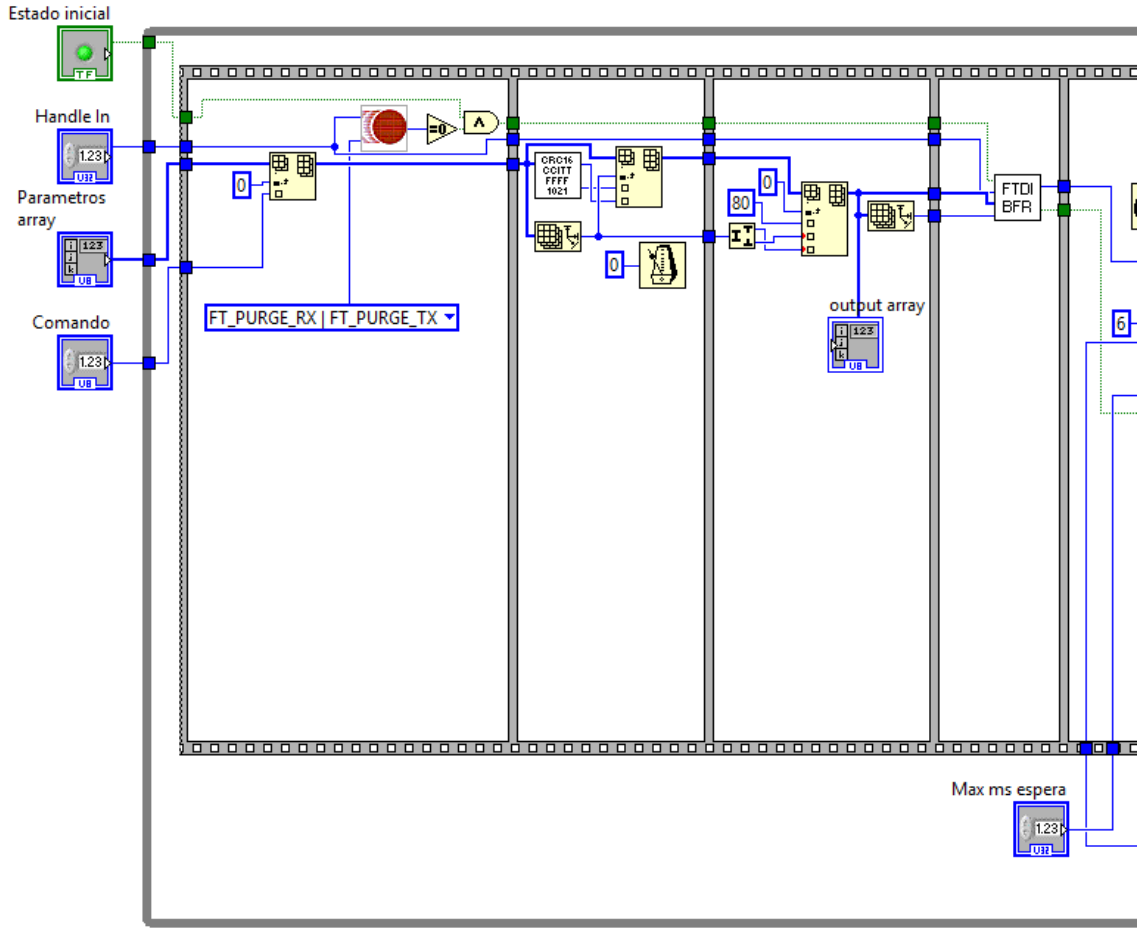

Figura 4.11: VI de comandos para comunicación serie (1)

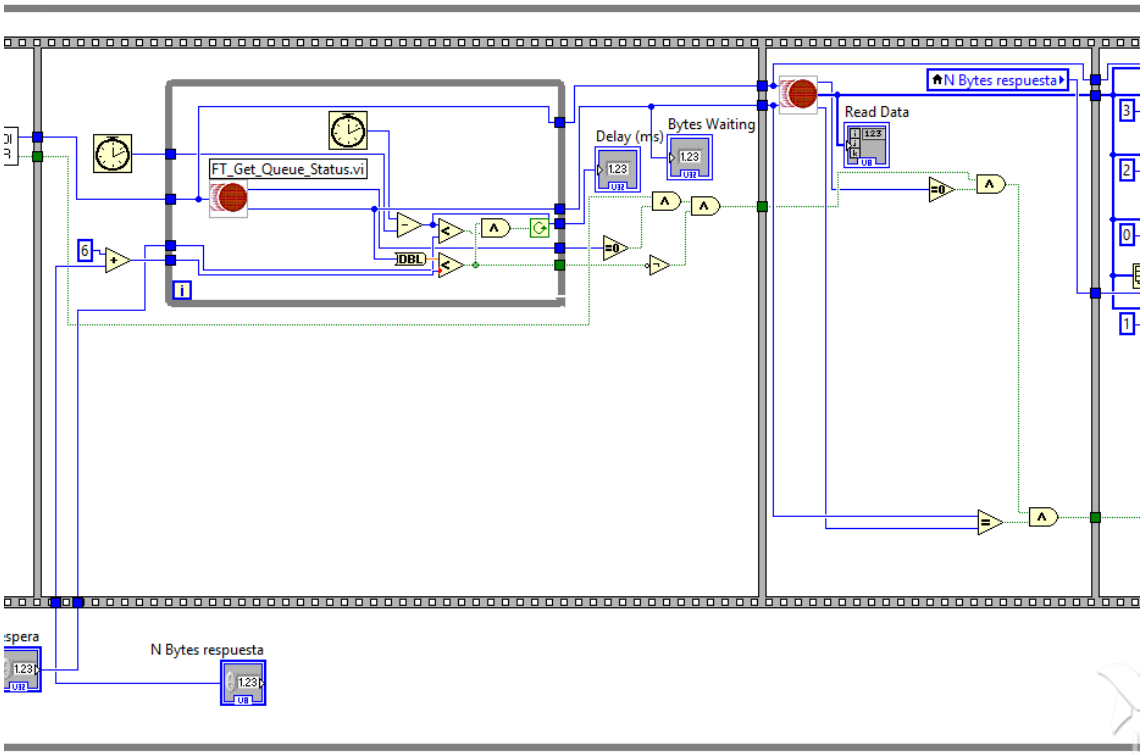

Figura 4.12: VI de comandos para comunicación serie (2)

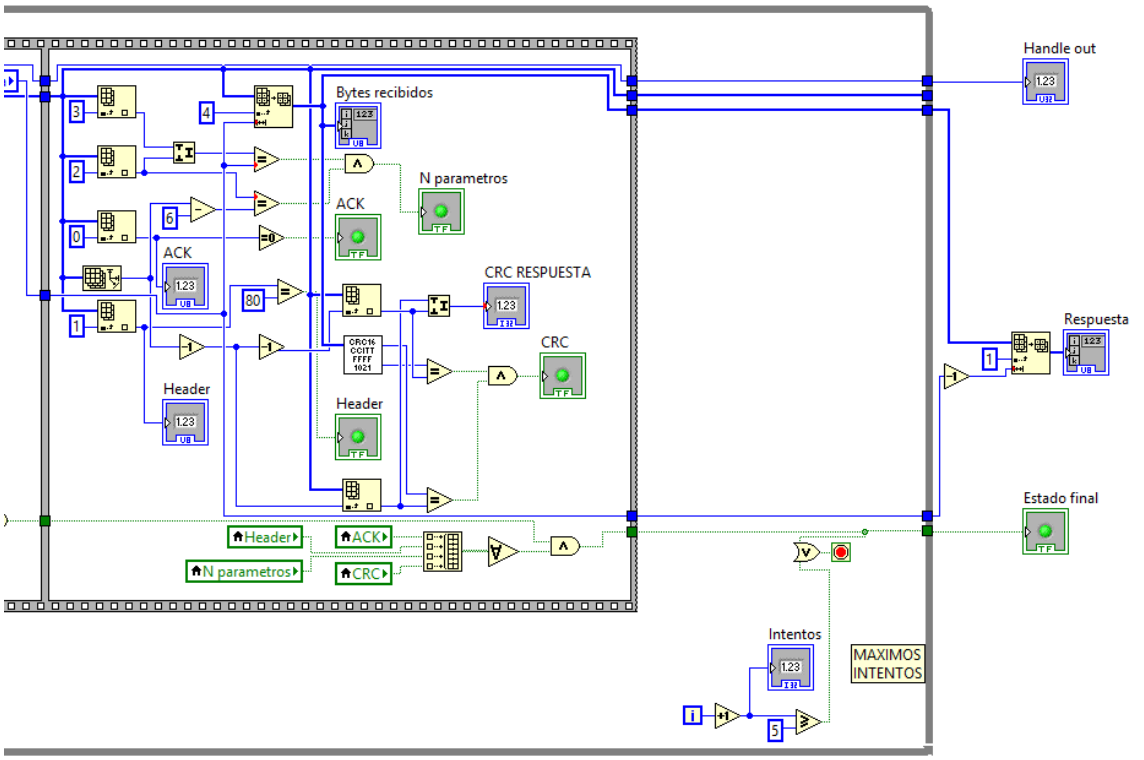

Figura 4.13: VI de comandos para comunicación serie (3)

Para la conexión serie (figuras 4.11, 4.12 y 4.13), nada más iniciar la VI se limpiará el buffer de entrada y de salida de datos y a continuación se procederá a dar forma al paquete que será transmitido al microcontrolador. Dicho paquete contendrá el número hexadecimal que representa al comando y los datos relacionados con este en caso de ser necesario. En el paquete también se incluirá la cabecera, la longitud del vector formado por el byte de comando más el parámetro de datos y los bytes de la suma de verificación o *Checksum*. Acto seguido se procederá a enviar el paquete al microcontrolador a través de una pequeña VI (FTDI BFR) creada para gestionar la cantidad de bytes enviados al microcontrolador para así evitar que se colapse el buffer. Esta aplicación divide el total de datos a transmitir en paquetes de 200 bytes, transmitiendo uno a uno en caso de ser necesario. Debajo se adjuntarán dos figuras (4.14 y 4.15) donde se podrá observar esta VI.

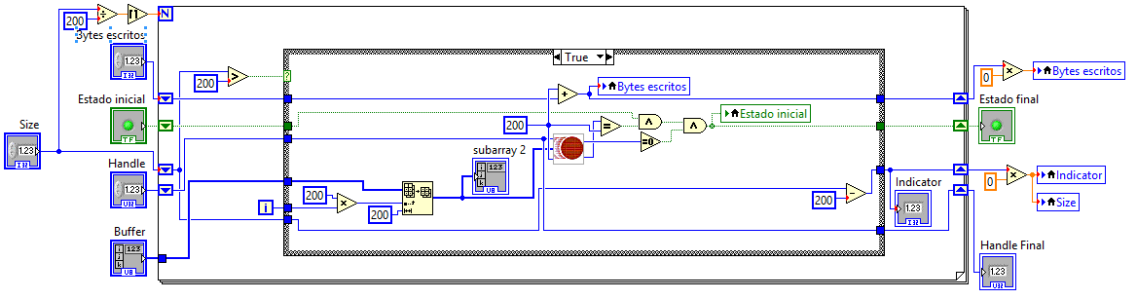

Figura 4.14: Estructura de la VI de gestión de bytes (1)

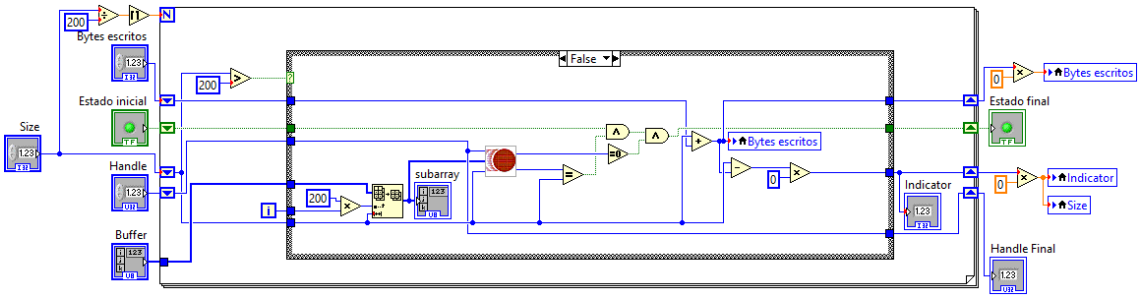

Figura 4.15: Estructura de la VI de gestión de bytes (2)

Como se puede apreciar en las figuras 4.14 y 4.15, esta aplicación recibirá a través del vector “Buffer” un conjunto de bytes para ser grabados en la memoria del microcontrolador. La función *Size* mide la longitud del vector para saber así la cantidad de bytes que se tienen que transmitir. Si la cantidad excede los 200 bytes, se almacenarán los primeros 200 en un vector de datos para después ser transmitidos. Si una vez transmitidos, la cantidad de bytes restantes sigue siendo mayor a 200, se repetirá el mismo procedimiento y así tantas veces como sea necesario. En caso de que la cantidad de bytes sea inferior a 200, estos serán transmitidos y finalmente se cerrará la aplicación, indicando el estado final de esta.

Una vez se ha enviado, interpretado y escrito el paquete en la memoria del microcontrolador, la aplicación dispone de un proceso de verificación de la respuesta del microcontrolador. En este proceso se verifican parámetros como la longitud y el CRC del mensaje recibido, la recepción de un ACK y de que no haya habido error durante toda la secuencia.

En el caso de la aplicación para la conexión Bluetooth, esta tendrá el siguiente aspecto:

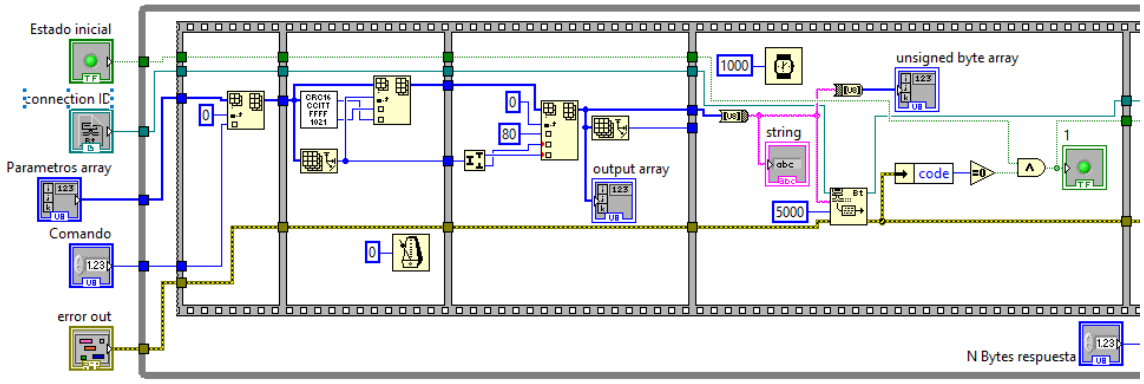

Figura 4.16: VI de comandos para comunicación Bluetooth (1)

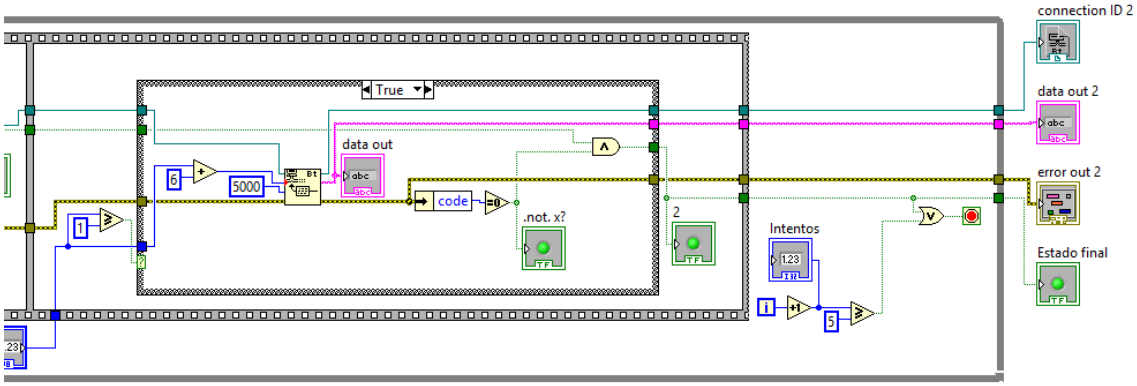

Figura 4.17: VI de comandos para comunicación Bluetooth (2)

Como se puede observar, la aplicación iniciará formando el paquete de la misma manera que en la VI para conexión serie, la única diferencia será que no se limpiará el buffer. Acto seguido, se realizará una conversión del formato del paquete de entero a script para que pueda ser interpretado por el módulo. Después, a través de la función *Bluetooth Write* enviaremos esta trama al microcontrolador para que sea grabada en la memoria. Si el comando transmitido dispone de respuesta, esta se obtendrá mediante la función *Read Bluetooth*. Por último, se comprueba que el estado final de la aplicación sea correcto.

4.3. Aplicación principal

A continuación, se explicarán las funcionalidades que ofrece el programa principal creado a partir de las funciones VI de la librería anteriormente descritas.

4.3.1. Cargar imagen a través de archivo de texto

Esta aplicación ha sido creada con el fin de transmitir al Bootloader los datos correspondientes al firmware de una aplicación a través de un fichero de texto. Este fichero de texto

ha sido creado a partir de los compiladores CodeComposer y IAR [7].

La figura 4.18 muestra el formato de este archivo de texto, distinguiendo dos tipos de datos los cuales estan separados por espacios. Los datos precedidos por el símbolo “@” indican la dirección de memoria en hexadecimal correspondiente a la primera dirección del siguiente dato. Esta dirección incrementará progresivamente con cada nuevo dato. El resto de datos corresponden a datos en hexadecimal de ocho bits de cada dirección de memoria. Por último, para indicar el fin de texto se hace uso de la letra “q”.

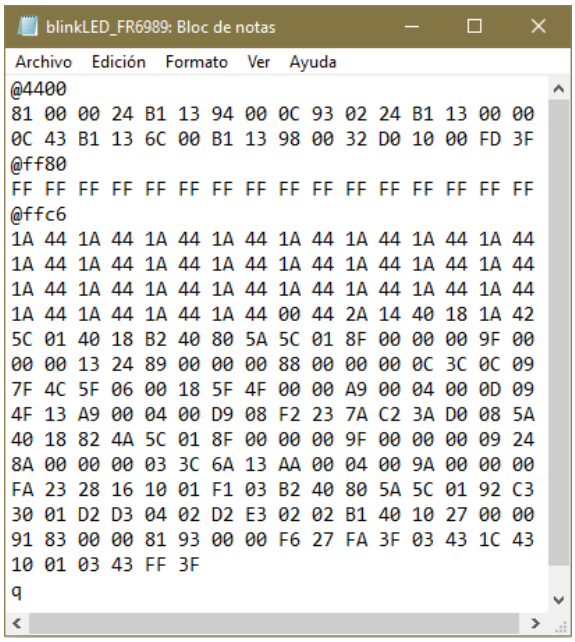

Figura 4.18: Archivo de texto TI-TXT

Para interpretar el archivo de texto, se ha creado una VI llamada VI TXT. Esta VI tiene una versión para la conexión serie y otra para la Bluetooth cuya única diferencia será la VI Comandos que se usará. Debajo se podrán observar dos figuras (4.19 y 4.20) que mostrarán la estructura de esta aplicación.

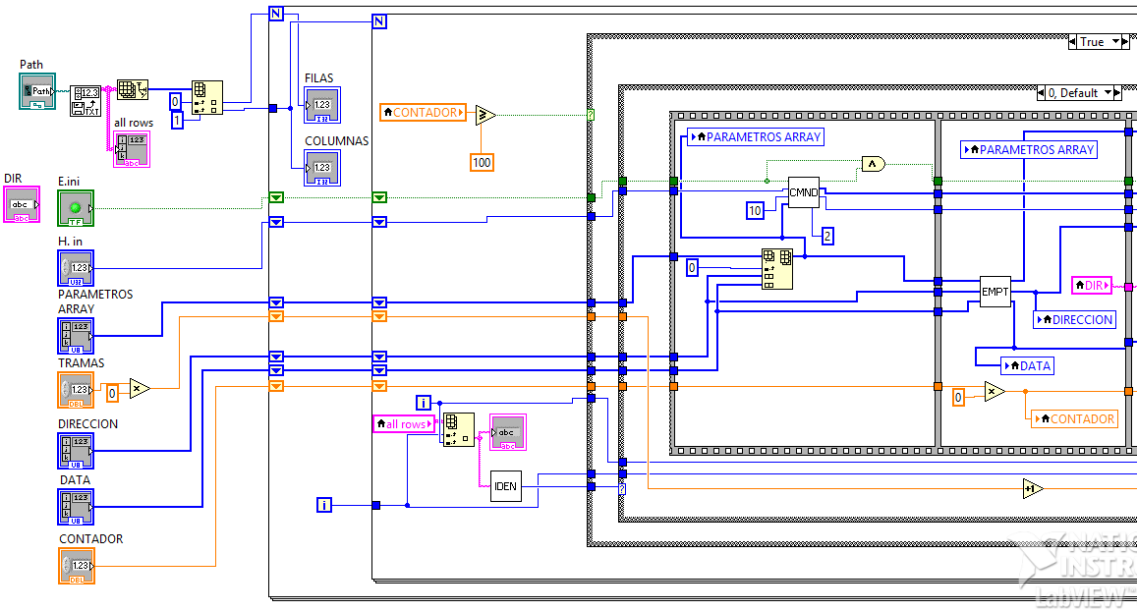

Figura 4.19: Estructura de la VI TXT (1)

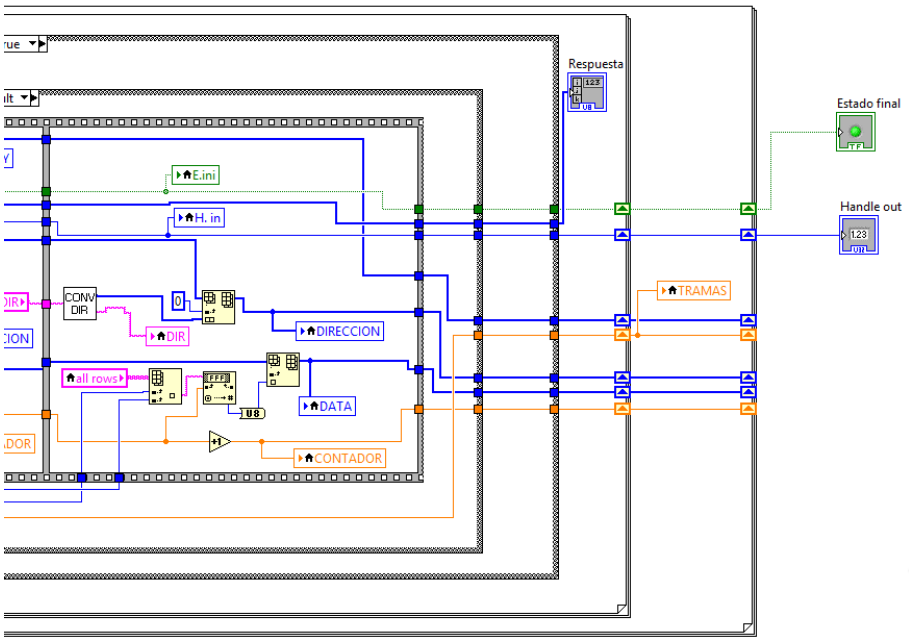

Figura 4.20: Estructura de la VI TXT (2)

Lo primero que realizará la aplicación será convertir el fichero de texto en una tabla de datos gracias a la función *Read text From Spreadsheet File*, introduciendo un dato del fichero de texto por celda. Después, medirá la cantidad de columnas y filas que tendrá que interpretar, para así poder recorrer todas las celdas de la tabla.

Una vez hecho esto, se dedicará a interpretar el contenido de cada una de las celdas. Para ello, se hará uso de una VI (IDEN) que identificará si el dato leído de la celda es un byte de dirección, de datos, un espacio o bien la letra q. En función de lo que identifique,

se procederá a ejecutar unas ciertas ordenes u otras. Todo byte de dirección o de datos leído será almacenado en dos vectores, uno para la dirección y otro para los datos. Estos dos vectores serán unidos más tarde para ser enviados como trama al microcontrolador.

Este programa se dedica a leer toda una fila entera, celda por celda, antes de saltar a la siguiente fila. Si la celda contiene un byte de dirección, esta se almacenará en el vector de dirección. Esta dirección corresponde a la dirección del primer byte de datos que le prosigue. Acto seguido, se deberán de leer el conjunto de bytes de datos que continúan la trama y se almacenarán en el vector de datos. En caso de llegar a una nueva dirección, tanto la dirección anterior como su vector de datos se enviarán al microcontrolador a través del comando RX Data Block. Después, se limpiarán ambos vectores (datos y dirección) a través de la VI EMPT para así poder almacenarlos con los datos que quedan por identificar y se recibirá la respuesta al comando por parte del Bootloader. También es importante mencionar que se enviará una trama al microcontrolador en caso de que el vector de datos contenga más de 100 bytes almacenados. Por último, en caso de que se lea una q, se enviará el vector de dirección y datos al microcontrolador, se recibirá respuesta del Bootloader y se finalizará la aplicación, habiendo así leído el archivo entero.

Una vez ya diseñada la VI TXT, ya se puede proceder a conectarla con el resto de VI necesarios en la aplicación principal. La estructura tendrá el siguiente aspecto:

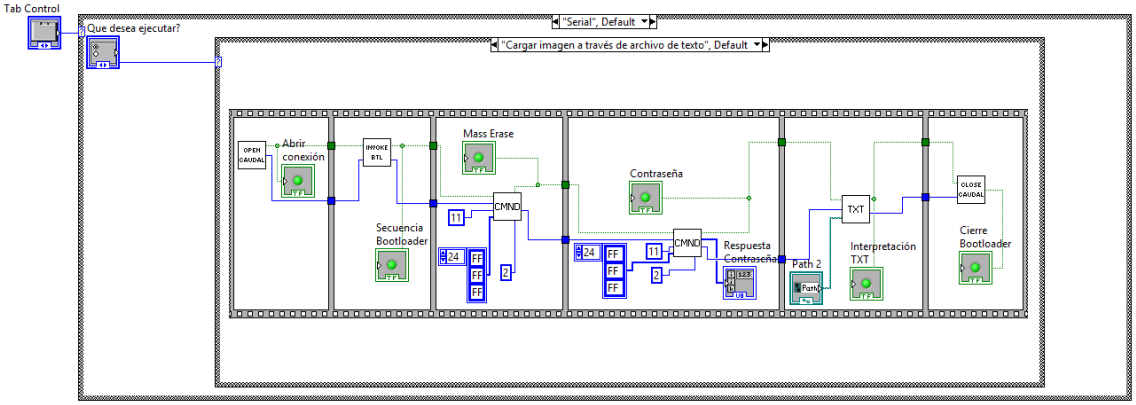

Figura 4.21: Estructura de la función de lectura de texto

La primera VI que se ejecutará será la de abrir conexión para así poder comunicarse con el dispositivo puente. A continuación, se realizará la secuencia de inicio del Bootloader para seguidamente poder enviar el comando de contraseña con la VI de comandos. La contraseña será enviada dos veces, la primera para limpiar la memoria y la siguiente para acceder al resto de comandos. Acto seguido se ejecutará la VI TXT, donde se interpretará el archivo de texto y se enviará al microcontrolador. Por último, a través de la aplicación de cerrar conexión se reseteará el microcontrolador y se cerrará la comunicación.

4.3.2. Comandos especiales del Bootloader

Esta otra funcionalidad nos permite escoger entre tres comandos a ejecutar tanto para la conexión serie y Bluetooth. Estos comandos son los siguientes: el comando RX Data Block con el que enviar bytes de forma manual, el comando BSL Version que permitirá

consultar la versión del Bootloader del microprocesador y el comando Mass Erase, que nos permitirá limpiar la memoria.

El aspecto de esta funcionalidad es el siguiente:

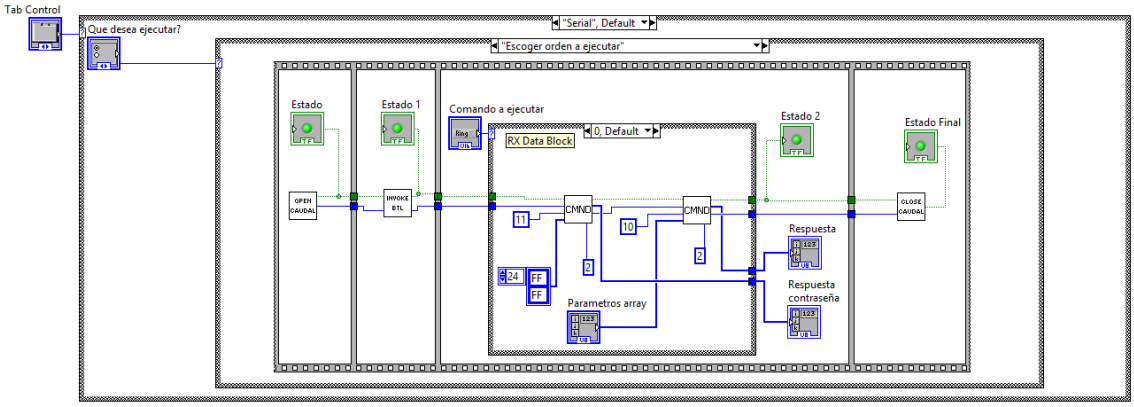

Figura 4.22: Estructura de la función de escoger orden para conexión serie

Como se ve en la anterior figura 4.22, primero se establecerá conexión con el dispositivo puente y después se accederá al Bootloader a través de la secuencia de invocación. A continuación, se ejecutará el comando que haya sido seleccionado por el usuario:

***RX Data Block*:** Este comando nos será útil en caso de que se quiera escribir un pequeño fragmento de código manualmente. Está formado por un comando de contraseña (11), seguido del comando *RX Data Block* (10).

***BSL Version*:** Esta opción la escogeremos en caso de querer consultar la versión de Bootloader del microcontrolador. Está formada por un comando para enviar la contraseña (11) y otro para consultar la versión (19). Una vez ejecutados nos devolverá la versión en forma de respuesta.

***Mass Erase*:** Por último, se nos dará opción de ejecutar una limpieza de memoria. Esto se realizará con el comando de contraseña (11), ya que al estar la memoria escrita, actuará como si fuese el comando Mass Erase.

Por último, se reseteará el microcontrolador y se procederá a cerrar la conexión.

4.4. Futuras aplicaciones

A pesar de haber desarrollado una aplicación que cumple ya con nuestras necesidades, son muchas más las funcionalidades que pueden ser implementadas. El ejemplo más claro es el de encriptar el fichero de texto con la información del firmware a actualizar, para así aumentar la seguridad y confidencialidad de los datos. De esta forma, podemos disminuir la posibilidad de que un agente externo pueda acceder a nuestro dispositivo y que plagie el código almacenado en la memoria o bien corromperlo. Para añadir esta

funcionalidad, será necesario descargar en la memoria FRAM un nuevo Bootloader que disponga de las funciones criptográficas, ya que el Bootloader original no dispone de ellas.

En este proyecto ya se trató de añadir esta funcionalidad, pero debido a las limitaciones de tiempo en la realización de este proyecto esta no pudo ser implementada. Es de vital importancia añadir una función como esta para asegurar la protección de los datos. Por ello, esta funcionalidad queda pendiente para futuros proyectos.

Por último, otra opción que podría ser implementada es la de aumentar el catálogo de dispositivos compatibles con la aplicación, añadiendo también algún otro dispositivo que haga uso de una conexión diferente y más efectiva, como puede ser la conexión WiFi.

CONCLUSIONES

Gracias a la investigación a través de diferentes documentos y tras todo el trabajo realizado, se ha podido crear una aplicación que cumple con el objetivo principal del proyecto. A pesar de ello, se ha desarrollado una aplicación que muestra ciertas carencias como por ejemplo en cuanto a la protección de datos se refiere. Estas carencias pueden ser de gran importancia y deben de ser solucionadas en posteriores trabajos. A continuación, se mostrarán las diferentes conclusiones que han podido ser obtenidas a lo largo del proyecto:

1. Texas Instrument ya ofrece una aplicación que sirve como interfaz de usuario para controlar el Bootloader. Desafortunadamente, esta aplicación ha sido desarrollada tan solo para ciertos dispositivos puentes que pertenecen a la misma compañía, que son el BSL Rocket y el MSP-FET. Estos dispositivos tienen un precio bastante elevado en comparación a muchos otros que podemos encontrar en el mercado, pudiendo ser un impedimento para el usuario cuando necesite actualizar el firmware. Es por ello que la solución final formada por la aplicación de LabVIEW, los módulos de comunicaciones y el kit MSP-EXP430FR6989 resulta ideal para controlar el Bootloader, permitiendo tanto la carga inicial de un primer firmware por el fabricante como su actualización por parte del usuario. En el primer caso evita que el fabricante acuda como suele ser de costumbre a un conector JTAG para cargar el firmware. A pesar de ello, en el segundo caso este sistema no ofrece al fabricante protección de los datos grabados en memoria ante posibles copias o ataques de algún agente externo, cosa que suele ser necesaria para los fabricantes. No obstante, esta solución será el primer paso para poder programar un nuevo Bootloader que ofrezca la solución a este problema, como por ejemplo la posibilidad de interpretar datos encriptados.

2. La interfaz de usuario creada a través de LabVIEW ha demostrado ser capaz de gestionar adecuadamente las señales de invocación y las comunicaciones con el Bootloader, cumpliendo las limitaciones en tiempo y los niveles de tensión indicados por Texas Instruments.

3. Para acceder al Bootloader, es necesario realizar una secuencia en los terminales RST y TEST del microcontrolador a través de los terminales de handshake RTS y DTR ubicados en el dispositivo puente que vaya a ser usado. Es necesario respetar el orden y los tiempos que se marcan para poder acceder al Bootloader.

4. Las comunicaciones entre el Bootloader y el usuario siguen un modelo conocido como maestro esclavo, donde el Bootloader permanece a la espera de recibir datos del usuario para poder actuar a través de comandos. Los comandos a los que responde el Bootloader son los de grabar bytes, borrar la memoria, transmitir bytes de la memoria, etc. Además, el Bootloader hace uso de la UART interna del microcontrolador para comunicarse via serie mediante los terminales TX y RX con los dispositivos puente.

5. Las tramas que forman el comando, tanto las de transmisión como las de respuesta por parte del Bootloader, a parte de incluir el código de comando y los paráme-

tros básicos para su ejecución, también incluyen un código para la verificación de errores por redundancia cíclica (CRC) que se encarga de verificar que los datos transmitidos y que los datos que han sido grabados en la memoria concuerden. En caso de que sean iguales el Bootloader responderá con una trama de confirmación (ACK).

6. Debido a la limitación de tiempo y a la poca documentación encontrada sobre como trabajar con datos encriptados, no ha sido posible añadir dicha funcionalidad en la aplicación. Para que nuestro dispositivo sea capaz de interpretar datos encriptados, primero es necesario actualizar el Bootloader del microcontrolador a una nueva versión que incorpore estas prestaciones. Consecuentemente, se ha dejado su desarrollo pendiente para posteriores trabajos.

BIBLIOGRAFÍA

[1] *SLAU550P. MSP430™ FRAM Devices Bootloader (BSL)*. (Texas Instruments Incorporated. Enero 2014).
Páginas: 4, 8

[2] *SLAU627A. MSP430FR6989 LaunchPad™ Development Kit (MSPEXP430FR6989)*. (Texas Instruments Incorporated. Mayo 2015).
Páginas: 16

[3] *SLAS789D. MSP430FR698X(1), MSP430FR598X(1) Mixed-Signal Microcontrollers*. (Texas Instruments Incorporated. Junio 2014).
Página: 17

[4] *FT232R USB UART IC Datasheet*. Versión 2.15 (Future Technology Devices International Limited. Abril 2019).
Página: 18

[5] *RN41/RN41N Class 1 Bluetooth® Module with EDR Support Datasheet*. (Microchip Technology Inc. Agosto 2014).
Página: 22

[6] *Bluetooth Data Module Command Reference & Advanced Information User's Guide*. Versión 1.0r (Roving Networks. Marzo 2013).
Página: 25

[7] *SLAU655F. Bootloader (BSL) Scripter*. (Texas Instruments Incorporated. Noviembre 2015).
Páginas: 29, 38

[8] *Basics of UART Communication*. (Circuit Basics)
http://www.circuitbasics.com/basics-uart-communication/

[9] Akshay Daga. *What is a Bootloader?*
https://www.engineersgarage.com/tutorials/bootloader-how-to-program-use-bootloader

[10] *EZ-FET lite*. (Texas Instruments Incorporated)
http://processors.wiki.ti.com/index.php/EZ-FET_lite

[11] *EnergyTrace Technology*. (Texas Instruments Incorporated)
http://www.ti.com/tool/ENERGYTRACE
Página

[12] *FT232R - USB UART IC*. (Future Technology Devices International Limited)
https://www.ftdichip.com/Products/ICs/FT232R.htm