UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN MECATRÓNICA

TEMA:

“GESTOR DE CARGA PARA MICROCONTROLADORES DE BAJO COSTO”

AUTOR: KEVIN PAUL ROJAS TERÁN

DIRECTOR: CARLOS XAVIER ROSERO

IBARRA-ECUADOR
2022

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1003988977		
APELLIDOS Y NOMBRES:	Rojas Terán Kevin Paul		
DIRECCIÓN:	Atuntaqui, barrio Las Palmas, Vicente Rocafuerte Junto a la casa comunal		
EMAIL:	kprojast@utn.edu.ec		
TELÉFONO FIJO:	2-906-133	**TELÉFONO MÓVIL:**	0995015706

DATOS DE LA OBRA	
TÍTULO:	Gestor de Carga para Microcontroladores de Bajo Costo
AUTOR :	Kevin Rojas
FECHA:	10/05/2021
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	PREGRADO
TITULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
ASESOR /DIRECTOR:	Carlos Xavier Rosero

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 06 días del mes de junio de 2022

EL AUTOR:

Kevin Rojas
El autor

UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CERTIFICACIÓN

En calidad de director del trabajo de grado "Gestor de Carga para Microcontroladores de Bajo Costo", presentado por el egresado Kevin Paul Rojas Terán, para optar por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra 06 dejunio de 2022

Carlos Xavier Rosero
DIRECTOR DE TESIS

Agradecimiento

”Amar la trama más que el desenlace”

J. Drexler

Dedicatoria

Dedico este trabajo con mucho amor a Dios y a mi familia que siempre me ha apoyado para poder alcanzar mis metas.

Kevin

Resumen

En la actualidad se encuentra dispositivos electrónicos casi en todo lo que rodea a los seres humanos. Están presentes en la industria, en la educación, en la salud, en el entretenimiento, en el transporte y hasta en el arte. Para la construcción de estos dispositivos se apunta cada vez más a la utilización de sistemas embebidos, los que llevan por núcleo a un microcontrolador o un microprocesador, y para que estos desempeñen sus funciones adecuadamente es necesario que manejen un gestor de carga. En este trabajo se desarrolla un gestor de carga para microcontroladores de bajo costo, con el fin de brindar a los usuarios de estos sistemas una herramienta práctica para trabajar eficientemente, ya que sin un gestor de carga el diseño y construcción de sistemas electrónicos se vuelve costoso, complicado y demoroso. Una vez terminado el desarrollo del gestor de carga se ha obtenido un sistema funcional que permite actualizar el firmware del microcontrolador sin necesidad de hardware de programación adicional. Se han realizado pruebas de funcionamiento, contrastando su desempeño con el de dos gestores disponibles, donde se ha determinado que el gestor de carga propuesto es el más pequeño de los que se han implementado, permitiendo a los usuarios de este sistema trabajar con más memoria libre para alojar un nuevo firmware.

Abstract

Nowadays, electronic devices are in almost everything that surround humans. They are in industry, in teaching, in health, entertainment, in transport systems and even in art. Embed systems are increasingly used to control these electronic systems which have a microcontroller or microprocessor based core and, for a correct working process of them is necessary to use a bootloader. In this project a bootloader for low-cost microprocessors is developed in order to give the electronic developers a practical tool which allows them work efficiently, as without a bootloader the development of electronic devices gets harder, slower and more expensive. When the bootloader development was finished it has obtained a functional system which allows upgrading the firmware in a microcontroller without extra programming hardware. Performance tests have been made, these tests compared the performance of the purposed bootloader with other bootloaders performance and then, the purposed bootloader has been established as the smallest bootloader between the tested bootloaders, leaving more extra free space for user firmware.

Índice general

Índice de Anexos

Índice de figuras

Introducción

Problema

Debido a la flexibilidad y a la gran cantidad de recursos de hardware que ofrecen los microcontroladores el diseño de aplicaciones en todos los ámbitos de la ingeniería se hace cada vez más popular utilizando dichos dispositivos [1]. Situándolos en ambientes tan diversos como el control de vehículos, viviendas inteligentes, dispositivos médicos, electrodomésticos entre muchas otras que acompañan el diario vivir del humano [2].

A causa de la importancia que han tomado los microcontroladores en el entorno, su estudio se ha vuelto fundamental en institutos de formación técnica y universidades en el área de mecatrónica, para lo que es indispensable actualizar constantemente la programación cargada en la memoria del microcontrolador por medio de hardware destinado únicamente a este propósito (dispositivo programador) [3].

Trabajar con dispositivos programadores se traduce en mayores costos al momento de de-

sarrollar aplicaciones con microcontroladores, y su utilización como tiempo extra de trabajo y disminución de la vida útil de los microcontroladores debido a que cada vez que se quiere actualizar la programación cargada en el microcontrolador es necesario desmontar el microcontrolador de su espacio de funcionamiento, montarlo en el programador y devolverlo a su lugar de funcionamiento [4].

Para evitar las desventajas mencionadas es necesario recurrir a pequeños programas conocidos como gestores de carga (bootloaders). Los gestores de carga permiten actualizar la programación de un microcontrolador sin necesidad de desmontarlo de su espacio de funcionamiento, sino que detectan cuando se envía una nueva programación por medio de los canales de comunicación del microcontrolador y la sobrescriben en su memoria [1].

Objetivos

Objetivo general

Desarrollar un gestor de carga para microntroladores de bajo costo.

Objetivos específicos

- Analizar gestores de carga disponibles en la ingeniería.
- Proponer un gestor de carga en base a la funcionalidad determinada en el análisis del estado del arte.
- Implementar el gestor de carga para la evaluación de su desempeño.

justificación

El desarrollo tecnológico en Ecuador es escaso, lo que ha traído consecuencias desfavorables para su desarrollo, dejándonos en una desventaja notoria en el mercado competitivo a nivel mundial [5], ubicando su competitividad local en el puesto setenta y seis según el Foro Económico Mundial [6].

Ecuador es un país que cuenta con una amplia diversidad de materia prima, pero lamentablemente es necesario adquirir tecnología extranjera para poder procesarla adecuadamente siendo

la maquinaria industrial y sus partes el producto más importado desde la Unión Europea [7].

Debido a la situación que atraviesa el país es importante poner atención a la formación de profesionales orientados a desarrollar sistemas tecnológicos para poder atender las necesidades tanto del sector público como privado. Gran parte de la demanda tecnológica ecuatoriana está relacionada a sistemas electrónicos [7] y consecuentemente a sistemas embebidos, donde los gestores de carga se han vuelto herramientas importantes para su desarrollo y aprendizaje.

Este trabajo pretende brindar al entorno una herramienta validada para el aprendizaje y desarrollo de aplicaciones electrónicas que se adapten a las necesidades del sector productivo ecuatoriano y poder entonces contribuir al desarrollo y nivel competitivo del país.

Alcance

En el presente trabajo se desarrollará e implementará un gestor de carga para microcontroladores de bajo costo usando lenguaje de programación de bajo nivel.

Se analizará el desempeño del gestor de carga propuesto y por lo menos dos gestores de carga existentes por medio de herramientas de software.

Capítulo 1

Revisión literaria

En este capítulo se explica de manera general los elementos que intervienen en el proceso de actualización del firmware de un microcontrolador y las tecnologías disponibles para realizar este procedimiento.

1.1. Microcontroladores

Los microcontroladores son dispositivos que, como su nombre lo indica, están destinados al procesamiento de información. Los microcontroladores reciben datos a través de sus puertos de entrada, luego estos son procesados a través de su unidad de procesamiento y finalmente la información procesada se transfiere a otros dispositivos por medio de sus puertos de salida.

1.2. Firmware

Según [8] se conoce como firmware al software de bajo nivel que puede acceder directamente al hardware de un dispositivo. Inicialmente el firmware de un microcontrolador debe ser escrito en un lenguaje de programación como: C, ensamblador, Micropython o cualquier otro disponible, todo esto en función de las necesidades y requerimientos del programador. Una vez que se ha escrito el firmware en un lenguaje de programación este debe ser procesado por un compilador o un ensamblador para obtener un archivo .HEX, como se muestra en la Fig. 1.1.

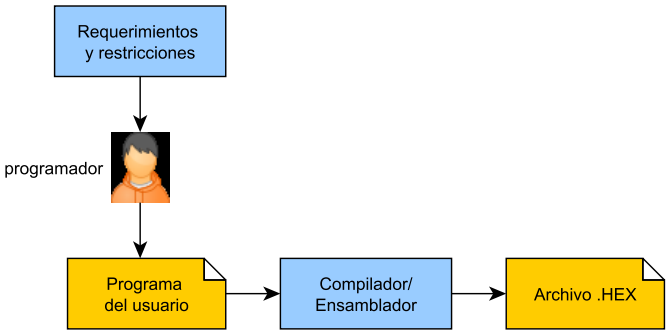


Figura 1.1: Proceso para desarrollo de Firmware

1.3. Archivos .HEX

El formato de archivos Hex corresponde a los archivos que resultan de ensamblar o compilar un código programado para un microcontrolador, y contienen información de las intrucciones, direcciones y operadores que se almacenarán en su memoria [9].

1.4. Dispositivos programadores

Son elementos de hardware que, por medio de un software, se encargan de transferir la información contenida en un archivo .HEX almacenado en la memoria de un computador a la memoria de programa del microcontrolador [10].

1.4.1. Programación con base ZIF

Algunos dispositivos programadores cuentan con una base donde se debe colocar el microcontrolador para realizar el proceso de grabación del nuevo firmware, como se muestra en la Fig. 1.2.

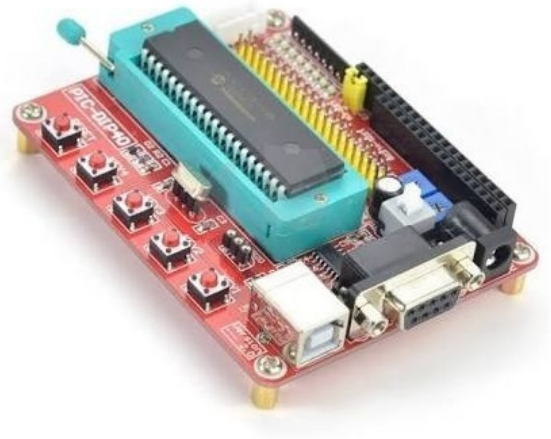


Figura 1.2: Dispositivo programador con base incluida

Para grabar un nuevo programa utilizando una base es necesario retirar el microcontrolador de su espacio habitual de funcionamiento, colocarlo en la base del dispositivo programador y luego devolverlo a su espacio de funcionamiento, lo que hace que el proceso de programación

sea complicado y demoroso. Además, de que al mover constantemente el microcontrolador de un lugar a otro es muy probable que este sufra daños físicos.

1.4.2. Programación ICSP

Como se ha mencionado anteriormente programar microcontroladores usando una base presenta varios inconvenientes que terminan entorpeciendo el proceso de programación. Para evitar estas dificultades existen dispositivos programadores que cuentan con tencología ICSP (In circuit serial programming), que permite al dispositivo grabador escribir datos en la memoria de programa del microcontrolador sin necesidad de retirarlo de su espacio de funcionamiento. Para esto el dispositivo programador usa cinco pines del microcontrolador, incluyendo los de energización [11].

1.5. Gestores de carga

Un gestor de carga es una herramienta de software que se almacena en el microcontrolador y que debe grabarse en su memoria por una única vez, utilizando un dispositivo programador. Una vez que el gestor de carga se encuentra en la memoria, este permite actualizar el firmware del microcontrolador sin la necesidad de un dispositivo programador, sino que lo hace a través de sus canales de comunicación. [1]

[1]Para que un microcontrolador sea capaz de funcionar con un gestor de carga es necesario que este cuente básicamente con tres caracerísticas: memoria suficiente para almacenar el gestor de carga, que sea capaz de recibir información desde otros dispositivos electrónicos y que tenga la función de reescribir su memoria por medio software.

En la Fig. 1.3 se puede observar de una manera muy general la lógica de funcionamiento de un gestor de carga, pero si se estudia más profundamente es posible entender que todos ellos están desarrollados de diferente manera y con distíntas características, las mismas que serán detalladas a continuación.

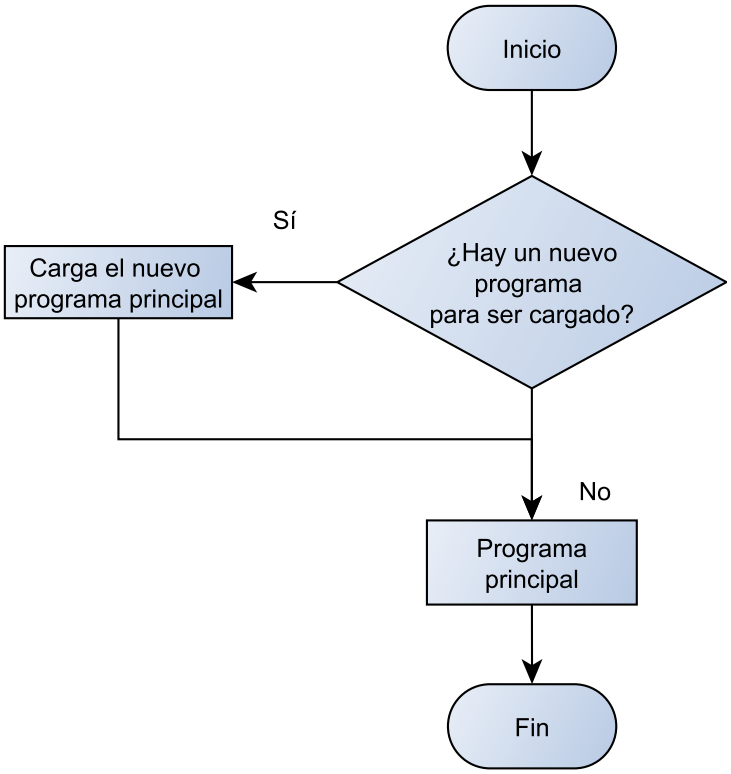


Figura 1.3: Lógica general de un Gestor de Carga

1.5.1. Familias y fabricantes de microcontroladores

En el mercado existen diferentes marcas de microcontroladores, cada una de ellas con su arquitectura y set de instrucciones particular. Consecuentemente los gestores de carga para mi-

crocontroladores de diferentes marcas y familias serán diferentes unos de otros. Entre los fabricantes más conocidos se puede encontrar nombres como: Intel, Microchip, Atmel (que ahora pertenece a Microchip), Motorola y ARM.

Cada fabricante ofrece una amplia gama de microcontroladores, donde se puede encontrar microcontroladores con espacios de memoria reducidos. Enfocándose en estos microcontroladores se han desarrollado Gestores de Carga simples y pequeños, como Tiny Bootloader, que ocupa solamente 107 palabras de memoria, generando un archivo .HEX de solo 654 bytes (para el microcontrolador Pic16f877a) [16], cuya lógica de funcionamiento se muestra en la Fig. 1.4. Por el otro lado están los microcontroladores que cuentan con espacios amplios de memoria y pueden almacenar gestores de carga más sofisticados y de mayor tamaño, como PICloader, que permite al usuario interactuar con el microcontrolador a través de una interfaz gráfica que se ejecuta en Windows o puede además proteger con una contraseña el código almacenado, llegando a generar un archivo .HEX de 11005 bytes [17].

1.5.2. Lenguajes de programación

Como se menciona anteriormente, cada fabricante de microcontroladores maneja su propio set de instrucciones, consecuentemente el lenguaje de programación para cada uno de estos es diferente, además es común encontrar distintos compiladores para un mismo fabricante.

Por todo esto existen gestores de carga escritos en distintos lenguajes de programación,

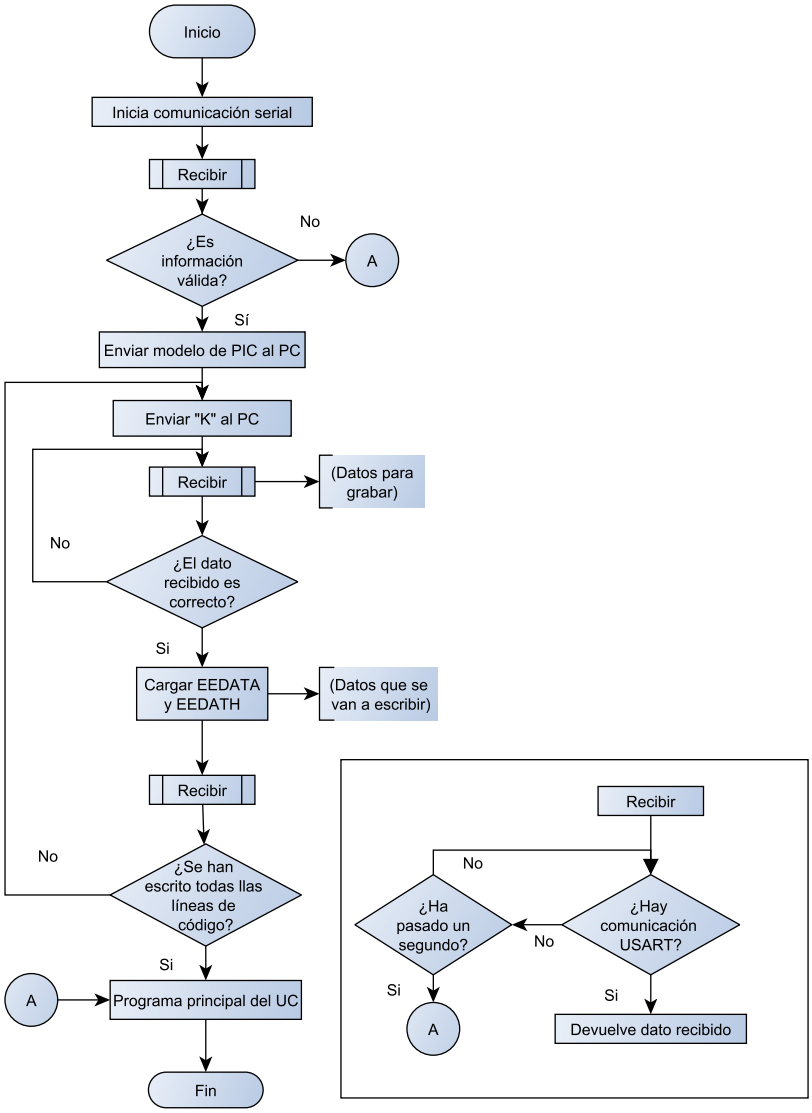


Figura 1.4: Lógica de Tiny bootloader

y aunque lo más habitual es encontrarlos en lenguaje ensamblador existen también algunos desarrollados en C, como el caso del gestor de carga que proporciona MicroC, que por estar desarrollado en C es un tanto más simple de desarrollar, lo cual se puede observar en la Fig. 1.5. Existen también propuestas más elaboradas que utilizan herramientas de más alto nivel, como python, y que pueden ser ejecutadas directamente desde un entorno de programación como Ec-

plipse. [12]

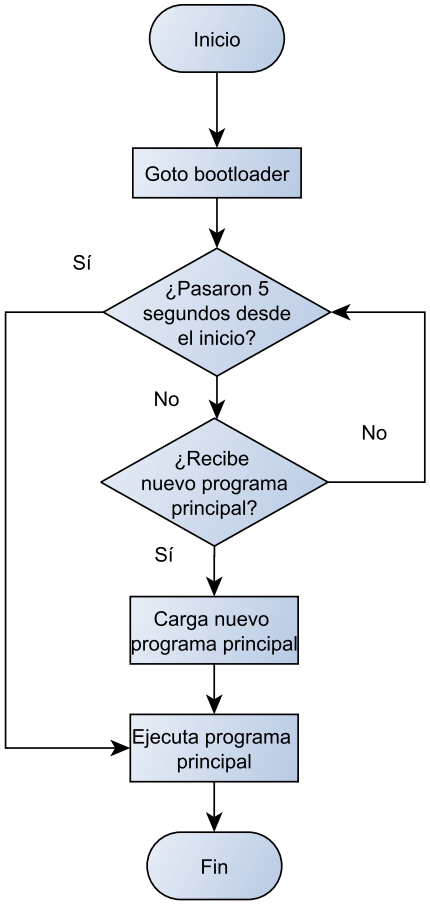


Figura 1.5: Lógica del Gestor de Carga de MicroC

1.5.3. Canales de comunicación

Para poder enviar grandes paquetes de información (Como es el caso de un firmware) desde un dispositivo electrónico hacia otro se suele usar canales de comunicación serial, es decir que los datos se envían uno tras otro, porque de lo contrario, si se enviasen todos a la vez, se necesitaría una gran cantidad de canales en paralelo, y por cuestiones de hardware esto no es posible.

Los avances tecnológicos han permitido que actualmente existan diferentes procedimientos para enviar información en formato serial. De aquí nacen los protocolos de comuncación, que no son sino reglas para que los dos (o más) dispositivos puedan entenderse correcta y eficientemente al momento de transferir información.

Los protocolos de comunicación más comunes en microcontroladores de bajo costo son: USART, I2C, SPI y USB.[2] Por esto los Gestores de Carga utilizan diferentes canales para cargar los archivos .HEX en la memoria del microcontrolador, como se indica en la tabla 1.1.

Cuadro 1.1: Gestores de carga disponibles y canales de comunicación utilizados

Gestor de carga	canales de comunicación
Tiny Bootloader	UART
Bootloader for STM32 with I2C	I2C
Bootloader for STM32 with SPI	SPI
Wloader	Puertos digitales
Pikme PIC Bootloader	Puerto digital
USB HID Bootloader	USB
USB DFU Bootloader	USB

1.5.4. Velocidad de transmisión

Como se ha mencionado, la comunicación serial consiste en enviar un dato después de otro, y para ello existen distintos protocolos de comunicación, que como se ha mencionado también, son parámetros y reglas que permiten un correcto intercambio de información entre dos o más

[2]Los puertos digitales no son propiamente un canal de comunicación serial, pero en algunos casos, a través de software, permiten al microcontrolador recibir información serial.

dispositivos. En muchas ocaciones estos parámetros pueden ser configurados por el usuario.

Un factor importante a tener en cuenta es la velocidad de transmisión de datos. La mayoría de Gestores de Carga disponibles permiten al usuario modíficar este parámetro a través de la recompilación y edición de su código fuente, pero hay que tener en cuenta que para esto es necesario tener conocimiento relativamente profundo sobre la arquitectura del microprocesador y en caso de hacerlo equivocadamente pueden presentarse errores de funcionamiento.

1.5.5. Ubicación del gestor de carga

Existen Gestores de Carga que por defecto están configurados para almacenarse en la parte alta (al final) de la memoria de programa del microcontrolador. Sin embargo, se pueden configurar para que se guarden en la parte baja. [18].

En la Fig. 1.6 se puede apreciar, al lado izquierdo, la distribución de memoria en un microcontrolador con un Gestor de Carga que se encuentra almacenado en la parte alta, y al lado derecho la distribución de memoria en caso de que el Gestor de Carga se encuentre en la parte baja de memoria.

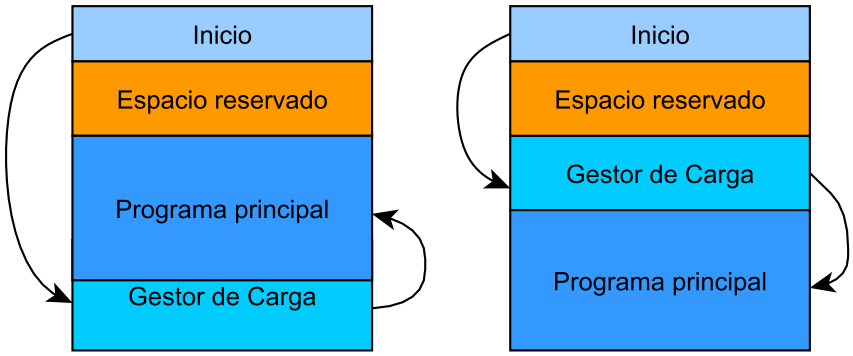


Figura 1.6: Gestor de Carga almacenado en la memoria alta y baja respectivamente.

1.5.6. Generador de gestores carga

La empresa Microchip Technology Inc. ha desarrollado Unified Bootloader Host Application, una herramienta de sofware que trabaja a través del plugin Code Configurator para MPLAB (el entorno de programación de Microchip). Esta herramienta de software permite generar un archivo .HEX de un gestor de carga y grabarlo directamente en la memoria del microcontrolador.

El generador de Gestores de Carga permite al usuario configurar el modelo de microcontrolador, el tipo de comunicación serial del Gestor (UART, USB o I2C) y varios parámetros de memoria, como se muestra en la Fig. 1.7. Los gestores de carga generados con esta herramienta detectan automáticamente la velocidad de transmisión serial, por lo que su configuración no es necesaria.

Desafortunadamente el plugin Code Configurator para MPLAB, en su versión 3.0.5, no pue-

de trabajar con varios de los modelos más conocidos de microcontroladores Microchip como el Pic16f877/A, Pic16f887, Pic18f4550, entre otros.

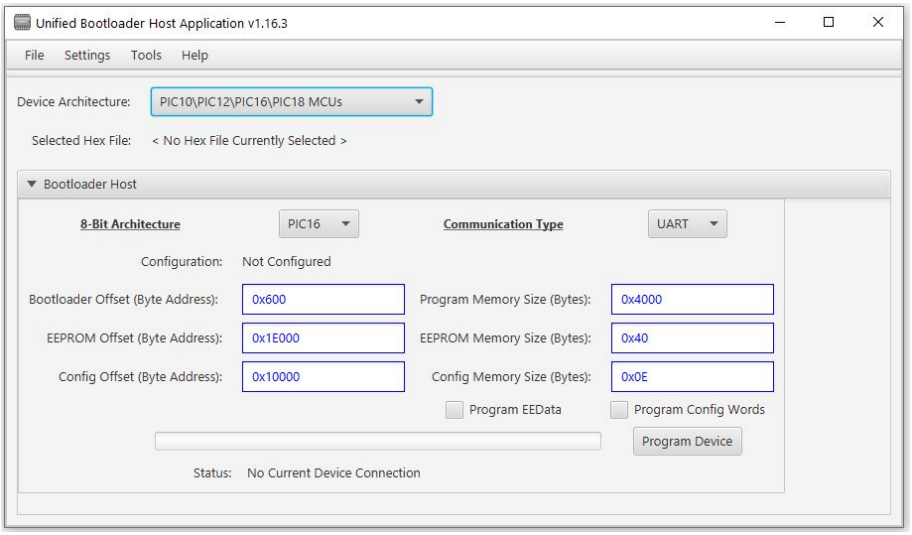


Figura 1.7: Interfaz gráfica Generador de Gestores de Carga.

Capítulo 2
Desarrollo

En base al análisis de información que se realizó en el capítulo anterior se ha propuesto un gestor de carga, cuyo proceso de desarrollo será explicado detalladamente en este capítulo. Para esto se ha dividido al gestor propuesto en diferentes subsistemas que a su vez se subdividen en bloques de funciones.

En la Fig. 2.1 se indica como el gestor de carga propuesto se divide principalmente en tres subsistemas: el firmware que se aloja en la memoria del microcontrolador, una interfaz gráfica que permite enviar el nuevo programa del usuario desde un computador y un canal de comunicación serial que permite comunicar el computador con el microcontrolador.

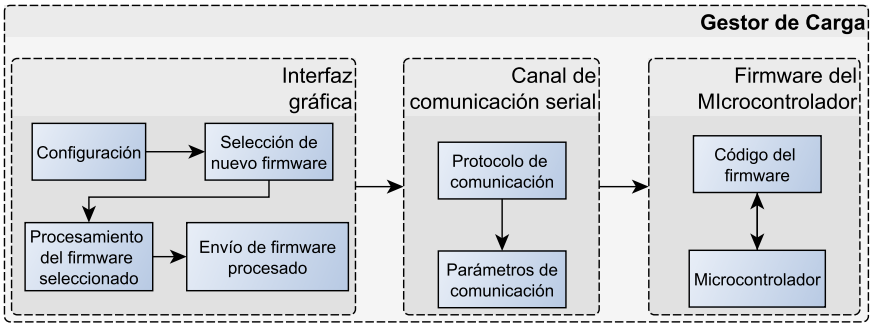


Figura 2.1: Estructura del gestor de carga propuesto.

2.1. Firmware del microcontrolador

Este subsistema consta de dos bloques, uno de hardware que corresponde al microcontrolador y uno de software que corresponde al código que será grabado en su memoria de programa.

2.1.1. El microcontrolador

Como se ha explicado anteriormente, para que un gestor de carga pueda ser implementado en un microcontrolador este debe tener memoria suficiente para alojar el gestor de carga, permitir la escritura en la memoria de programa por medio de software y contar con canales de comunicación. Adicionalmente, para el desarrollo del proyecto, se ha buscado que el microcontrolador sea de bajo costo y fácil de conseguir en el mercado.

En base a los requerimientos mencionados se ha decidido implementar el gestor de carga en el microcontrolador Pic16f877a de Microchip, que permite la escritura de datos en su memoria de programa en bloques de 4 direcciones consecutivas, cuenta con 8192 direcciones de memoria de 14 bits para almacenar el programa y canales de comunicación SPI, I2C, UART y USART que pueden ser utilizados para recibir el nuevo firmware [13].

Antes de comenzar con la programación del firmware del gestor de carga es necesario entender que los archivos .Hex que han sido generados para el Pic16f877a ordenan la información por líneas y los datos en cada línea se distribuyen como se muestra en la Fig. 2.2

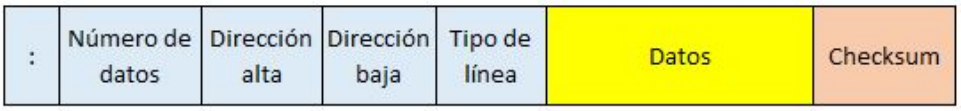


Figura 2.2: Distribución de datos en una línea de un archivo .HEX

Los dos puntos indican el inicio de una nueva línea, el número de datos indica cuantos datos se enviarán al microcontrolador, la dirección alta junto con la dirección baja se juntan para indicar la dirección de memoria donde se inicará la grabación, los datos son información de instrucciones y operadores que harán funcionar al microcontrolador y el checksum es un valor para verificar la correcta escritura de datos.

En la Fig. 2.3 se muestra un ejemplo real de lo explicado en el párrafo anterior. Para esto se ha tomado una línea de un archivo .HEX compilado para un Pic16f877a, que contiene la siguiente información: :1000000000308A000328830183138316063 09F0083 donde los datos están representados en notación hexadecimal y se agrupan de dos en dos. ND corresponde al número de datos, en este caso 10 en notación hexadecimal, que indica que el microcontrolador recibirá 16 datos de dos dígitos. DirA y DirB corresponden a la dirección alta y baja de memoria, que se juntan para formar una sola dirección, en este caso la dirección 0000 hexadecimal. TD corresponde al tipo de línea, el Pic16f877a entiende solo dos tipos de línea, las de grabación (00) que son todas las que tienen datos disponibles para grabar en la memoria de programa y las de finalización (01) que se hallan al final del archivo .HEX. Cada uno de los valores de DB se juntan con el siguiente valor de DA para formar un valor de 8 bits que se almacenará en la memoria del microcontrolador. Finalmente, Check muestra el valor que sirve para comprobar

si los datos han sido enviados exitosamente. [13]

:	ND	DirA	DirB	TD	DB	DA	DB	DA	DB	DA	DB	DA	DB	DA	DB	DA	DB	DA	DB	DA	Check
:	10	00	00	00	00	30	8A	00	03	28	83	01	83	13	83	16	06	30	9F	00	83

Figura 2.3: Ejemplo de distribución de datos en una línea de un archivo .HEX

La Fig. 2.4 indica como se organiza la información que se ha tomado para este ejemplo una vez que ha sido cargada en la memoria del microcontrolador.

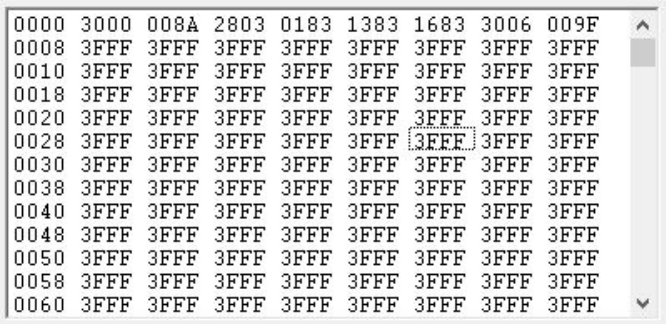


Figura 2.4: Datos del ejemplo almacenados en la memoria del microcontrolador.

2.1.2. Código del firmware

Para el desarrollo del firmware se ha decidido trabajar en lenguaje ensamblador, buscando reducir el tiempo de carga de archivos y los espacios ocupados en la memoria del programa. El código del firmware ha sido programado en el entorno de desarrollo MPLAB X en su versión 4.20 y para ensamblarlo se ha utilizado MPASM en su versión 5.77.

Mplab X permite trabajar con varios archivos en un mismo proyecto, por ejemplo se puede tener archivos de listas o de macros y llamarlos desde un archivo principal para que este sea más

pequeño y ordenado; sin embargo, para el desarrollo del gestor de carga se ha decidido colocar todo el código en un único archivo, con la finalidad de facilitar su análisis y modificación en caso de ser necesario.

Para evitar problemas en el direccionamiento y que el usuario modifique accidentalmente el código del gestor de carga se ha visto conveniente almacenarlo al final de la memoria del microcontrolador. Sabiendo que el gestor de carga ocupa 84 direcciones (palabras) de memoria y que el Pic16f877a cuenta con 8192 (de 0 a 1FFF en notación hexadecimal) direcciones disponibles para el programa; la memoria del microcontrolador luego de haber grabado el gestor de carga se organiza como se muestra en la Fig. 2.5

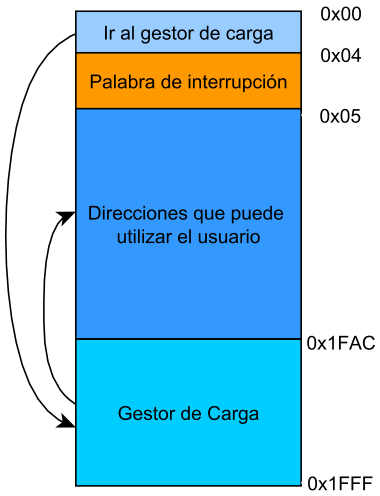


Figura 2.5: Distribución del gestor de carga en la memoria del microcontrolador

Como se muestra en la Fig. 2.5 las primeras cuatro direcciones de memoria contienen un salto al gestor de carga, por esto es importante que el usuario no programe instrucciones en

estas direcciones. Una vez que el microcontrolador ha saltado al gestor de carga este evalua si se encuentra o no activado el modo de carga.

El modo de carga del microcontrolador se activa poniendo la entrada RB7 del microcontrolador en estado lógico bajo. Si el modo de carga se encuentra desactivado el microcontrolador ejecutará el firmware programado por el usuario, caso contrario el gestor de carga activa la salida RB6 y a continuación espera recibir un nuevo firmware y actualiza la memoria del programa siguiendo la lógica que se muestra en la figura 2.6.

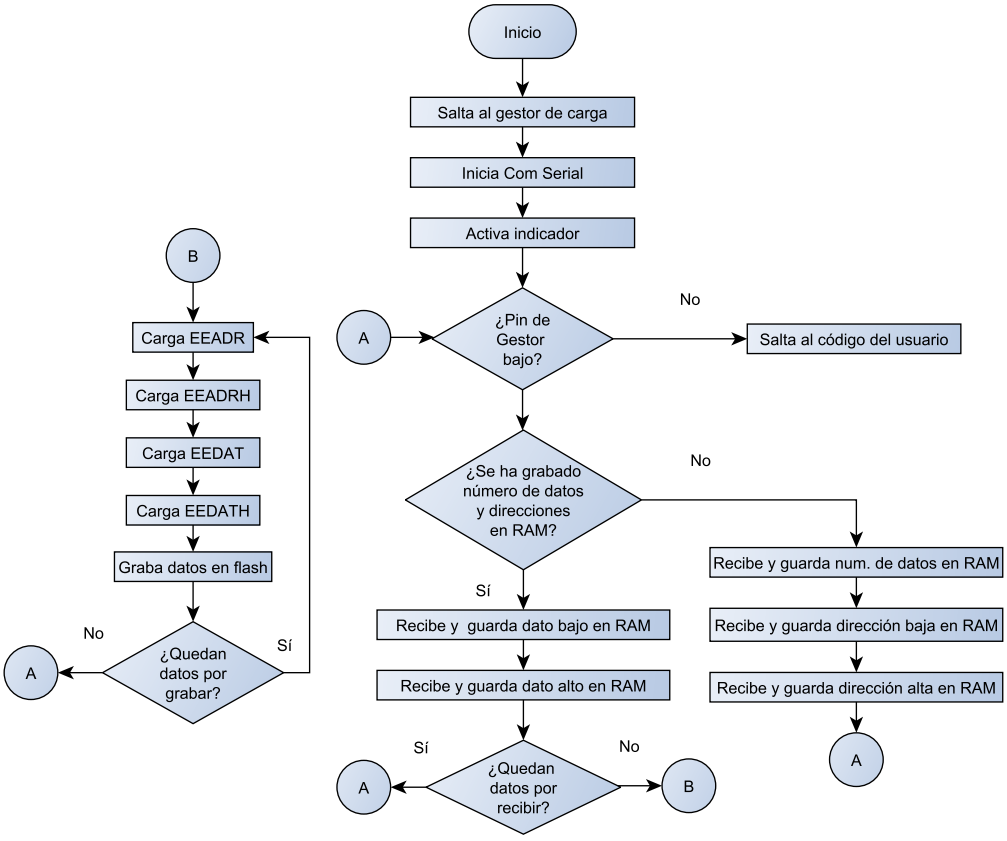


Figura 2.6: Lógica del firmware del gestor de carga

2.1.2.1. Recepción del nuevo firmware del usuario

El gestor de carga ha sido programado para recibir línea por línea la información de un archivo .HEX por medio de los pines RX y TX usando comunicación UART.

El primer dato recibido es interpretado como el número de datos que serán grabados en la memoria de programa y se almacena en la dirección 0x30 de la memoria RAM, los dos datos siguientes son interpretados como las direcciones baja y alta que se almacenan en las direcciones 0x31 y 0x32 de RAM respectivamente. Después de recibir el número de datos y las direcciones el microcontrolador continuará recibiendo los datos que se guardarán en la memoria del programa, hasta que haya determinado que se han recibido tantos datos como se indicó en un inicio.

Una vez que se han recibido todos los datos de la línea del .HEX (excepto el checksum) el microcontrolador procede a tomar los datos guardados en la memoria RAM para escribirlos en la memoria de programa.

2.1.3. Escritura en la memoria de programa

Para grabar los datos almacenados en la memoria RAM en la memoria del programa el microcontrolador toma los valores de 0x31 y 0x32 y los mueve a los registros EEADR y EEADRH respectivamente para indicar la dirección de memoria del programa que será escrita, luego se toma los dos valores siguientes de la memoria RAM y se almacenan en los registros EEDATA y EEDATH para formar el dato que se guardará en la memoria del programa, como se indica

en la Fig. 2.3. Una vez que se ha determinado la dirección y el valor de grabación se realiza un ciclo de escritura en la memoria del programa, siguiendo las indicaciones de [13].

Después de que se ha realizado un ciclo de grabación el microcontrolador se mueve a la dirección siguiente en la memoria del programa y toma los dos valores siguientes de la memoria RAM para guardarlos en esta dirección, hasta que se determine que todos los datos almacenados en la memoria RAM han sido ya grabados.

Una vez que se ha recibido y grabado la información de una línea del archivo . HEX el microcontrolador quedará siempre a la espera de una nueva línea, a menos que el usuario desactive el modo de carga, para indicar que ya se ha enviado toda la información del archivo, como se muestra en la Fig. 2.6.

2.2. Canal de comunicación serial

2.2.1. Protocolo de comunicación

Para que el usuario pueda enviar un nuevo firmware desde una computadora personal hasta el microcontrolador es necesario que los dos dispositivos manejen un mismo protocolo de comunicación. Como se ha indicado en el apartado 2.1.1 de este capítulo, el microcontrolador Pic16f877a cuenta con canales de comunicación: SPI, I2C, UART y USART; sin embargo, las computadoras personales no disponen directamente de estas tecnologías, sino que utilizan el

protocolo serial USB para comunicarse con otros dispositivos electrónicos. Se entiende entonces que por parte del computador se tiene una única opción, el protocolo USB, y que por parte del microcontrolador se puede escoger entre los protocolos SPI, I2C, UART y USART para recibir el nuevo firmware.

Sabiendo que la computadora y el microcontrolador no pueden comunicarse directamente es necesario utilizar una interfaz, que por un lado maneje el protocolo USB y por el otro uno de los protocolos admitidos por el microcontrolador. En el mercado local no ha sido posible encontrar interfaces USB a SPI, ni USB a I2C; pero si ha sido posible encontrar interfaces USB a UART, como el dispositivo FT232 que se muestra en la Fig. 2.7, que es con el que se trabajará para el desarrollo del gestor de carga, aunque podría hacerse uso de cualquier otra disponible.

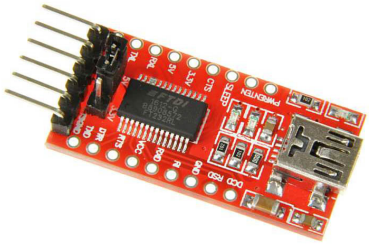


Figura 2.7: Interfaz USB a UART FT232

2.2.2. Parámetros de comunicaión

El protocolo de comunicación UART admite la configuración de varios parámetros como: la cantidad de bits que se transmiten en cada ciclo, el uso de bits de paridad, la velocidad de

transmisión de datos y el uso de bits de parada [14]; sin embargo, no todos los dispositivos o entornos de programación permiten la modificación de estos parámetros; por esto se ha decidido establecer la comunicación tan simple como sea posible, modificando unicamente la velocidad de transmisión de datos, que puede ser útil para los usuarios del gestor de carga. La tabla 2.1 muestra los estados y valores que se utilizarán para la comunicación serial entre la computadora y el microcontrolador.

Cuadro 2.1: Parámetros de la comunicación serial

Parámetro	Valor/estado
Bits de parada	0
Bis de transmisión	8
Bit de paridad	desactivado
Velocidad de transmisión (bauds)	Definido por el usuario

El usuario del gestor de carga puede utilizar los archivos .HEX del Anexo B que han sido compilados para trabajar con osciladores de 4, 8, 16 o 20 MHz y con velocidades de transmisión de 9600, 19600, 57600 y 115200 bauds. Si fuese necesario trabajar con una configuración diferente es necesario modificar la sección ”MODIFICAR”del Anexo A para usar el oscilador y velocidad de transmisión requeridas. Es necesario tener en cuenta que la velocidad de transmisión debe ser la misma tanto en el microcontrolador como en la interfaz para que la información sea transmitida correctamente.

2.3. Interfaz gráfica

Como se ha indicado en la Fig. 2.1 la interfaz del gestor de carga se encuentra dividida en cuatro bloques de funciones a los que se debe acceder desde el menú principal de la interfaz gráfica que se muestra en la FIg. 2.8.

Buscando que la interfaz del gestor de carga sea simple y amigable para el usuario se ha decidido tomar a Processing en su versión 3.5.4 como herramienta de desarrollo. Processing es un software multiplataforma, gratuito, de código abierto, potente y facil de usar, que cuenta con su propio lenguaje de programación basado en JAVA [15].

La interfaz trabaja sobre un espacio de trabajo de 600x400 pixels, donde se han creado botones utilizando figuras, colores y texto; con los que el usuario puede interactuar a través del mouse del computador [15]. Esta puede ejecutarse tanto en Windows como en Linux y de ser necesario modificarla su código completo, y totalmente comentado, se encuentra disponible en el Anexo A.

Figura 2.8: Menu principal de la interfaz

2.3.1. Configuración de la interfaz gráfica

Como se muestra en la Fig. 2.1 los bloques que conforman la interfaz gráfica del usuario se ejecutan uno tras otro, iniciando por el bloque de comunicación; caso contrario no se podrá ejecutar ninguna de las otras funciones y se mostrará un aviso como se ve en la Fig. 2.9.

Figura 2.9: Advertencia de puerto COM no seleccionado

Para el desarrollo de este bloque se utiliza la librería Serial de Processing, que crea un puerto de comunicación serial con el puerto COM y la velocidad de transmisión de datos que ha establecido el usuario.

2.3.1.1. Puerto COM

Cuando la computadora detecta que se ha conectado un dispositivo que utiliza comunicación serial esta crea un puerto de comunicación (Puerto COM) para identificarlo, por esto lo primero que el usuario debe hacer es indicarle a la computadora cual es el puerto COM correspondiente al microcontrolador. Al ejecutar la aplicación de la interfaz la computadora detecta automáticamente los puertos COM disponibles y el usuario debe seleccionar el puerto COM correspondiente al microcontrolador. Para esto se debe seleccionar la opción ”NoCOM”que se muestra en la Fig. 2.8.

2.3.1.2. Velocidad de transmisión de datos

Luego que se ha configurado el puerto COM correspondiente al microcontrolador es necesario determinar la velocidad de transmisión de datos, el valor por defecto es 9600 bauds, pero si el usuario necesita trabajar con valores diferentes debe seleccionar el botón que indica el número de bauds. Una vez hecho esto se mostrará la pantalla de la Fig. 2.10 donde se muestra un selector que ha sido programado para desplazarse desde 400 hasta 115200 en intervalos de 400 bauds.

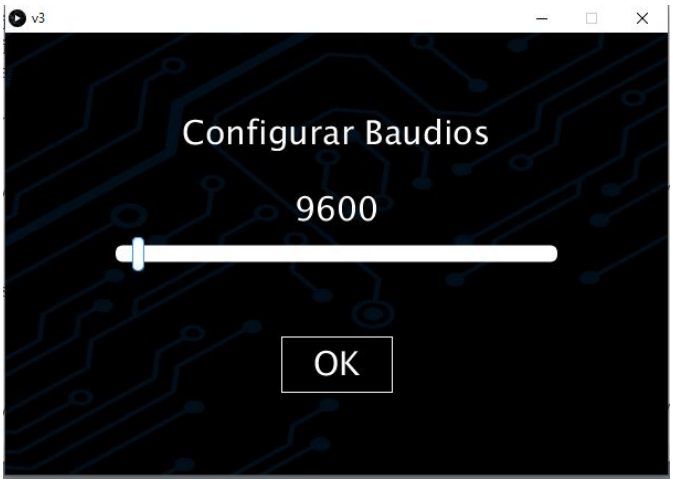


Figura 2.10: Seleccionar velocidad de transmisión

2.3.2. Selección del nuevo firmware

Para el desarrollo de este bloque se utiliza la funcionalidad de processing para leer archivos de texto plano, lo que quiere decir que no es estrictamente necesario que el nuevo firmware tenga formato .HEX, sino que puede estar contenido en cualquier otro formato de archivo de texto plano, como por ejemplo TXT.

Para la selección del nuevo firmware se debe primeramente configurar los parámetros de comunicación serial y luego se debe seleccionar la opción Cargar .HEX desde el menu principal, lo que abrirá el explorador de archivos para que el usuario seleccione el archivo de texto plano correspondiente al nuevo firmware.

2.3.3. Procesamiento del nuevo firmware

Cuando se ha seleccionado un archivo de nuevo firmware el programa evalua si se ha configurado anteriormente el canal de comunicación serial, si es así el programa descarta las líneas que nos son de grabación y agrupa la información en cuatro array, uno del número de datos, dos de direcciones y otro de datos, que luego los agrupa de tal modo que cada línea del array de datos contenga 16 elementos, tomándolos de las líneas siguientes, y si faltan elementos para completar la última línea esta se rellena con ceros; Todo esto siguiendo la lógica que se muestra en la Fig. 2.11.

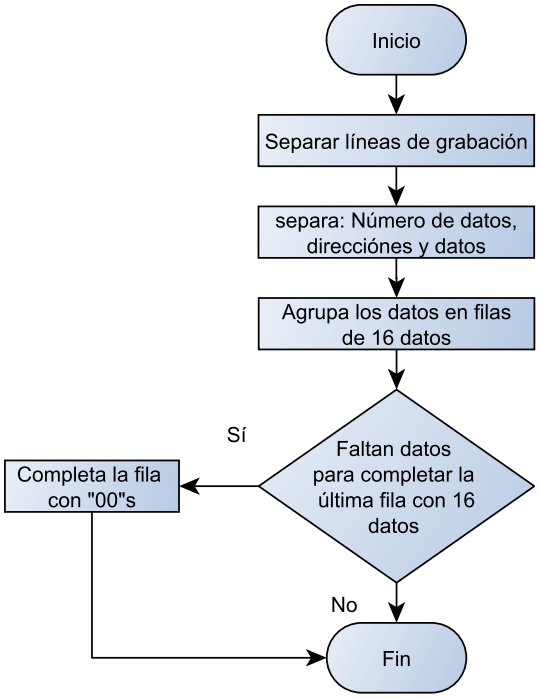


Figura 2.11: Lógica de procesamiento del nuevo firmware

2.3.4. Transmisión del nuevo firmware

Una vez que se tiene los array de tamaño de líneas, de direcciones y de datos ya agrupados estos se envían por el puerto COM que el programa ha creado en el bloque de configuración, siguiendo la secuencia indicada en la sección 2.1.2.1 de este capítulo.

Mientras se está enviando la información desde la computadora hasta el microcontrolador la interfaz gráfica ha sido programada para mostrar el estado actual, indicando que se está cargando la información. Cuando el programa ha pasado de grabar la última línea se cierra el puerto de comunicación y se muestra el estado ”Cargado” en la interfaz, como se muestra en la Fig. 2.12. Entonces el usuario puede desactivar el modo de carga del microcontrolador poniendo en alto el pin RB7 del microcontrolador, para que deje de esperar una nueva línea de información y pase a ejecutar el nuevo programa principal. como se indica en la Fig. 2.6.

Figura 2.12: Interfaz cuando se ha cargado el nuevo firmware

Capítulo 3

Implementación y Pruebas de Funcionamiento

3.1. Implementación

Para implementar el gestor de carga propuesto se utiliza el dispositivo programador que se muestra en la Fig. 3.1, desarrollado por la empresa Megatrónica en Ecuador. Este dispositivo permite la grabación y lectura de las memorias de programa y EEPROM del microcontrolador. Con él se puede comprobar que efectivamente, el gestor de carga se almacenará en la memoria de programa tal y como se muestra en la Fig. 2.5.

Figura 3.1: Dispositivo utilizado para implementar el gestor de carga .

Para guardar el firmware del gestor de carga en la memoria del microcontrolador no basta con tener un dispositivo programador; se debe contar además con un software compatible que gestione el intercambio de información entre la computadora y el microcontrolador, en este caso se ha utilizado el software de pickit 2, que se muestra en la Fig. 3.2.

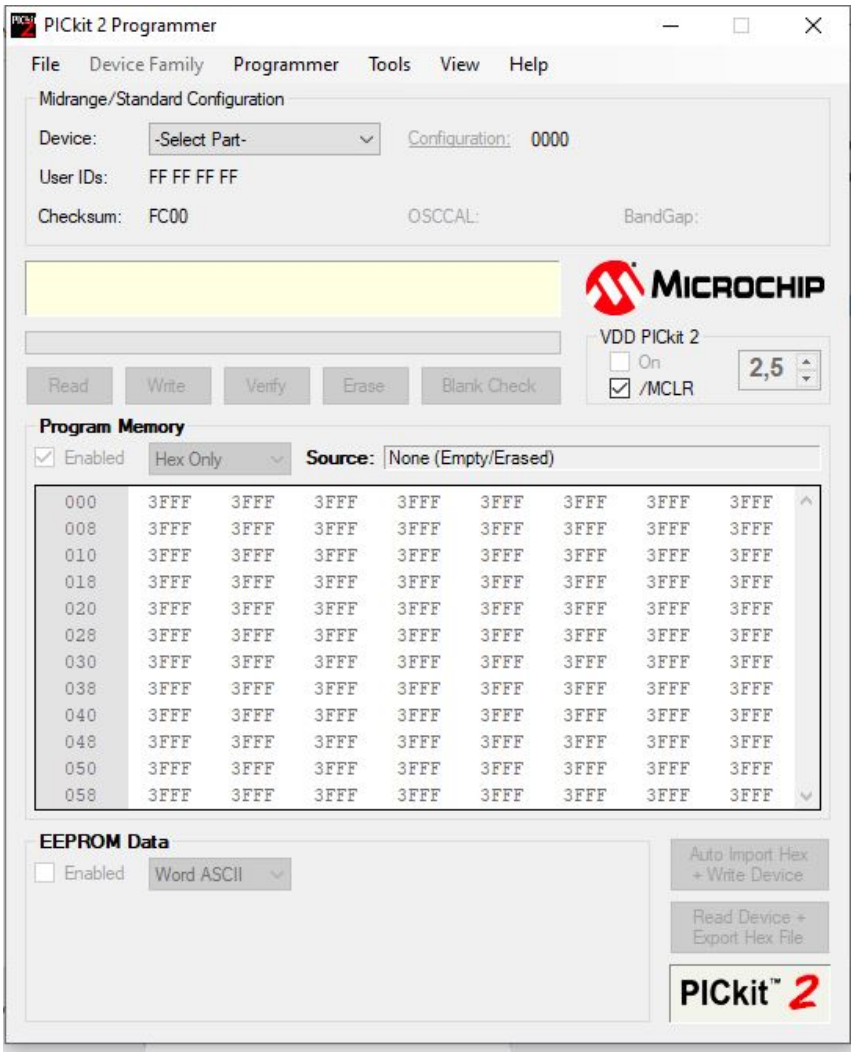


Figura 3.2: Interfaz de Pickit2

Una vez que el gestor de carga se encuentra grabado en la memoria del microcontrolador este ya puede recibir nuevo firmware desde la computadora por comunicación UART, a través de la interfaz FT232. Para esto se debe conectar el pin RX del microcontrolador con el TX de la interfaz, y el pin TX del microcontrolador con el RX de la interfaz. El microcontrolador puede energizarse desde la interfaz o desde una fuente externa, esos es opcional, pero sus pines de GND si deben conectarse entre si, como se muestra en la figura 3.3.

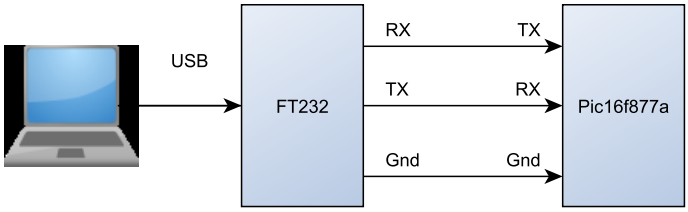


Figura 3.3: Conección de la computadora con el microcontrolador con FT232

3.1.1. Gestores de carga implementados

Buscando tener contraste en la estructura de los gestores de carga implementados se ha decidido trabajar adicionalmente con Tiny bootloader y con el gestor de carga del IDE MicroC, el primero está desarrollado en lenguaje ensamblador y el segundo en lenguaje C, ambos cuentan con versiones disponibles para el Pic16f877a y, al igual que el gestor de carga propuesto, utilizan una interfaz USB a UART.

3.1.1.1. Gestor de carga propuesto

Como ya se ha mencionado, el gestor de carga propuesto ocupa 84 palabras de memoria y se almacena al final de la memoria de programa, y ocupa las primeras cuatro direcciones para dirigirse hasta el gestor de carga. El microcontrolador se mantendrá en modo de carga siempre y cuando se ponga el pin RB7 en estado lógico bajo. Es decir, el modo de carga se activa y desactiva manualmente, este debe ser activado cuando se quiera cargar un nuevo firmware y debe desactivarse cuando la interfaz indique que ya se ha cargado el nuevo firmware. Cuando el modo de carga se encuentra desactivado el microcontrolador ejecutará inmediatamente el firmware del usuario, es decir no existe un tiempo de espera para iniciar a funcionar con normalidad.

3.1.1.2. Tiny Bootloader

Este gestor de carga está desarrollado en lenguaje ensamblador, cuenta con una interfaz gráfica que permite trabajar con cinco velocidades de transmisión diferentes, además aquí se puede monitorear el proceso de grabación del nuevo firmware en el microcontrolador y cuenta con varias seguridades, entre estas: evitar la escritura en las direcciones del gestor de carga, verificar que el nuevo firmware contenga un salto al inicio del programa, que el microcontrolador esté conectado, que la velocidad de transmisión de datos sea la correcta y que el microcontrolador se encuentre en modo de carga. El modo de carga de tiny bootloader se activa automáticamente el primer segundo de funcionamiento del microcontrolador, es decir siempre se tomará un segundo antes de ejecutar el firmware programado por el usuario y es en este instante cuando

debe enviarse el nuevo código desde el computador. Este gestor de carga ocupa cien palabras de memoria y, al igual que el gestor de carga propuesto, ocupa las cuatro primeras direcciones de memoria para saltar al gestor de carga.

3.1.1.3. Gestor de carga del IDE MicroC

Este gestor de carga ha sido desarrollado en lenguaje C y cuenta con su propia interfaz para cargar el nuevo firmware. La interfaz de este gestor de carga permite trabajar con quince velocidades de transmisión diferentes y verifica además que el microcontrolador se encuentre en modo de carga, el modo de carga se activa automáticamente durante los cinco primeros segundos de encendido del microcontrolador, esto facilita un el proceso de programación, porque el usuario tiene más tiempo para poder cargar el firmware; sin embargo, esto trae la desventaja de que el firmware programado por el usuario se demora siempre cinco segundos antes de comenzar a funcionar cuando se encienda el microcontrolador. Al estar programado en C este gestor de carga maneja ineficientemente la memoria, dejando espacios en blanco entre una y otra instrucción, se almacena al final de la memoria ocupando 687 espacios de la misma, incluyendo los cuatro espacios iniciales donde se guarda un salto al gestor de carga.

3.2. Pruebas a realizar

Para realizar las pruebas de funcionamiento del gestor de carga se han desarrollado cuatro archivos de firmware, que serán implementados en el gestor de carga propuesto y en dos gesto-

res estudiados en el Capítulo 1.

3.2.1. Firmware de prueba

Los archivos que se han utilizado para comprobar el funcionamiento de los gestores de carga implementados pueden descargarse desde el link que se encuentra en el Anexo B, se ha decido nombrarlos como Test1, Test2, Test3 y Test4. A continuación se explica las funciones y los recursos ocupados por cada uno de estos archivos de prueba.

- Test1: Blink de RB0, RB1 y RB2 del Pic16f877a programado en ensamblador.[1]
- Test2: Contador binario de 0 a 7 en el puero B del Pic16f877a programado en ensamblador.
- Test3: Blink alternado en RB0 y RB1 del Pic16f877a programado en C.
- Test4: Replica lo recibido por UART (con una interrupción) en el puerto B. La comunicación se hace a 9600 bauds con cristal de 4MHz.

En base a las características de estos cuatro archivos estaremos entonces evaluando la capacidad de losogestores de carga para:

- Grabar en la memoria de programa firmware escrito en lenguaje C.

[1]Es necesario recordar que si el código ha sido desarrollado en ensamblador el usuario debe colocar un salta hacia el inicio de programa en las cuatro primeras direcciones de memoria (de 0 a 3) para que los gestores de carga fucnionen correctamente.

- Grabar en la memoria de programa firmware escrito en lenguaje ensamblador.
- Grabar en la memoria de programa firmware que haga uso de la unidad de procesamiento del microcontrolador.
- Grabar en la memoria de programa firmware que utilice los puertos digitales del microcontrolador.
- Grabar en la memoria de programa firmware que utilice las interrupciones del microcontrolador.
- Grabar en la memoria de programa firmware que utilice los canales de comunicación del microcontrolador.

Las pruebas de funcionamiento han sido realizadas en simulación por software y también por aplicación en un modelo real, Para la simulación se han utilizado los arhivos que se adjuntan en el anexo B. Para la implementación real se ha utilizado el mismo esquema eléctrico, que una vez armado se muestra como en el anexo C.

3.3. Resultados

Una vez que se han implementado los cuatro firmware de prueba en los tres gestores de carga, tanto en simulación como en el modelo real se ha determinado lo expuesto en el cuadro 3.1, Se ha realizado además una comparativa de como los tres gestores de carga gestionan los recursos del microcontrolador después de que estos han sido implementados en la memoria de

programa. Basándose en la información del archivo Resultados del anexo B se ha obtenido la información expuesta en la Fig. 3.4.

Cuadro 3.1: Resultados de las pruebas realizadas

Parámetro	Tiny bootoader	Gestor de MicroC	Gestor propuesto
Manejo de firmware en C	Bien	Bien	Bien
Manejo de firmware en ensamblador	Bien	Bien	Bien
Manejo de puertos digitales	Bien	Bien	Bien
Manejo de interrupciones	Bien	Bien	Bien
Manejo de puerto COM	Bien	Bien	Bien

RECURSOS	Comparación cualitativa			Comparación cuantitativa			Comparación porcentual		
	Tiny Bootloader	Gestor de MicroC	Gestor Propuesto	Tiny Bootloader	Gestor de MicroC	Gestor Propuesto	Tiny Bootloader	Gestor de MicroC	Gestor Propuesto
Palabras en Memoria	100	687	84	100	687	87	98%	0%	100%
Tiempo de espera	1	5	0	1	5	0	80%	0%	100%
Verifica Comunicación	Sí	Sí	No	2	2	1	67%	67%	33%
Lenguaje de Programación	Assembly	C	Assembly	2	1	2	67%	33%	67%
Ubicación en memoria	Final	Final	Final	2	2	2	67%	67%	67%
Pines reservados	0	0	1	0	0	1	100%	100%	0%
Velocidades (Bauds) disponibles	5	15	288	5	15	288	0%	4%	100%
Protección de escritura de memoria	Sí	Sí	Sí	2	2	2	67%	67%	67%
Uso de interrupciones	Sí	Sí	Sí	2	2	2	67%	67%	67%
							68%	**45%**	**67%**

Figura 3.4: Desempeño de los gestores de carga implementados

3.4. Análisis de resultados

Basándose en lo expuesto en el cuadro 3.1 se evidencia que todos los gestores de carga implementados son exactamente iguales con respecto a su funcionalidad; todos son capaces de

manejar firmware escrito en distintos lenguajes, de usar los puertos digitales del microcontrolador, su unidad de procesamiento, los puertos de comunicación y sus interrupciones.

Por el contrario, si hablamos del desempeño de los gestores implementados no se puede decir definitivamente que un gestor de carga es mejor que otro, sino que el usuario debe seleccionar uno en base a sus necesidades, para ayudar al usuario a tomar esta decisión se ha realizado , primeramente una comparación cualitativa y luego cuantitativa del manejo de recursos por parte del microcontrolador, como se muestra en la Fig. 3.4.

Capítulo 4

Conclusiones y Trabajo Futuro

4.1. Conclusiones

- Generalmente el firmware de los gestores de carga está desarrollado en lenguaje de bajo nivel como ensamblador.

- Utilizando lenguaje de bajo nivel se ha programado el gestor de carga más pequeño disponible para microcontroladores de bajo costo, ocupando solamente 87 palabras de memoria.

- El gestor de carga propuesto es tan funcional como otros gestores disponibles, siendo capaz de almacenar nuevo firmware de la misma manera y sin inconvenientes.

- En base a los resultados obtenidos en la figura 3.4 se puede decir que de los gestores de carga implementados tiny bootloader es en general el que maneja de mejor manera los

recursos del microcontrolador.

4.2. Recomendaciones

- Antes de desarrollar un gestor de carga revisar investigaciones previas, para tener un punto de partida claro.
- Para desarrollar un gestor de carga se recomienda programar por bloques, tanto la interfaz como el firmware.
- Antes de implementar y trabajar con un gestor de carga se recomienda tener claro cuales son las necesidades del proyecto, para que en base a eso se pueda escoger el más adecuado.

4.3. Trabajo futuro

- Se propone, llevar el firmware a otros modelos y familias de microcontroladores.
- Además, en relación con la comunicación se podría sustituir la interfaz USB-UART por otros medios, como bluetooth o Wifi.
- Se propone también, programar la verificación de escritura de datos desde la interfaz gráfica.

Anexos

Anexo A

Firmware del gestor de carga

```
;-------------------------------------------------------------------------------
;       Archivo : Bootloader.
;       Fecha : 26/03/2021.
;       Autor : Kevin Rojas          kerojas95@gmail.com       kprojast@utn.edu.ec
;       Discpositivo: PIC16F877A.
;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
;       Descripción:
;       Si el estado lógico de RB7 es 0 después de un reinicio el Microcontrolador
;       entra en modo de Carga, esperando a que un nuevo firmware llegue a través
;       de comunicación UART. Mientras el microcontrolador se encuentra en modo de
;       carga RC6 y RB6 funcionan como salidas, mientras que RC7 y RB7 funcionan
;       como entradas. Después de recibir el nuevo firmware es necesario cambiar el
;       estado lógico de RB7 a 1, para que se ejecute el nuevo firmware.
;       Los nuevos vectores de RESET e INTERRUPCIÓN se encuentran en 0x03 y 0x04
;       respectivamente.
;       Toda la información del proyecto se encuentra disponible en:
;       https://github.com/KevinPaul1995/
;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
;      Hardware: FT232, Oscilador externo de 4000000.
;-------------------------------------------------------------------------------
```

```
;palabra de configuración
__CONFIG b'11111110110010'

;tipo de dispositivo
List p=16F877A ;Pic que se está usando
include "P16F877A.INC";Registros del PIC

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; MACROS ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
banco0          MACRO
                bcf STATUS,RP0
                bcf STATUS,RP1
ENDM

banco1          MACRO
                bsf STATUS,RP0
                bcf STATUS,RP1
ENDM

banco2          MACRO
                bcf STATUS,RP0
                bsf STATUS,RP1
ENDM

banco3          MACRO
                bsf STATUS,RP0
                bsf STATUS,RP1
ENDM

iniciobl        EQU 0x1FB0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; RESET ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 0x00 ;Cuando inicia el PIC
reinicio
PAGESEL configuracion
goto configuracion ;Ir a Inicio

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; PROGRAMA PRINCIPAL ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org iniciobl ;Cuando inicia el PIC
configuracion ; solo se ejecuta una vez
movlw b'10000000'
banco1
movwf TRISB ; TRIS B como salida, exepto b7
```

```
movlw b'10111111'
movwf TRISC ;RC7/Rx entrada, RC6/Tx salida
movlw b'00100100'
movwf TXSTA ;TX, sincrono, 8 bits, alta velocidad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; MODIFICAR;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;——————————————————————————————————————-
movlw .25 ;SPBRG=25D, (4000000/9600*16)-1
movwf SPBRG
;——————————————————————————————————————-

bcf STATUS,RP0 ;de banco1 a banco 0
movlw b'10010000'
movwf RCSTA ;USART en On, recepción contínua
movlw b'01000000' ; enciende indicador
movwf PORTB
MOVLW 0x30
MOVWF FSR ;empieza el direccionamiento en ram 30
bootloader
btfsc PORTB,7
goto $+3 ;si RB7=
goto recibir ;si RB7=0
goto bootloader
goto usuario ;bootloader off, va al goto del usuario
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; RECIBIR ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
recibir
btfss PIR1,RCIF ;test RX, si hay datos en el puerto
goto bootloader ;si esque no ha recibido regresa
movf RCREG,w ;Mueve lo del puerto a w
MOVWF 0x00 ;muevo a 0x0 o INDF, que es lo mismo

; revisar si está en el número de datos (30h)
movf FSR,W ;muevo la posición a w
sublw 30 ;le resto 30 a w
btfsc STATUS,2 ;si la bandera de cero es 0
goto cargatama
datos
INCF FSR,1 ;fsr++ para llenar los datos desde 30h
decfsz 30 ;si ya recibió todos los datos
goto bootloader ;mientras no se llenen todas las líneas
goto escribeflash
```

```
cargatama ;aumenta en dos INDF y mueve w a 20h
movf 0x00,W
movwf 0x20 ;para mantener guardado el tam de linea
incf 0x00 ;posiciones extra para la dirección
incf 0x00 ;posiciones extra para la dirección
incf 0x00 ;posiciones extra para la dirección
goto datos

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; ESCRIBE FLASH ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
escribeflash
;Apunta a los datos guardados en RAM
MOVLW 0x31
MOVWF FSR ;inicia FSR con 33h
movf INDF,W ;Lee el dato recibido

banco2
;Escribe dirección
movwf EEADR
incf FSR,1 ;FSR++
movf INDF,W ;Lee el dato recibido
movwf EEADRH

;escribe datos
escribedatos
banco2

;datos L
incf FSR,1 ;FSR++
movf INDF,W ;Lee el dato de RAM Low
movwf EEDATA ;datos LOW

;datos H
incf FSR,1 ;FSR++
movf INDF,W ;Lee el dato de RAM High
movwf 0x10E ;datos HIGH

;ciclo de escritura
BSF STATUS,RP0 ;banco3
BSF EECON1,EEPGD ;accesamos al bloque de memoria Flash
BSF EECON1,WREN ;habilita el ciclo de escritura
MOVLW 0x55 ;damos la secuencia de escritura
MOVWF EECON2
```

```
MOVLW 0xAA
MOVWF EECON2
BSF EECON1,WR ;inicia el ciclo de escritura
NOP
NOP
BCF EECON1,WREN ;deshabilita el ciclo de escritura
BCF STATUS,RP0 ;banco2
INCF EEADR,1 ;incrementa direccion de FLASH
clrf STATUS ;banco0
decf 0x20
decfsz 0x20 ;ya se escribió toda la linea?
goto escribedatos ;mientras no se termine
goto configuracion

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; FIRMWARE DEL USUARIO ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GOTO del Usuario
org iniciobl-4
usuario
nop
nop
nop
end ;Fin de Gestor de Carga
```

Anexo B

Enlace de descarga

Para el desarrollo del proyecto se ha realizado varios códigos de programación y simulaciones por computadora, que por motivos prácticos puede ser más conveniente descargarlos directamente que adjuntarlos como imágenes o como texto. Para esto se debe ingresar al siguiente enlace:

https://github.com/KevinPaul1995/Gestor-de-Carga-para-Microcontroladores-de-Bajo-Costo

De aquí se puede descargar:

El firmware del gestor de carga propuesto;

Archivos .HEX generados para funcionar con diferentes cristales y velocidades de transmisión;

Los códigos con los que se probó el funcionamiento de los gestores de carga implementados;

La simulación del gestor de carga propuesto;

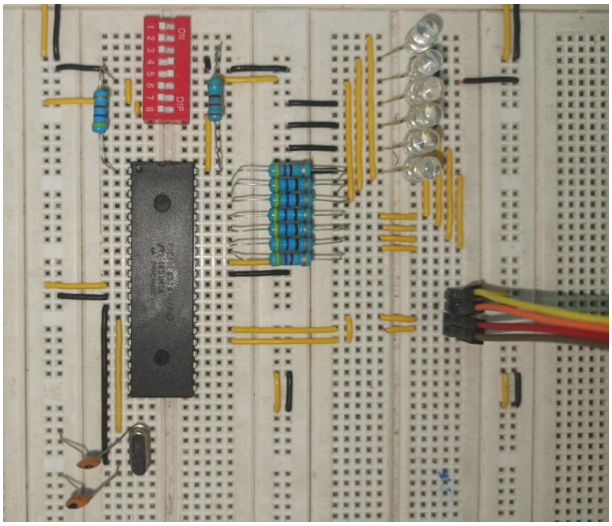
Un ejecutable de la interfaz para windows;

El código de la interfaz.

Anexo C

Circuito donde se han implementado las pruebas de funcionamiento

Este circuito no es más que los bits del puerto B del microcontrolador configurados como salidas digitales, todos exepto el último (RB7) que está configurado como entrada digital.



Bibliografía

[1] Francisco Villalobos “Boot Loader para Microcontroladores PIC serie 18 ”, Conciencia tecnológica , México 2006.

[2] Marcin Lewandowski “Dedicated AVR Bootloader for Performance Improvement of Prototyping Process ”, University of Silesia , Polonia 2017.

[3] Ricardo Guadrón Gutierrez “Implementación de Bootloaders en Microcontroladores PIC16 y PIC18 de Microchip Inc. ”, ITCA-FEPADE , El Salvador 2015.

[4] G. Navarroz “Control de un módulo bluetooth mediante microcontrolador ”, Universidad Politécnica de Cataluña , España 2005.

[5] C. Minalla “Avances tecnológicos en Ecuador”, Universidad Ecotec, Ecuador 2011.

[6] W. E. Forum “The Global Information Technology Report 2016”, Conell University, Nueva York, 2016.

[7] P. C. Sáenz “Informe mensual de comercio exterior octubre”, Ministerio de producción, comercio exterior, inversión y pesca, Ecuador, 2018.

[8] Sunha Ahn, Sharad Malik “Automated Firmware Testing using Firmware-Hardware Interaction Patterns ”, Princeton University , Estados Unidos de América 2014.

[9] Microchip Technology Inc. “MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User’s Guide”, Microchip Technology Incorporated, Estados Unidos de América 2013.

[10] Microchip Technology Inc. “MPLAB® PICkit™ 4 In-Circuit Debugger User’s Guide”, Microchip Technology Incorporated, Estados Unidos de América 2018.

[11] Microchip Technology Inc. “14/20-Pin, Low-Power, High-Performance Microcontroller with XLP Technology”, Microchip Technology Incorporated, Estados Unidos de América 2020.

[12] Carlos Xavier Rosero “Creating an open source platform to develop embedded control systems, and performing a control application on a real-time distributed system ”, Technical University of Catalonia , España 2015.

[13] Microchip Technology Inc. “Pic 16f87xa Data Sheet”, Microchip Technology Incorporated, Estados Unidos de América 2013.

[14] Texas instruments “KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) User Guide”, Texas instruments, Estados Unidos de América 2010.

[15] Casey Reas, Ben Fry. “Processing: A programming handbook for visual designers and artists ”, Massachusetts Institute of Technology, Estados Unidos de América 2007.

[16] Claudio Chiculita, “Tiny Bootloader 1.93”, Software, desde www.etc.ugal.ro/cchiculita/software/picbootloader.htm

[17] Rick Farmer, “PICloader”, Software, desde www.dontronics.com/rfarmer.html

[18] Ricardo Guadrón Gutiérrez,Juan José Guevara Vásquez, “Implementación de Bootloaders en Microcontroladores PIC16 y PIC18 de Microchip Inc.”, Revista Tecnológica Volumen 7, N 1, Escuela Especializada en Ingeniería ITCA-FEPADE.