UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERIA EN ELECTRONICA Y REDES DE COMUNICACIÓN

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TEMA:

"GESTOR DE ARRANQUE PARA UNIDADES DE MICRO CONTROL"

AUTOR: LUIS RUBÉN PADILLA PUETATE

DIRECTOR: CALOS XAVIER ROSERO CHANDI

IBARRA-ECUADOR

2020

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 DEL CONTACTO			
CÉDULA DE IDENTIDAD	100346916-8		
APELLIDOS Y NOMBRES	Luis Rubén Padilla Puetate		
DIRECCION:	Machala 13-06 y Del Colibrí		
EMAIL	lrpadilla@gmail.com		
TELÉFONO FIJO	062604060	**TELÉFONO MÓVIL**	0984539214
DATOS DE LA OBRA			
TÍTULO	GESTOR DE ARRANQUE PARA UNIDADES DE MICRO CONTROL		
AUTOR	Luis Rubén Padilla Puetate		
FECHA	13 de Enero del 2020		
PROGRAMA	[X] PREGRADO [ ] POSGRADO		
SOLO PARA TRABAJOS DE GRADO			
TÍTULO POR EL QUE OPTA	Ingeniería en Electrónica y Redes de Comunicación		
DIRECTOR	Carlos Xavier Rosero C.		

2. CONSTANCIA

El Autor manifiesta que la obra objeto de la presente Autorización es original y se la desarrollo, 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 13 días del mes de Enero del 2020

EL AUTOR:

Luis Rubén Padilla P.
C.I. 100346916-8

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

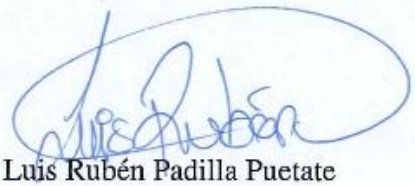
CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO

A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, Luis Rubén Padilla Puetate con cédula de identidad Nro. 1003469168, manifiesto mi voluntad de ceder a la Universidad Ténica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autor (es) de la obra o trabajo de grado denominado "GESTOR DE ARRANQUE PARA UNIDADES DE MICRO CONTROL", que ha sido desarrollado para optar por el título de Ingeniero en Electrónica y Redes de Comunicación, en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Ibarra, 13 de Enero del 2020



Luis Rubén Padilla Puetate

C.I.: 1003469168

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

En calidad de director del trabajo de grado "GESTOR DE ARRANQUE PARA UNIDADES DE MICRO CONTROL ", presentado por el egresado LUIS RUBÉN PADILLA PUETATE, para optar por el título de Ingeniero en Electrónica y Redes de Comunicación, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 13 de Enero del 2020

Carlos Xavier Rosero
DIRECTOR DE TESIS

Índice general

Índice de figuras

Índice de cuadros

Resumen

Debido al costo elevado de programadores para Unidades de Micro Control (MCU) hace que la disponibilidad de esta herramienta de hardware para laboratorios de electrónica en universidades públicas sea limitada. Para lo cual el presente trabajo se enfoca en la creación de un gestor de arranque que pueda servir como una herramienta alternativa de software para la lectura, borrado y escritura en la memoria de la MCU mediante comunicación serial y apoyada por una interfaz humano máquina (MHI). Se usa la metodología Botton-Up, generalmente implementada para despliegue de software, la cual es adaptada en algunos de sus niveles de desarrollo que permite desde el planteamiento de requerimientos y funcionalidades del gestor de arranque, comunicación serial e MHI hasta llegar a la obtención de un producto final.

El gestor de arranque es colocado en una sección de memoria de un procesador digital de señales incorporado en una tarjeta de desarrollo para aplicaciones integradas FLEX-Full de la empresa EVIDENCE. Dicha tarjeta es utilizada para desarrollos de control en tiempo real y se basa en un dsPIC33FJ256MC710 y un PIC18F2550, ambos sin firmware de fábrica. Se desarrolla un firmware para la MCU secundaria (PIC18F) que actúe como puente de comunicaciones *UART/USB* entre el dsPIC y un ordenador que contenga la MHI. Para ello se realizan configuraciones (conexiones) previas sobre la tarjeta FLEX-Full.

La MHI se desarrolla para sistemas operativos GNU/LINUX bajo el lenguaje de programación Python que se ejecuta por interfaz de línea de comandos. Esta interfaz usa librerías para efectuar la comunicación *USB/UART* y la decodificación de archivos ejecutables de aplicaciones de usuario. Dichos archivos ejecutables son generados por el editor MPLAB IDE con formatos *.elf* y *.hex*, comúnmente usados por grabadores convencionales para diferentes MCUs.

En efecto, la MHI una vez que decodifica el archivo, éste es enviado a través del puerto USB del ordenador para que sea recibido y administrado por el gestor de arranque previamente cargado por única vez en el dsPIC, para ser colocado en una sección de su memoria flash. Este conjunto de desarrollos permite reemplazar el hardware externo de programación convencional utilizado para dicho fin, para lo cual se realiza pruebas de laboratorio con la programación de aplicaciones comúnmente desarrolladas en este ambiente.

Finalmente se respalda dichas pruebas con resultados obtenidos a partir de la realización de un taller a estudiantes involucrados en el tema de niveles entre 5to a 7mo de la carrera de Ingeniería en Electrónica y Redes de Comunicación con una posterior encuesta. Todo esto permite concluir que esta alternativa de programación pueda cubrir la problemática que se plantea pudiendo perfeccionarse y extenderse para otras MCUs con nuevas características y funcionalidades.

Abstract

Due to the high cost of MCU programmers, its availability for laboratories in public universities is limited. This work focuses on the creation of a bootloader as an alternative software tool for reading, deleting and writing in the memory of the MCU through serial communication, supported by a human machine interface (HMI). The Botton-Up methodology is used, for software deployment, which is adapted in some of its developmental levels that allows from the approach of requirements and functionalities of the bootloader, serial communication and HMI until reaching a final product.

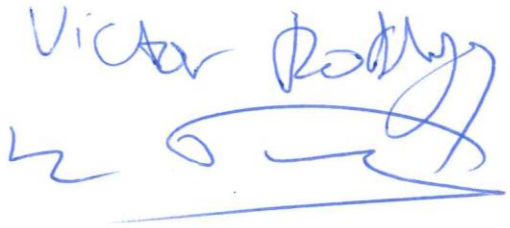
The bootloader is placed in a memory section of a digital signal processor incorporated in a development card for integrated applications FLEX-Full from the company EVIDENCE. This card is used for real-time control developments and is based on a dsPIC33FJ256MC710 and a PIC18F2550, both without factory firmware. Firmware is developed for the secondary MCU (PIC18F) that acts as a UART/USB communications bridge between the dsPIC and a computer containing the HMI. For this, previous configurations (connections) are made on the FLEX-Full card.

The HMI is developed for GNU/LINUX operating systems under the Python programming language that is executed by command line interface. This interface uses libraries to per-

form USB/UART communication and decoding executable files of user applications. These executable files are generated by the MPLAB IDE editor with .elf and .hex formats, commonly used by conventional recorders for different MCUs.

In fact, once the file is decoded by the HMI, it is sent through the USB port of the computer to be received and managed by the bootloader previously loaded only once in the dsPIC, to be placed in a section of its Flash memory. This set of developments allows replacing the external hardware of conventional programming used for this purpose, for which laboratory tests are carried out with the programming of applications developed in this environment.

Finally, these tests are supported with results obtained from the realization of a workshop for students involved in the topic of levels between 5th and 7th of the Electronics and Communication Networks Engineering major with a survey. All this allows us to conclude that this alternative programming covers the problem that arises and can be perfected and extended to other MCUs with new features and functionalities.



Capítulo 1

Antecedentes

El presente trabajo tiene la finalidad de beneficiar en la preparación profesional de los estudiantes que estén cursando el proceso de formación académica de ingeniería en el área de electrónica, los que podrán tener acceso a una herramienta adicional de software para la programación de una MCU.

1.1. Problema

En la actualidad las MCUs tienen un área de aplicación muy amplia como la regulación de velocidad en máquinas, control de motores, apertura y cierre automático de puertas, procesamiento de datos obtenidos de sensores, en general, en las áreas de automatización, control y robótica (Cucho, Orihuela, Sánchez y Rodríguez, 2006). Las soluciones tecnológicas propietarias y sus elevados costos, aunque proveen del software (compiladores) y hardware (programadores) necesarios, como plataforma de desarrollo de sistemas electrónicos aplicados, no hacen más que encajar y limitar al usuario a un solo fabricante (González y Moreno, 2004). Esta situación se agrava en entornos de laboratorio de universidades públicas en donde los re-

cursos económicos son limitados y el estudiante debe asumir la inversión, lo que puede tener un impacto directo en su economía y a su vez en su desempeño académico.

Villalobos, Rodríguez, Molina y Trejo (2006) refieren que una de las principales causas para que una MCU tenga algún desperfecto se debe al proceso manual a la cual es sometida, que consiste en insertar o sacar de una placa de experimentación para ser trasladada a un hardware externo para que pueda ser grabado el archivo ejecutable en la memoria de la MCU, convirtiéndose en un proceso repetitivo hasta terminar con el objetivo de la aplicación. Estas tareas deben ser llevadas con mucha cautela para preservar la integridad física de la MCU, situación que podría empeorar en ambientes de laboratorio, ya que generalmente se desarrolla varias prácticas para una misma clase. Guadron y Guevara (2010) mencionan que la utilización de hardware externo para grabado en las MCUs es un problema en la enseñanza de estos sistemas, ya que si bien una institución puede adquirir programadores, estos no siempre estarían disponibles para todos los estudiantes ya sea por limitaciones de tiempo o para no acortar la vida útil de los mismos.

Una técnica avanzada para grabar archivos ejecutables en la memoria de las MCUs es con el uso de gestores de arranque “bootloaders”, los cuales gestionan de forma automática el proceso de grabado en la memoria con la ventaja de permitir realizar actualizaciones del archivo ejecutable sin necesidad de removerla desde el circuito o placa de práctica (Torres, 2007). En las aulas de laboratorio de microcontroladores los estudiantes que se encuentran en un proceso de aprendizaje pueden experimentar inconvenientes con el armado de la práctica debido a la cantidad de elementos que podrían conformar la misma, lo que restaría tiempo del estudiante para completar el objetivo de desarrollo.

La falta de disponibilidad o acceso a los programadores podría convertirse en una debilidad considerable pudiendo lograr que el desempeño del estudiante baje. Con el desarrollo de un gestor de arranque se podrá ayudar a que en los ambientes de laboratorio de sistemas microprocesados pueda existir una herramienta complementaria para el grabado de archivos ejecutables sin necesidad de usar hardware externo, reduciendo costos y tiempos.

1.2. Objetivos

1.2.1. Objetivo Principal

Desarrollar un gestor de arranque que permita la lectura, borrado y escritura en la memoria de la MCU mediante comunicación serial con ayuda de una interfaz humano máquina.

1.2.2. Objetivos Específicos

- Determinar los requerimientos y funcionalidades de la solución.
- Desarrollar el gestor de arranque para la MCU considerando requerimientos de software y hardware.
- Crear una interfaz humano máquina para GNU/LINUX que coloque archivos ejecutables en la memoria de la MCU a través de comunicación serial.
- Realizar pruebas de funcionamiento del gestor de arranque de la MCU y de la interfaz humano máquina.

1.3. Alcance

Este trabajo está enfocado en el desarrollo de un gestor de arranque que se constituirá como una herramienta adicional para ambientes de laboratorio de sistemas microprocesados, permitiendo reemplazar el hardware externo para el grabado en las MCUs por un gestor de arranque y una herramienta de software, que conjuntamente permitirán realizar la acción de lectura, escritura y borrado en la memoria del mismo. Este trabajo constará esencialmente de dos MCUs, el primero será usado como puente entre el puerto USB del computador y el puerto serial del MCU principal de manera transparente para la interfaz humano máquina (MHI). El segundo (MCU principal) será el controlador usado para las aplicaciones del usuario, sobre el cual se colocará el gestor de arranque.

Inicialmente se recopilará información de las tecnologías, dispositivos y elementos que formarán parte del desarrollo para el gestor de arranque, permitiendo de esta forma obtener argumentos válidos para la fundamentación teórica de este proyecto. Seguidamente se determinará los requerimientos y funcionalidades para la solución que serán concebidas en diagramas de bloques y flujo lo que permitirá dividir en subsecciones que ayudarán a obtener una secuencia de planificación y realización del proyecto con el objetivo de llegar una solución viable.

Para el desarrollo inicial en base a los requerimientos tanto de hardware como de software se realizará el gestor de arranque que permitirá efectuar la gestión (lectura, escritura, borrado) de la memoria de la MCU principal y paralelamente podrá manejar la comunicación serial con la MCU puente. El gestor de arranque se encontrará en una región de memoria que es dedicada solamente para este pequeño programa el cual no podrá solaparse con las aplicaciones de usuario que serán colocadas en la memoria FLASH de la MCU.

Posteriormente se creará una herramienta de software (MHI) para sistemas operativos GNU/LINUX bajo el lenguaje de programación Python la cual será usada como interfaz de línea de comandos (CLI). Dicha herramienta se encargará de la decodificación del archivo ejecutable generado por el editor MPLAB IDE para que posteriormente sea enviado a través de un puerto de comunicación del ordenado y colocado en la memoria de la MCU principal.

Finalmente se realizarán las pruebas de funcionamiento de manera íntegra de las partes descritas, una vez que se concluya dichas pruebas se procederá a grabar un ejemplo de aplicación (archivo ejecutable) en la memoria Flash de la MCU principal haciendo uso de la MHI lo que permitirá verificar el funcionamiento integral de la solución para la problemática planteada.

1.4. Justificación

El presente trabajo tiene la finalidad de ayudar en la preparación profesional en beneficio de los estudiantes que estén cursando el proceso de formación académica en carreras de ingeniería en el área de electrónica, los cuales podrán poner en práctica los conocimientos adquiridos en aulas de clase e implementarlos en laboratorios de sistemas microprocesados teniendo acceso a una herramienta indispensable para el grabado en la memoria de una MCU. Con esto se abrirá la posibilidad de dejar de lado el uso de hardware externo lo que podría ayudar a los estudiantes a reducir costos para prácticas de laboratorios como también facilitar la implementación de la práctica.

Un laboratorio visto desde un punto de vista didáctico es un entorno equipado especialmente para realizar aprendizajes específicos que no se pueden desarrollar en aulas convencionales permitiendo experimentar aplicando fundamentos teóricos mediante cálculos, mediciones

y comprobaciones (Herrán, 2011). Para la realización de prácticas en un laboratorio de experimentación de sistemas microprocesados un grabador es una herramienta indispensable por la necesidad de programar por varias ocasiones a la MCU hasta llegar a culminar el objetivo de la práctica (Guadrón y Guevara, 2010).

El uso de técnicas avanzadas mediante el uso de gestores de arranque tiene una principal ventaja de grabar el archivo ejecutable en la memoria de la MCU sin el uso hardware externo. Un gestor de arranque es un pequeño programa el cual se ejecuta al momento en que se inicializa la MCU y que únicamente se coloca por una sola ocasión en una parte de la memoria haciendo uso de un grabador externo. Otras ventajas considerables del uso de gestores de arranque son: bajo costo de implementación, borrado y grabado constante en la memoria de la MCU, facilita el diseño de placas de experimentación y desarrollo escalables, códigos accesibles y gratuitos (Martínez, Marceleño, Govea y Medina, 2011).

Con el desarrollo de un gestor de arranque para MCUs se pretende beneficiar a los estudiantes tanto en el ámbito académico como social, disminuyendo el egreso económico en la adquisición de herramientas para laboratorio de sistemas microprocesados ya que es indispensable tener a disponibilidad para lograr el objetivo de una práctica. Con todo esto se podría ayudar a los estudiantes a optimizar el tiempo de armado en las prácticas y así pueda tener un proceso de aprendizaje en el que complemente tanto la teoría con la experimentación, motivando y abriendo nuevas apreciaciones sobre el óptimo uso de las MCUs pudiendo facilitar el diseño de placas de experimentación.

Capítulo 2

Revisión Literaria

En este capítulo se recopilan diversas fuentes bibliográficas, para realizar un análisis que facilite la comprensión de la temática planteada. Se enfatiza el estudio del gestor de arranque utilizado en la plataforma Arduino, que ayudará en la elaboración del presente trabajo.

2.1. Introducción

Una MCU es considerado como circuito integrado programable embebido que contiene una memoria para datos, una CPU (Unidad Central de Procesamiento), periféricos de E/S y varios recursos adicionales que ayudan al desarrollo de aplicaciones de usuario, tiene la capacidad de ejecutar instrucciones que están colocadas en parte de la memoria en codificación hexadecimal. El uso habitual de una MCU es aplicado en sistemas de control, monitoreo mediante sensores, encendido y apagado de actuadores lo que desemboca en avances de la electrónica, razón por la cual el incremento del desarrollo de sistemas embebidos autónomos lo que hace relevante su estudio en el proceso de formación profesional de esta área (Alley, 2011).

El uso de MCUs en el desarrollo de aplicaciones hace que para la etapa de implementación sea obligatorio el uso de hardware externo para que pueda ser programado, esto hace que sean manipuladas manualmente desde su circuito para ser transportadas a un programador externo o a su vez conectar un programador en algún puerto del circuito que haya sido destinado para este propósito, con estos procesos tediosos y repetitivos se coloca el archivo *.hex* (fichero de texto en formato hexadecimal) en la memoria de la MCU, esto se realiza las veces necesarias hasta alcanzar el objetivo de la aplicación.

2.2. Sistemas Embebidos Autónomos

Este tipo de sistemas electrónicos incluyen procesamiento de datos que se define en una MCU, fueron diseñados para efectuar funciones relacionadas a sistemas de ejecución en tiempo real que solventen determinadas necesidades con el objetivo de poder comunicarse con otros dispositivos a través de interfaces estándar. La idea de estos sistemas es que la mayor parte de elementos estén contenidos en una sola placa y que estén orientados a mejorar su funcionalidad cubriendo el mayor número de tareas posibles, mejorando tiempos de respuesta y confiabilidad. El principio de su creación es de que puedan ser programados de manera directa sin uso de programadores externos (UNED, 2009).

2.3. MCUs en Sistemas Embebidos Autónomos

El desarrollo de sistemas embebidos son basados en MCUs, pudiendo citar la plataforma Arduino entre una de las primeras herramientas lanzadas al mercado, siendo una plataforma open source con simplicidad tanto en hardware como software, usa AVRs (familia de MCUs)

que incorporan puertos de E/S para uso de pulsadores, variedad de sensores, encender LEDs como hasta activar actuadores mediante instrucciones que hayan sido enviadas desde el IDE (Entorno de Desarrollo Integrado) de Arduino, mismo que realiza las funciones de codificación y colocación de aplicaciones en la memoria del AVR sin tener la necesidad de usar programadores externos (Martínez, 2013).

Este sistema embebido, Arduino, usa una técnica para programado de archivos ejecutables en parte de la memoria del AVR que es mediante la implementación de un gestor de arranque o bootloader, el cual gestiona de forma automática la colocación de archivos ejecutables en parte de su memoria con la gran ventaja de permitir realizar actualizaciónes de la misma sin necesidad de usar programadores externos. Para lograr con este objetivo Arduino usa las dos secciones de memoria flash del AVR, una del gestor de arranque y otra de aplicación. Todo esto es una ventaja desde el punto de vista del usuario desarrollador que utiliza esta plataforma y que no cuente con una vasta experiencia ya que podría realizar aplicaciones en tiempos cortos y sin necesidad de un estudio profundo de una MCU.

2.4. Gestor de Arranque

Es tomado como una pequeña aplicación que es colocada en una parte de memoria de la MCU que se ejecuta al instante de ser inicializada. Se utiliza para ayudar a cargar el archivo *.hex* en parte de la memoria flash de la MCU. Para que esto ocurra debe existir dos partes, el gestor de arranque del lado de la MCU que es programada por única vez con un programador externo y la segunda el uso de una herramienta de software del lado del ordenador que tienen la función de decodificar el archivo *.hex* y enviar a través de un puerto estándar de comunicación

(USB-UART) y el gestor de arranque de la MCU reciba y coloque el archivo decodificado en una sección de su memoria (Zamora, 2017).

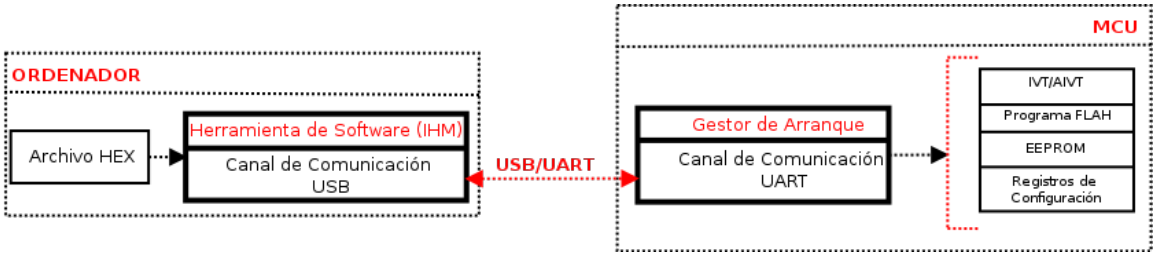


Figura 2.1: Proceso Básico del Gestor de Arranque

Fuente: Elaboración Propia

En la figura 2.1 se observa el proceso de programación de una MCU mediante el uso del gestor de arranque, se puede detallar los siguientes procesos básicos:

- La herramienta de software decodifica el archivo *.hex* previamente generado.
- Envía el archivo decodificado a través de un canal de comunicaciones (USB/UART).
- El gestor de arranque recibe y coloca el archivo en parte de la memoria flash de la MCU, evitando sobre escribirse.

2.4.1. Gestor de Arranque del Arduino

Para este análisis se ha tomado como referencia a la tarjeta Arduino UNO que tiene como núcleo el microcontrolador ATMega328P en el que es colocado un gestor de arranque o también llamado ”firmwareza que no es común su manipulción por parte de los usuarios, se ocupa de cargar y hacer funcionar los sketches programados en el IDE de Arduino, ocupando 512 bytes de los 32KB disponibles en la memoria flash (Lewis, 2016). Las características principales del ATMega328P se resume en la tabla 2.1.

Tabla 2.1: Principales Características ATMEGA328P

CARACTERÍSTICA	VALOR
Memoria FLASH, SRAM, EEPROM	32KB, 2KB, 1KB
Interrupciones, Canales PWM	26, 6
Voltaje de Operación	1.8 - 5.5V
USART	1
CPU	8 bits
Frecuencia Max.	20Mhz
Terminales E/S	23
Oscilador Interno	1Mhz/8Mhz

Fuente: Datasheet ATMEGA328P

Podríamos mencionar una analogía del gestor de arranque de Arduino con la conocida y tradicional BIOS que se ejecuta sobre los ordenadores en el instante de que son encendidos, pero en este caso aguarda un tiempo para detectar actividad por parte del usuario y ser configurada, en caso de no detectar alguna actividad empieza la ejecución del sistema operativo preinstalado a diferencia del proceso semejante del gestor de arranque de Arduino ya que cada vez que se reinicia o se enciende busca actividad de entradas externas y si no existe ejecuta la última aplicación de usuario que se haya cargado.

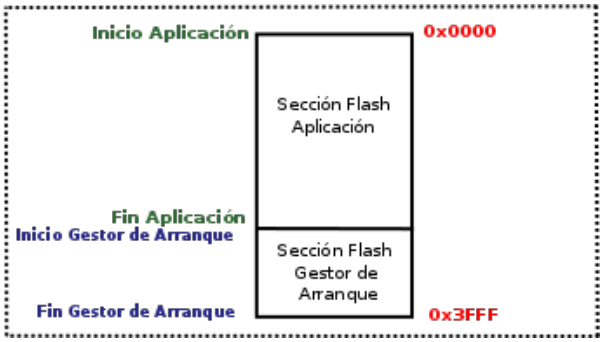


Figura 2.2: Secciones de Memoria Flash Atmega328P

Fuente: https://goo.gl/1of8u1

En la figura 2.2 hace referencia a la memoria flash del microcontrolador Atmega328P el cual se caracteriza por tener dos secciones de memoria, sección para aplicación de usuario y sección para gestor de arranque ubicada en la parte inferior de esta memoria.

2.4.1.1. Modo de Inicio del Gestor de Arranque

Cuando el microcontrolador Atmega328P se pone en marcha o a su vez se reinicia, básicamente entra en ejecución la aplicación cargada desde el vector de reinicio (desde la dirección 0x0000). Existe la manera de cambiar la dirección de vector de reinicio a la de inicio de la sección del gestor de arranque, en caso de que se esté usando, con esto cada vez que se encienda o se reinicie el microcontrolador entrará en ejecución el gestor de arranque que es colocado en la sección de memoria correspondiente (Electronicwings, 2018). En la figura 2.3 muestra el diagrama de proceso del gestor de arranque que se ejecuta una vez reiniciado el microcontrolador.

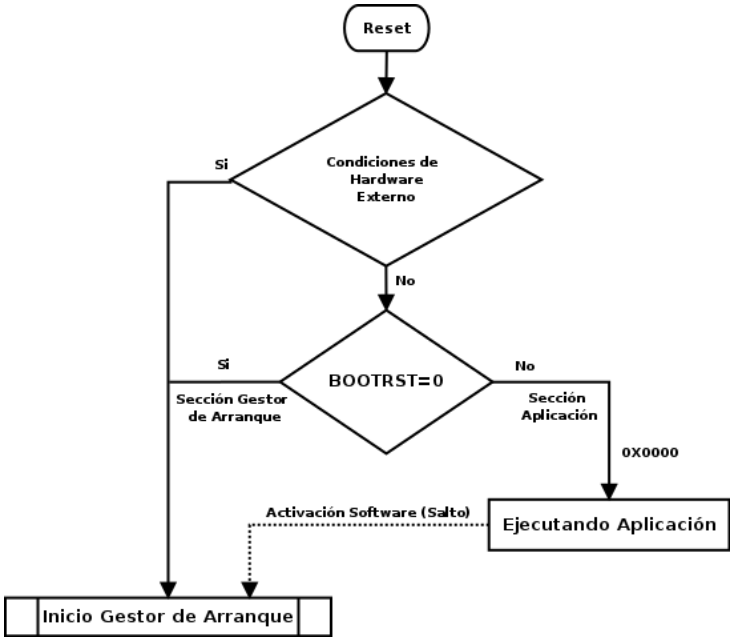


Figura 2.3: Diagrama del Proceso del Gestor de Arranque Atmega328P

Fuente: https://goo.gl/V5dLgo

Este principio aprovecha la plataforma Arduino que bajo el Atmega328P configura el fusible de inicio/reinicio (BOOTRST) para que dicho vector sea dirigido al inicio de la sección del gestor de arranque, según se puede evidenciar en la tabla 2.2.

Tabla 2.2: Fusible de Reinicio del Gestor de Arranque ATMEGA328P

Característica	BOOTRST	Dirección de Reinicio
No Programado	1	Reset Vector = Application Reset (dirección 0x0000)
Programado	0	Reset Vector = BootLoader Reset (dirección 0x3E00)

Fuente: https://goo.gl/V5dLgo

El funcionamiento del gestor de arranque entra en ejecución cuando el Atmega328P es iniciado o después de un reset, en un reducido tiempo espera verifica alguna actividad (interrupción serial) de llegada de un reciente sketch proveniente del IDE de Arduino para ser guardado y ejecutado, de no ser así pasa a ejecutar el sketch que se haya cargado con anterioridad es decir pasa a redirige al inicio de la sección de aplicación. En la actualidad las placas Arduino poseen una función ”autoreset” que ayuda al IDE de Arduino a programar código sin tener la necesidad de pulsar un reset.

2.4.1.2. Tamaño de la Sección para el Gestor de Arranque

La cantidad de espacio que tiene disponible el Atmega328P para la sección del gestor de arranque es parte fundamental para que Arduino haya diseñado el mismo. Para que una aplicación tenga acceso a la sección del gestor de arranque debe hacerlo mediante el uso de rutinas o saltos. Este microcontrolador trae de fábrica el BOOTZ=”00” que permite obtener 32 páginas para la sección del gestor de arranque y por la razón mencionada en la sección 2.4.1.1. este microcontrolador al momento de ser encendido se ejecuta la aplicación de usuario (Electronicwings, 2018). El fusible ”BOOTRST” se puede configurar de tal manera que después de un reset el microcontrolador inicie desde la sección de gestor de arranque como se muestra en la tabla 2.2. Según el datasheet el tamaño para esta sección es seleccionable por el desarrollador

como se muestra en la tabla 2.3.

Tabla 2.3: Configuración de Tamaño de Memoria

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Secction	Boot Loader Flash Section
1	1	256 words	4	0x0000- 0x3EFF	0x3F00- 0x3FFF
1	0	512 words	8	0x0000- 0x3DFF	0x3E00- 0x3FFF
0	1	1024 words	16	0x0000- 0x3BFF	0x3C00- 0x3FFF
0	0	2048 words	32	0x0000- 0x37FF	0x3800- 0x3FFF

Fuente: https://goo.gl/V5dLgo

Como se menciona en la sección 2.4.1, para la configuración de inicio Arduino selecciona un tamaño de 512 Bytes, como el tamaño para el gestor de arranque, lo que se da en palabras que equivale a la mitad del tamaño en Bytes entonces significa que usa 256 palabras que implica que la rutina debe estar configurada como BOOTZ=”11”, en base a la tabla 2.3. podemos concluir que para la plataforma Arduino tenemos:

- Selecciona el tamaño de 256 palabras para la sección de gestor de arranque.
- El gestor de arranque debe caber en 512 bytes de memoria.
- El gestor de arranque debe estar dentro de la dirección 0x3F00 a 0x3FFF.
- La dirección de inicio es 0x3F00.

2.4.1.3. Importancia del Gestor de Arranque

La plataforma Arduino hace uso del gestor de arranque que se encuentra en parte de la memoria del microcontrolador para facilitar la colocación de archivos *.hex* en la sección de aplicación de su memoria flash haciendo uso de la comunicación serial (UART). Este gestor de arranque se ejecuta en el primer segundo después de que se reinicie el microcontrolador. La figura 2.4 esquematiza el proceso de cómo el gestor de arranque coloca el código hexadecimal decodificado recibido desde el IDE de Arduino directamente en la sección de memoria de

aplicaciones.

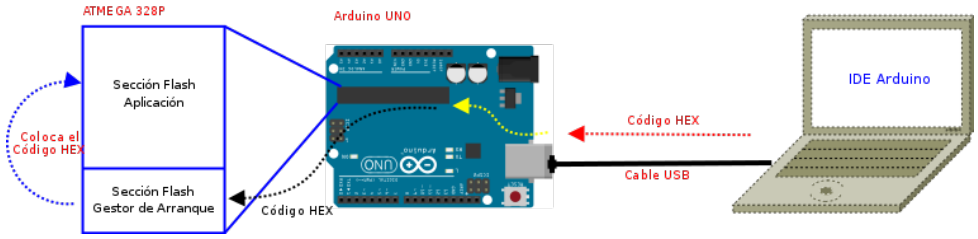


Figura 2.4: Proceso de Programación para Arduino UNO

Fuente: https://goo.gl/1of8u1

La tarjeta Arduino incluye un elemento (adaptador) USB/USART lo que es aprovechado por el gestor de arranque para no usar hardware externo de comunicaciones para la programación de aplicaciones.

2.4.1.4. Proceso de Programación en el Atmega328P

La manera por la cual la memoria flash de aplicación se programa o actualiza es bajo la conceptualización de página por página variando el tamaño dependiendo del microcontrolador utilizado, según el datasheet posee un tamaño de página de 64 palabras equivalente a 128 bytes, para el proceso de escritura de la memoria flash. Según Electronicwings (2018) menciona una idea general del proceso de programación en la siguiente secuencia:

- Los bytes hexadecimales que son transmitidos mediante comunicación serial desde el IDE de Arduino son primero copiados en la memoria temporal de datos (RAM).
- Ejecuta el borrado de la primera página de la memoria flash de aplicación.
- Rellena (no escribe) con bytes hexadecimales almacenados en la memoria de datos temporal.
- Utilizando instrucciones de programación de páginas SPM (modo de auto programación) hace que la escritura o actualización de páginas se realice de manera efectiva.

- Los procesos anteriores se repiten hasta que todos los bytes hexadecimales se escriban o actualicen en la memoria de aplicación.

Una vez completado todas las operaciones, procede a verificar si el archivo hexadecimal se escribió o actualizó en la memoria de aplicación realizando un proceso opuesto, el cual realiza una operación de lectura de la memoria de aplicación y sea transmitido por comunicación serial al ordenador página por página. Estos procesos son posibles gracias a que los microcontroladores AVR poseen un mecanismo SPM tanto para cargar y descargar código del mismo microcontrolador usando cualquier puerto de comunicación tanto para leer el código como para programar en la memoria de aplicaciones.

2.4.1.5. Proceso de Programación desde IDE de Arduino

La plataforma Arduino mediante el gestor arranque y con ayuda del protocolo serial (UART) puede cargar o descarga el archivo *.hex* de la aplicación desde el ordenador ya que el IDE de Arduino es el encargado de compilar el archivo de aplicación y enviar el código compilado a la tarjeta Arduino por el puerto USB. Lewis (2016) menciona que el IDE de Arduino utiliza la herramienta de software AVRDUDE (AVR Downloader Uploader) para realizar las acciones antes mencionadas teniendo las siguientes características:

- Programar la Flash y la EEPROM.
- Compatible con el protocolo de programación en serie.
- Programar fusibles y bits de bloqueo.
- Utiliza el protocolo de comunicación STK500 para cargar el archivo codificado a la memoria del microcontrolador.

El protocolo STK500 (propietario de AVR) es usado para la comunicación entre AVRDUDE y el gestor de arranque para programar el archivo *.hex* a la memoria del microcontrolador AVR. Entonces, con la ayuda de la herramienta AVRDUDE el IDE de arduino puede cargar el archivo compilado directamente al microcontrolador Atmega328P de la tarjeta.

2.4.1.6. Comunicación del IDE de Arduino con Gestor de Arranque

Tiene mucha importancia la comunicación entre el microcontrolador de la tarjeta con el IDE de Arduino ya que es la parte fundamental para que el archivo *.hex* compilado pueda llegar a ser colocado en la memoria de aplicación del AVR, una vez que el IDE de Arduino logra codificar y compilar el archivo ejecutable pasa a ser enviado mediante el puerto USB por comunicación serial y así exista actividad en el mismo puerto del microcontrolador para que el gestor de arranque coloque el archivo codificado recibido en el sector de memoria destinado. Entonces, de manera general el gestor de arranque se encarga de recibir el archivo compilado desde el IDE de Arduino por comunicación serial y de manera ”simultánea” coloca en la sección de memoria de aplicación como se muestra en el esquema de la figura 2.4.

2.5. Interfaz de línea de comandos AVRDUDE

Si hemos hablado de la plataforma Arduino no podemos dejar de lado al software de programación de para AVRs llamado AVRDUDE el que fue creado para programar la memoria flash y EEPROM mediante comunicación serial, también usada para leer o escribir sobre la memoria del AVR a través de línea de comandos. Hace uso de hardware adicional para cumplir con este objetivo mismo que es conectado al puerto ISP de la tarjeta Arduino, la figura 2.4 se muestra la interfaz de línea de comandos usada por AVRDUDE (Savannah, 2010).

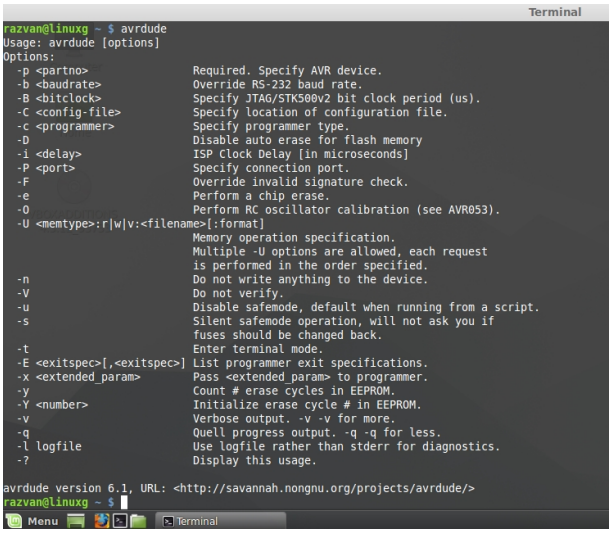


Figura 2.5: Interfaz de línea de comandos AVRDUDE

Fuente:http://bit.ly/2DujsQ1

Las principales características de AVRDUDE son las siguientes:

- Interfaz de usuario controlada por línea de comandos.
- Ofrece una opción para modificar los parámetros operativos.
- Compatible con el protocolo de programación en serie.
- Programa fusibles y bits de bloqueo.
- Utiliza el protocolo de comunicación STK500 para cargar el archivo codificado a la memoria del AVR.

Se ejecuta en los principales sistemas operativos (LINUX, WINDOWS y MacOS X) admitiendo una amplia gama de hardware de programación. Usado principalmente para actualizar el firmware de la placa Arduino en caso de tener algún tipo daño por alguna causa o se necesite cambiar de chip a la tarjeta.

2.6. Tarjeta de Desarrollo FLEX-Full

Esta tarjeta de desarrollo también considerada como un sistema embebido la cual fue diseñada con el objetivo de desarrollar y ensayar aplicaciones en tiempo real, principalmente contiene dos MCUs sin ningún firmware, un PIC18F2550 (PIC18F) y un dsPIC33FJ256MC710 (dsPIC33F). Basada estructuralmente en una arquitectura modular piggybacking que permite agregar nuevo hardware exportando la mayoría de los pines de los microcontroladores para facilitar la construcción de aplicaciones específicas. La incorporación del PIC18F en la tarjeta facilita el desarrollo de aplicaciones ya que puede servir para obtener una comunicación y programación directa del dsPIC33F mediante el uso del puerto USB, se alimenta con voltajes que van desde los 9V a 36V, posee un puerto para ICD2 o ICD3, un conector USB con conexión directa al PIC18F y seis Leds de estado que se detalla en las tablas del Anexo A (Camacho, 2009).

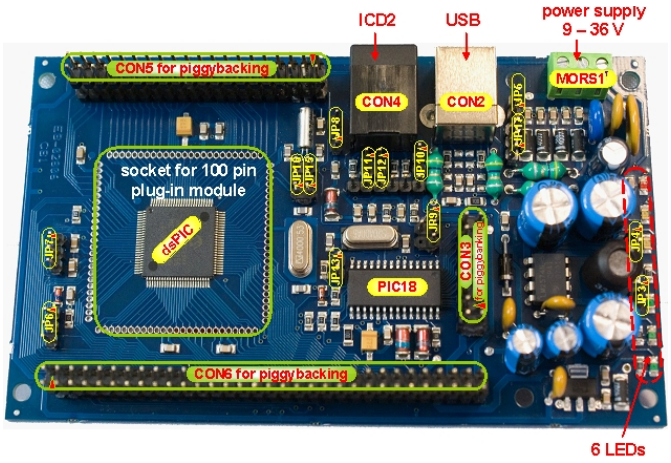


Figura 2.6: Tarjeta de Desarrollo Flex-Full

Extraído de: https://goo.gl/RXS29X

La figura 2.6 trata de mostrar que los pines de los microcontroladores se extienden hasta los puertos ”CON3”, ”CON5” y ”CON6” como se muestra en las tablas del anexo A lo que

facilita su uso para diferentes desarrollos de aplicaciones. Las tarjetas de desarrollo con este tipo de dsPIC33F son orientadas al tratamiento de señales por medios digitales, entre sus principales características sobresalen el poseer un tamaño de núcleo de 16 bits, 256 KB para memoria de programa, dos convertidores A/D de 12 bits, un controlador de DMA (Acceso Directo a Memoria) de 8 canales, dos módulos CAN y 12 canales PWM.

2.6.1. Campos de Aplicación para dsPIC33F

Este tipo de MCU podrían asemejarse a los microcontroladores habituales, pero integrando recursos físicos y lógicos para soportar aplicaciones específicas del procesamiento digital de señales, aunque su parecido es muy grande también los distingue cada campo de aplicación. Los dsPIC33F están preparados para que puedan ser programados en un lenguaje de alto nivel, como el C, y al disponer de conversores A/D rápidos y precisos con velocidad y rendimiento superiores a los habituales microcontroladores permite el desarrollo de aplicaciones robustas que hagan uso de sensores que exijan convertidores A/D de precisión y alto volumen de procesamiento (Angulo, 2016).

Son comúnmente usadas en el tratamiento de señales cuando se necesita llegar a una solución eficiente en aplicaciones en tiempo real, también son idóneas para aplicaciones de tratamiento de voz y audio, incorporan mayores niveles de integración para aplicaciones de control de motores ofreciendo diversidad de aplicaciones, como los de fase simple o trifásica, inducción y corriente continua. También son muy apropiados para la gestión de los sistemas de alimentación ininterrumpibles, módulos para la corrección del factor de potencia, telecomunicaciones, conversores o algún equipamiento en el ámbido industrial (Angulo, 2016).

2.6.2. Programación dsPIC33F para Tarjeta FLEX-Full

La forma convencional que se usa para programar el dsPIC33F es a través de un programador externo. Por ejemplo, en la figura 2.7 se muestra un esquema básico de la programación convencional de la tarjeta FLEX-Full donde, el archivo ejecutable que es producido desde el código de usuario debe ser traducido a un archivo binario mediante la utilización de un IDE adecuado para que posteriormente sea enviado a través del programador externo (ICD3) y así logre el objetivo de que el dsPIC33F sea programado, siendo procesos manuales que aumenta costos de aplicación sin mencionar que se debe conectar el programador a la tarjeta Flex-Full cada vez que se intente programar un nuevo archivo ejecutable.

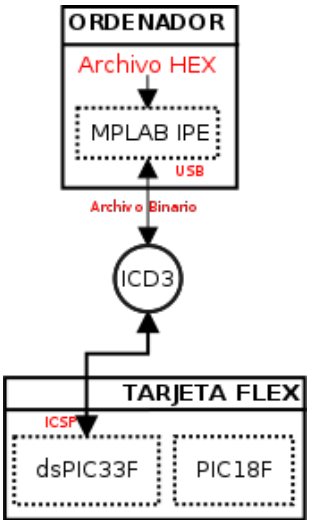


Figura 2.7: Programación habitual del dsPIC33F

Fuente: Elaboración Propia

El esquema de la figura 2.7 muestra un proceso habitual que se utiliza para la programación del dsPIC33F dejando de lado al PIC18F, aún perteneciendo a la misma tarjeta, lo que abre la posibilidad de poder desarrollar algún método (por hardware o software) de programación tomando como referencia el análisis que se a tomado de la plataforma Arduino, que podría integrar las dos MCUs y optimizar recursos de esta tarjeta de desarrollo.

2.7. Metodología de Desarrollo Bottom-Up

Esta metodología reúne sistemas que deben ser bien especificados de manera individual para cuando actúen en conjunto puedan conformar un producto final, esta metodoloía permite unir partes de desarrollo hasta conformar un producto final es comparado al modelo metodológico llamado ”semilla”, en el cual inicia desde algo relativamente pequeño y va evolucionando hasta llegar a un sistema terminado. También es conocida como modelo de desarrollo evolutivo, empieza con la definición de los objetivos generales para posteriormente identificar los requisitos y funcionalidades puntualizando en las áreas de esquemas importantes (Caicedo, 2018).

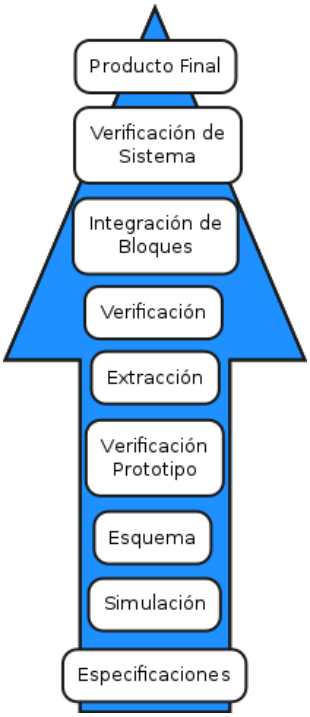


Figura 2.8: Etapas Metodología Botton-Up

Fuente: http://bit.ly/2jJ2irq

Esta metodología plantéa realizar un seguimiento de las diferentes etapas y pasos para la obteción de un producto final, como se puede evidenciar en la figura 2.8 en la cual se presenta el ciclo que realiza siguiendo la estructura propuesta por la metodología dividida en bloques

funcionales. Este modelo es versátil ya que no es necesario tener una idea o imagen clara para el producto final sino basta tener características requeridas para en posteriores etapas juntar los diferentes desarrollos que conformarán el sistema.

Capítulo 3

Requerimientos y Funcionalidades

En este capítulo se determina los requerimientos y funcionalidades para el desarrollo del gestor de arranque, puente USB/UART e MHI, haciendo uso de la metodología Bottom-Up con apoyo en diagramas de bloque y de flujo que permita obtener un adecuado proceso de desarrollo de las mismas.

3.1. Metodología Botton-Up

Esta práctica metodología que es usada para el desarrollo de hardware y software permite describir en varios niveles y subniveles el conjunto de desarrollos para el gestor de arranque. Esta metodología plantea un ciclo metodológico versátil para la creación del gestor de arranque, puente USB/UART y MHI como se muestra en la figura 3.1.

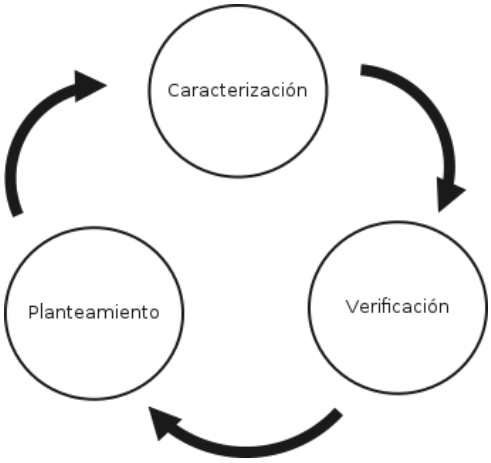


Figura 3.1: Ciclo metodológico Botton-Up

Extraído de: http://bit.ly/2XVgqzz p.34

Se toma en cuenta que se requiere precisión para la programación de memoria flash del dsPIC33F, para la cual se divide en tres desarrollos funcionales como son: desarrollo del gestor de arranque, puente UART/USB e MHI mismos que son independientes pero complementarios.

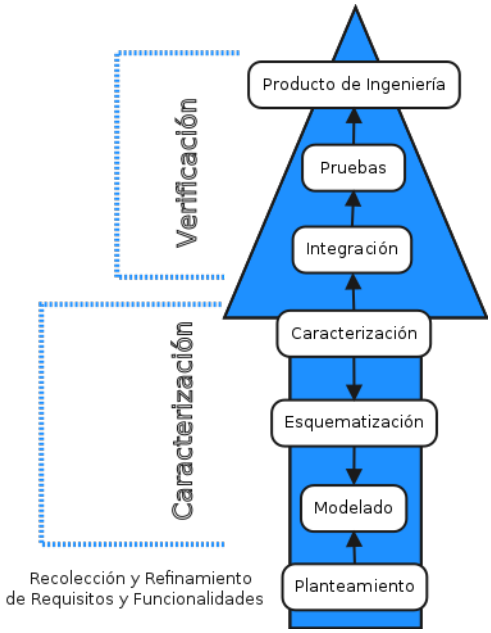


Figura 3.2: Subsecciones Metodología Botton-Up

Extraído de: http://bit.ly/2XWl0O7

Los tres desarrollos que se necesita para una integración final en función al gestor de

arranque se sintetizan en tres etapas del ciclo metodológico de Bottom-Up, que direcciona hacia la calidad del desarrollo total del sistema, como se muestran en la figura 3.2. El orden de desarrollo consecutivo de esta metodología según la figura 3.2 es determinada en las siguientes subsecciones:

- **Planteamiento:** Recolección y refinamiento de requisitos y funcionalidades partiendo desde una situación actual de la forma convencional de programas los dsPIC33F basándose en esquemas de planteamiento.
- **Esquematización:** Planteamiento de esquema que pueda mostrar el sistema completo para el gestor de arranque.
- **Modelado:** Breve diseño para el gestor de arranque en consecuencia de los requisitos y funcionalidades.
- **Caracterización:** Construcción del gestor de arranque basado en la subsección anterior.
- **Integración:** Complementación de desarrollos de la subsección anterior.
- **Pruebas:** Evaluación de la integración para el uso del gestor de arranque.
- **Producto de Ingeniería:** Ejemplificación con el uso del gestor de arranque y resultados finales.

3.2. Planteamiento

En base a la situación actual de la forma convencional de cómo se programa la memoria flash de los dsPIC33F se plantea los requerimiento y funcionalidades que pueden abarcar tanto hardware como software pudiendo ser parte fundamental para que este desarrollo logre los objetivos básicos con lo cual el usuario pueda tener acceso a un nuevo método de programación haga uso del mismo sin ningún problema, todo ello se trata a lo largo de todo este capítulo.

3.2.1. Situación Actual

Es importante señalar que para este trabajo se utiliza la tarjeta de desarrollo FLEX-Full misma que en la sección 2.6.2 se menciona a detalle de como el dsPIC33F el que integra dicha tarjeta es programada de forma habitual, lo que puede no ser muy eficiente desde el punto de vista que ha sido creado. Esta tarjeta es usada por desarrolladores que tienen conocimientos previos en el ámbito de microcontroladores específicamente de Microchip ya que tanto su estructura lógica como física obligan a hacerlo. Se puede observar en el esquema de la figura 2.7 el proceso de programación convencional que se usa para el dsPIC33F para el caso que se necesite grabar cualquier aplicación de usuario, el proceso es descrito a continuación:

- El archivo *.hex* (código hexadecimal de programa) es cargado a MPLAB IPE (Entorno de Programación Integrada) que conjuntamente con el framework de MPLAB IDE, la base de datos del depurador de Microchip, el puerto de comunicación y controladores respectivos puedan proporcionar capacidades de programación.
- Una vez cargado el archivo *.hex* es decodificado y enviado por el puerto USB al ICD3 (Depurador en Circuito) para que pueda ser depurado y posteriormente enviado directamente a la sección de memoria flash de aplicación del dsPIC33F.
- El ICD3 conectado al puerto correspondiente de la tarjeta FLEX-Full la cual tiene comunicación directa con el puerto ICSP del dsPIC33F para que finalmente ser programada.

3.2.2. Forma Propuesta de Programación

Las razones citadas en las secciónes 1.1. y 2.6.2, hacen que la tarjeta FLEX-Full que incorpora dos microcontroladores sea aprovechada de manera eficiente, pudiendo realizar configuraciones para que las dos MCUs actúen conjuntamente y el dsPIC33F pueda ser programado

sin hacer uso de un programador externo, para lo cual se plantea una nueva forma de programación.

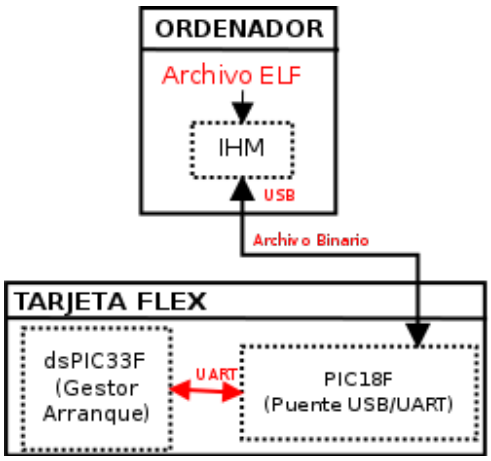


Figura 3.3: Esquema de Planteamiento de Programación

Fuente: Elaboración Propia

A partir del esquema de la figura 3.3 se plantea el siguiente proceso de programación:

- El archivo *.elf* (Formato Ejecutable y Vinculable) debe ser cargado a la MHI para ser decodificado.
- Una vez tratado el archivo *.elf*, la MHI envía por comunicación serial hacia el dsPIC33F mediante el puente USB/UART.
- La tarjeta FLEX-Full mediante el puerto correspondiente debe ser conectada al ordenador que contiene la MHI, para que el dsPIC33F que debe tener precargado el gestor de arranque ayude a colocar el archivo *.elf* decodificado en la sección de memoria flash correspondiente.

3.2.2.1. Comparación de Procesos de Programación

Para una mejor apreciación la forma de programación propuesto para el dsPIC33F, en la figura 3.4 se realiza una comparación entre esquemas de programación de las figuras 2.7 y 3.3

como se muestra a continuación.

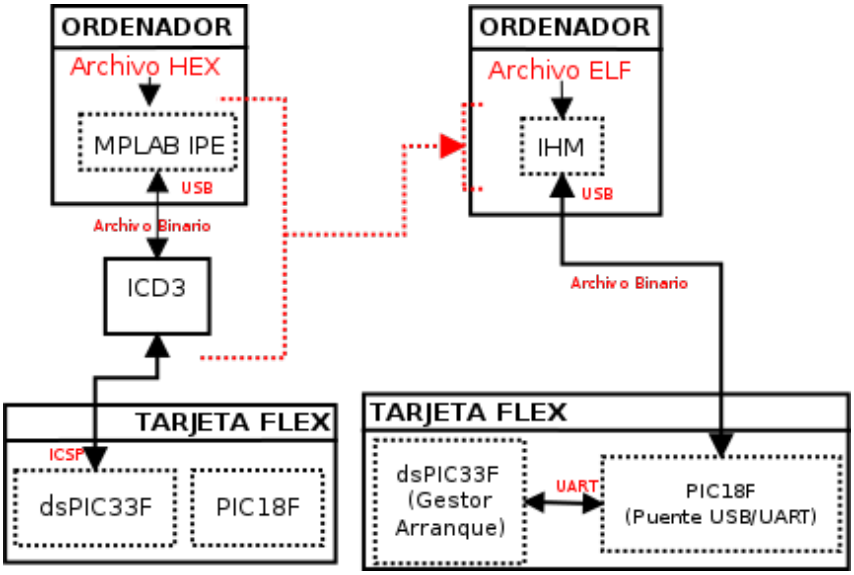


Figura 3.4: Comparación Esquemas de Programación para dsPIC33F

Fuente: Elaboración Propia

En la figura 3.4 se trata de poner en evidencia que el nuevo enfoque plantea reemplazar en gran medida al programador externo por un gestor de arranque colocado en el dsPIC33F, un puente USB/UART sobre el PIC18F que deben actuar de manera conjunta con la aplicación (MHI) para el gestor de arranque logre el objetivo de programar archivos *.elf* en parte de la memoria flash del dsPIC33F destinada para aplicaciones de usuario.

3.3. Requerimientos

Como se menciona en la sección 1.4 donde pone en evidencia una necesidad de la creación de una herramienta MHI y un gestor de arranque que reemplacen al programador externo, para ello, el desarrollo de la MHI se basará en lenguaje de programación Python el que permita decodificar archivos *.elf* como también enviar los mismos mediante el puente USB/UART a la memoria del dsPIC33F el cual contendrá el gestor de arranque escrito en lenguaje C. Es-

ta propuesta permite dejar de lado el uso de hardware adicional para la comunicación entre el dsPIC33F y un ordenador para la etapa de ejecución de alguna aplicación final de usuario.

3.3.1. Requerimientos de Hardware

Para el diseño tanto del gestor de arranque como del puente USB/UART se programarán sobre las MCUs (dsPIC33F y PIC18F respectivamente) de la tarjeta de desarrollo FLEX-Full, como se muestra en la figura 2.7 el proceso habitual que es sometido el dsPIC33F para poder ser programada, esta es comparada con la nueva propuesta de programación que reemplaza al programador externo y así facilitar a los desarrolladores el uso de nuevos enfoques para este tipo de tarjetas.

La tarjeta FLEX-Full no tiene colocado ningún firmware en sus MCUs, haciendo esto que sea idónea para el planteamiento propuesto en la figura 3.3, lo cual permite implementar el gestor de arranque sobre la dsPIC33F y un firmware que actúe como puente USB/UART sobre el PIC18F, estos dos al trabajar conjuntamente permite obtener una comunicación directa entre el ordenador (HMI) y el dsPIC33F y permita colocar (leer, actualizar y borrar) aplicaciones de usuario las veces que sean necesarias sin el uso de algún programador externo. Basado en la figura 3.3 se detallan los requerimientos de hardware siguientes:

- Utilización por única vez de un programador externo para colocar el firmware (gestor de arranque y puente USB/UART) desarrollado a las dos MCUs.
- Utilización del PIC18F como intermediario de comunicación entre el dsPIC33F y el ordenador.
- Utilización del puerto USB tipo B de la tarjeta FLEX-Full para conexión cableada (cable serial) con el ordenador (MHI).

3.3.2. Requerimientos de Software

Para estos requerimientos que se centralizan en dos partes principales de desarrollo que se apoyan en plataformas GNU/LINUX y el editor MPLAB IDE. Para el desarrollo del gestor de arranque y del puente UART/USB se hará uso del editor MPLAB IDE de Microchip en lenguaje C y ensamblador, tomando en cuenta la comunicación UART entre las MCUs y el puerto USB del PIC18F que permita alcanzar el planteamiento del esquema de la figura 3.3.

Los requerimientos para la MHI se basa en lenguaje de programación Python para sistemas operativos GNU/LINUX, mediante el cual obtener una aplicación que se ejecute por la terminal de consola para que pueda realizar las funciones de la decodificación de archivos *.elf* y de la gestión de la comunicación serial entre el ordenador con el dsPIC33F además del proceso de lectura, escritura y borrado de las diferentes secciones de la memoria flash del dsPIC33F. Para cualquier aplicación de usuario que haga uso del gestor de arranque debe cumplir dos parámetros básicos que son:

- El código de usuario no puede hacer uso del espacio de memoria reservado para el gestor de arranque.
- El tamaño no debe exceder del espacio de memoria que se asigne para aplicaciones de usuario.

3.3.3. Requerimientos para Usuarios

El usuario como desarrollador de aplicaciones basadas en microcontroladores, debe poseer conocimientos en los ámbitos como: programación de microcontroladores Microchip (usado en la tarjeta FLEX-Full), manejo básico de sistemas operativos GNU/Linux (interfaz de línea

de comandos), lenguajes de programación C, ensamblador (en caso de ser requerido) y Python (uso básico e instalación). Con los requerimientos de usuario nombrados anteriormente el desarrollador puede hacer uso de la forma de programación que se plantea para el dsPIC33F sin mayor problema.

3.4. Funcionalidades

Las funcionalidades principales se acentúan en dos partes fundamentales para este desarrollo, el gestor de arranque y la MHI, para las cuales se las trata de manera independiente.

3.4.1. Funcionalidades de Hardware

El dsPIC33F además de ofrecer precisión de resultados tiene la ventaja de poseer gran velocidad de procesamiento, esto ayuda a que sea usada de una manera óptima por el gestor de arranque, este será colocado en cierta parte de sección su memoria flash, con el propósito de ayudar a colocar archivos *.elf* en la memoria destinada para programas de usuario. Consta de dos partes, el gestor de arranque por parte del dsPIC33F y la herramienta de software para el ordenador (MHI).

Para la comunicación del dsPIC33F y el ordenador es necesario el desarrollo de un método de conexión que permita existir transmisión de datos entre los dos, esto se podrá lograr haciendo uso de la funcionalidad de la tarjeta que incorpora un PIC18F que permite desarrollos con conexiones USB 2.0 a velocidades de 480Mbps. Para lo cual ayuda a la creación de un firmware que actúe como puente USB/UART y permita la comunicación USB de un ordenador con la UART del dsPIC33F que permitirá lograr registrar actividad en el puerto destino y así

gestor de arranque pueda entrar en actividad.

3.4.2. Funcionalidades de Software

Como ya se mencionó, las funciones de la herramienta MHI se basa en lenguaje de programación Python que tiene como funciones principales la codificación de los archivos *elf* a binario y de mantener la comunicación serial con el dsPIC33F, la interfaz humano máquina se ejecuta desde línea de comandos (CLI), para ello se debe tener en cuenta los parámetros principales como la interpretación de comandos de cadenas de datos (string) y baudrate que define la velocidad de transferencia. Se toma en cuenta que tanto la MHI y el gestor de arranque deberán trabajar a una misma velocidad de transmisión.

Capítulo 4

Desarrollo del Sistema

En este capítulo, se procede a seguir con las dos subsecciones de esquematización, modelado y carecterización para el gestor de arranque en base al planteamiento que se logró durante el desarrollo del capítulo anterior.

4.1. Modelado

Una vez que se pudo obtener los requerimientos y funcionalidades para el desarrollo del gestor de arranque, se inicia con el modelado o una descripción breve del conjunto de desarrollos para el gestor de arranque que ayude a obtener una idea más clara para las siguientes subsecciones de la metodología.

4.1.1. Descripción del Sistema

El desarrollo del gestor de arranque puede simplificar el proceso de desarrollos de proyectos prácticos cuando se utilice el dsPIC33F, para lo cual se reemplaza el uso de herramientas de hardware convencionales por una alternativa de software para programación del dsPIC33F,

de tal manera se cambia el uso de MPLAB IPE y el programador ICD3, como se muestra en la figura 3.3 que propone implementar mediante un ordenador con sistema operativo GNU/Linux y una tarjeta FLEX-Full una forma alternativa de programación para su dsPIC33F. La secuencia básica para la programación que se planteada es la siguiente:

- Se debe obtener el archivo ejecutable *.elf* después del procedimiento normal de desarrollo de aplicaciones en MPLAB IDE.
- La MHI hace una ”llamada” al gestor de arranque del dsPIC33F para que pueda recibir el archivo *.elf* decodificado mediante el puente USB/UART y gestione la colocación en una sección de su memoria Flash.

Es importante mencionar que el uso de la aplicación para el gestor de arranque se convierte en obligatorio para poder tener una actualización del código en el dsPIC33F. Este proceso es mucho más simple al convencional para el desarrollo de aplicaciones de usuario. La caracterización de la propuesta planteada abarca el uso de PIC18F como puente USB/UART que reemplaza a la comunicación serial convencional utilizada para la transmisión de datos entre el dsPIC33F y un ordenador.

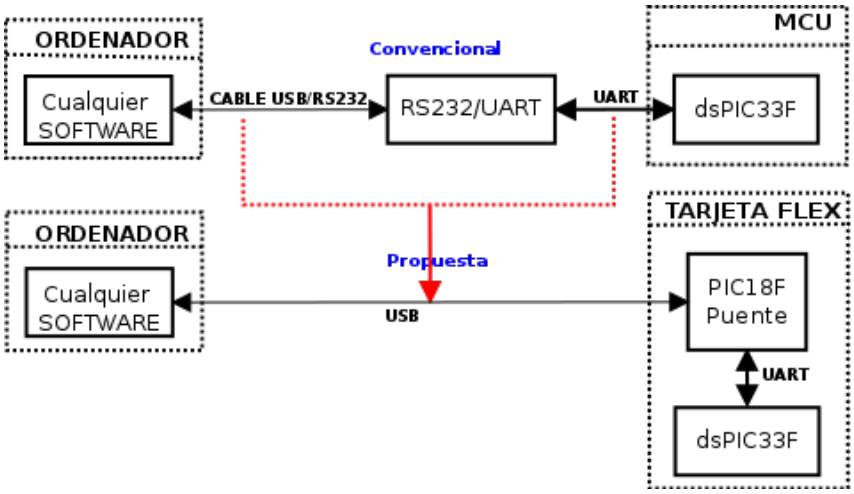


Figura 4.1: Comunicación común dsPIC33F-Ordenador y forma propuesta

Fuente: Elaboración Propia

En la figura 4.1 se compara la forma común de comunicación serial con la forma propuesta que ayuda a un mejor modelado para el puente USB/UART.

El gestor de arranque tiene dos modos para su desarrollo, modo depuración y automático. El modo depuración tiene el objetivo de depurar el código de cargado haciendo uso de herramientas de hardware como el ICD3 y el modo autónomo no depura el código de cargador, este último se utiliza para nuestro diseño ya que la herramienta MHI es el encargado de realizar el proceso de depuración.

4.2. Esquematización

Para esta subsección iniciaremos representando en un esquema de bloques que representan cada uno de los 3 desarrollos para el sistema. Se parte desde el núcleo que es el gestor de arranque sobre el dsPIC33F, el puente USB/UART en el PIC18F y la MHI en un ordenador como se indica el esquema de la figura 4.2, este esquema pone en evidencia los procesos consecutivos en el que inicia obteniendo de alguna manera el archivo *.elf* para que finalmente se programe en la memoria Flash del dsPIC33F.

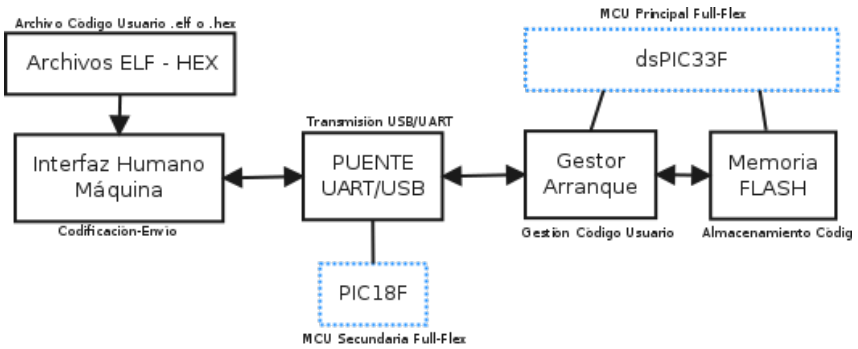


Figura 4.2: Diagrama de Bloques de Sistema

Fuente: Elaboración Propia

4.3. Caracterización

4.3.1. Desarrollo del Gestor de Arranque

Cuando se produce un reset total o se alimenta el dsPIC33F en primera instancia esta ejecuta el vector *reset* que se encuentra en la dirección *0x000* que también contiene una instrucción de salto hacia la sección donde se encuentra el código de usuario desde la dirección *0xC02*, para cambiar todo esto y que la instrucción realice el salto hacia el gestor de arranque desde la sección *0x400* se logra cambiando la configuración del fusible *reset* para que cada vez que exista un inicio/reset de la MCU ejecute el gestor de arranque en lugar del código de usuario.

4.3.1.1. Distribución de Memoria

Aunque el gestor de arranque requiere un mínimo de memoria, la arquitectura del dsPIC33F restringe un rango de memoria para ser dedicada para esta pequeña aplicación. Una aplicación para que pueda ser colocada en la memoria del dsPIC33F, el gestor de arranque debe primero borrarla, también tiene la funcionalidad de hacer un borrado de toda la memoria flash o borrado por direcciones de memoria.

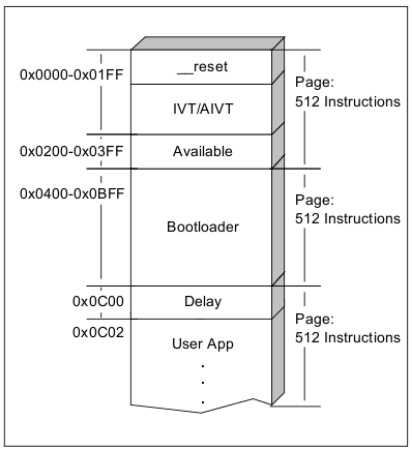


Figura 4.3: Distribución Memoria dsPIC33F

Fuente: Datasheet

En la figura 4.3 se muestra la organización de la memoria del dsPIC33F.

4.3.1.2. Valores de Retardo

Estos valores de retardo mostrados en la tabla 4.3.1.2 son los escogidos para el diseño del gestor de arranque.

Tabla 4.1: Valores de Retardo Válidos

Valores de retardo (Segundos)	**Resultado**
0	Suspende el gestor de arranque y transfiere la ejecución a la aplicación del usuario.
1-254	Espera el tiempo especificado para transferir archivos *.elf*, si no se actividad UART en ese tiempo transfiere la ejecución a la aplicación de usuario.
255	Espera ilimitadamente para la transferencia de archivos *.elf*

Fuente: Datasheet

4.3.1.3. Gestor de Arranque

El gestor de arranque se encuentra en una sección de memoria del dsPIC33F dedicada para esta pequeña aplicación que va a partir de la dirección *0x400* hasta *0xBFF*, cuanto este se esté ejecutando espera un tiempo indefinido de retardo hasta recibir un comando UART, en caso de detectar actividad UART realiza la acción recibida según el comando UART en el cual el principal es la ejecución regular de la aplicación de usuario a partir de la dirección de memoria *0xC02*, si el gestor de arranque detecta actividad UART para programar la memoria Flash empieza una transmisión para recibir una aplicación de usuario desde el ordenador que coloque en parte de su memoria flash. Entonces, las aplicaciones que se programen en el dsPIC33F con el uso del gestor de arranque serán almacenadas desde la dirección *0xC02*. Esto implica que en el proceso de compilación debe realizar algunas alteraciones para que los archivos *.elf* o *.hex* cumplan con este requisito.

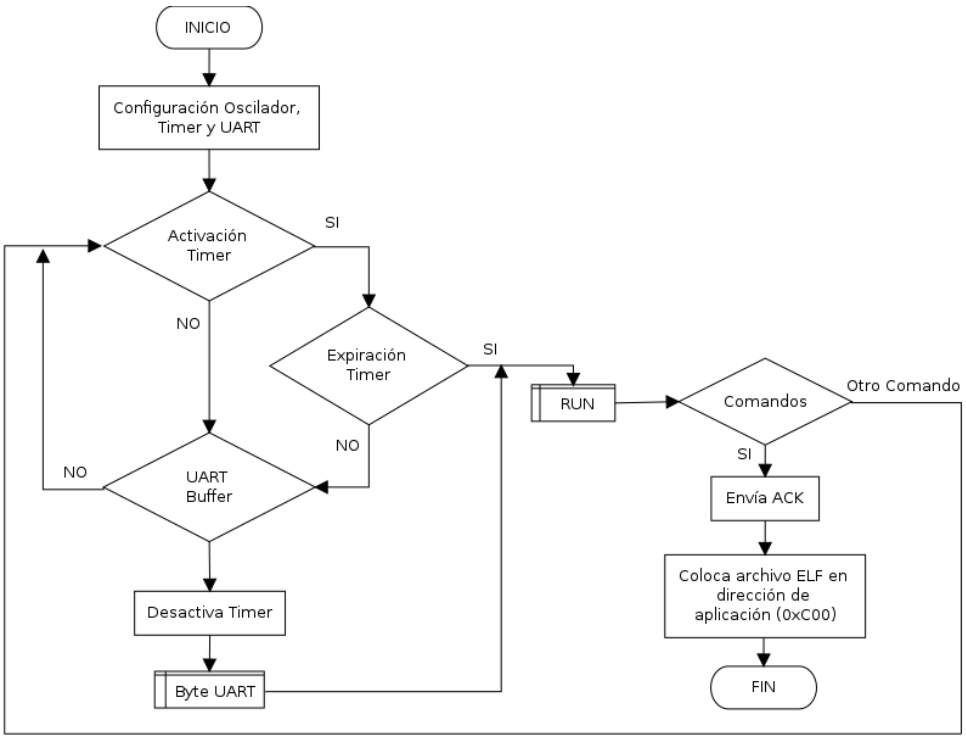


Figura 4.4: Diagrama de Flujo Interrupciones Gestor Arranque

Fuente: Elaboración Propia

El diagrama de la figura 4.4 representa el flujo del gestor de arranque una vez que todos los módulos de hardware hayan sido inicializados, un timer de conteo regresivo está siempre activo, una vez dicho timer haya expirado, la variable correspondiente se carga con una instrucción de ejecución (run) para que posteriormente el módulo intérprete de comandos salga del gestor de arranque a la aplicación del usuario. Si se recibe un comando UART el timer de conteo regresivo pasa a desactivarse y el comando se carga con el byte recibido para que el gestor de arranque no pueda salir sino solo hasta que la aplicación de gestor de arranque envíe un comando para la ejecución de aplicación de usuario.

En la figura 4.5 se muestra el diagrama de flujo para la interpretación de comandos ingresados por el usuario por intermedio de la MHI.

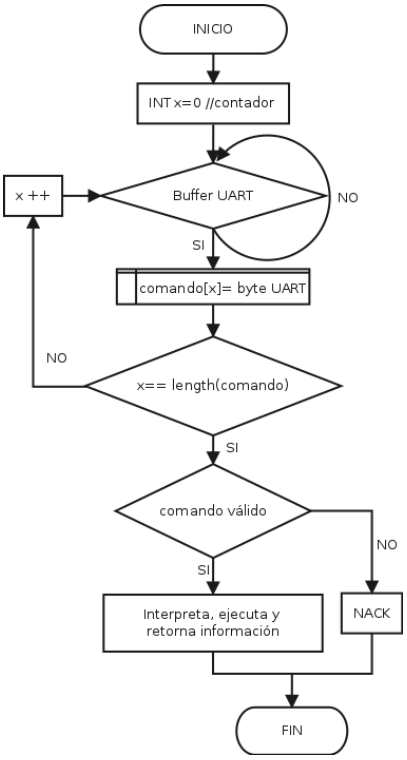


Figura 4.5: Diagrama de Flujo Interpretación Comandos

Fuente: Elaboración Propia

A partir de la figura 4.5 se toma como referencia para este desarrollo, el MHI interpreta y ejecuta las siguientes sentencias de códigos:

Tabla 4.2: Comandos seriales desde la MHI

Comando	Formato	Retorno	Interpretación
readPM	[0x02],[Dirección]	Contenido de la página de memoria	512 ubicación * 3 bytes/ubicación = 1536 bytes
writePM	[0x03],[Dirección], [Nueva página memoria]	ACK	Borra toda la página y escribe nuevos datos.
runProg	[0x08]	ACK	Inicia automáticamente la aplicación del usuario.
readID	[0x09]	ACK	Lee si el cargador de arranque está listo para comunicarse.
erasePM	[0x0a]	ACK	No permite borrar páginas 0x400 a 0x800.
testDevice	[0x0b]	ACK	Prueba físicamente la tarjeta, útil para uso de varias tarjetas.

Fuente: Elaboración Propia

Se debe tomar en cuenta que la lectura, escritura y borrado no se pueden realizar por bytes sino solo por página ya que es una especificación de fabricante. De una manera más específica una página tiene 512 ubicaciones de memoria de 4 bytes de las cuales cada una tiene 24 bits útiles de los 32 bits que especifica el fabricante. Por otra parte, cuando un comando no es reconocido o a su vez se reconoce, pero el argumento no es válido el gestor de arranque responde con un NACK.

4.3.2. Desarrollo Puente USB/UART

Este pequeño programa que actúa conjuntamente con el módulo interno USB del PIC18F. Posee una frecuencia de oscilación de 48 Mhz que es requerida por USB2.0 y aprovechando que la tarjeta FLEX-Full que tienen incorporada un cristal de 20 MHz. Este PIC18F tiene la ventaja de suministrar las bibliotecas necesarias para el ámbito de comunicación USB que le da mucha comodidad para transferencias de datos donde el tiempo es muy crítico.

El gestor de arranque sobre el dsPIC33F hace uso de las direcciones de memoria desde la *0x400* a *0xBFF* y utiliza los periféricos UART y TIMER, también hace uso de los vectores de reinicio que constan en la tabla de vectores de interrupción (IVT). Según el datasheet de fabricante el rendimiento máximo medido para un gestor de arranque es de 8.1 KB/s.

Como se muestra en el diagrama de flujo de la figura 4.6, se inicializan los módulos de hardware y esperar a detectar actividad UART para poder establecer las interrupciones, con ello se podría saber si existe flujo de información.

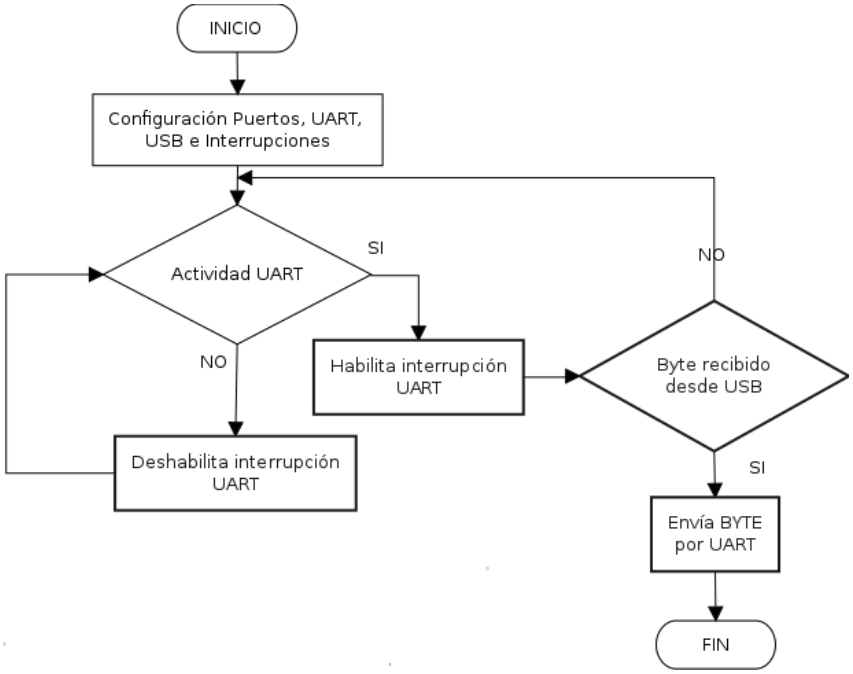


Figura 4.6: Diagrama de Flujo Puente USB/UART

Fuente: Elaboración Propia

Para el restablecimiento del dsPIC33F una salida está conectada al pin maestro para que pueda ser reiniciado antes de iniciar la comunicación con el gestor de arranque, para el UART está configurado a 115200 bps, 8 bits de largo, sin paridad, sin control de flujo y para las interrupciones tiene ajustes de eventos USB y recepción UART.

En esta parte el puente USB/UART también es el encargado de realizar un subproceso de enumeración, para lo cual el microcontrolador debe responder a varias ”preguntas” (ID de fabricante, requisitos de alimentación, versión del dispositivo y USB que soporta) que son emitidas por la MHI con todo esto la tarjeta FLEX-Full podrá ser identificado de otros en caso de que se conecte más de uno. Entonces, las funciones básicas del puente USB/UART son que cada byte que se recibe desde el ordenador mediante el USB se reenvía a través del UART hacia dsPIC33F esto es realizado por el bucle principal del programa y cada byte se recibe del dsPIC33F a través del UART se reenvía al ordenador mediante USB, proceso encargado una

subrutina de interrupción.

4.3.3. Desarrollo Interfaz Humano Máquina

La MHI analiza el archivo *.elf* de la aplicación de usuario y copia en parte de la memoria flash del dsPIC33F a través de UART, pudiendo también tener un nivel de gestión en la lectura de las direcciones de memoria de programa. Se toma el ejemplo de la interfaz CLI del software AVRDUDE para dar la idea más acertada para el desarrollo de la MHI como se muestra en la figura 4.8.

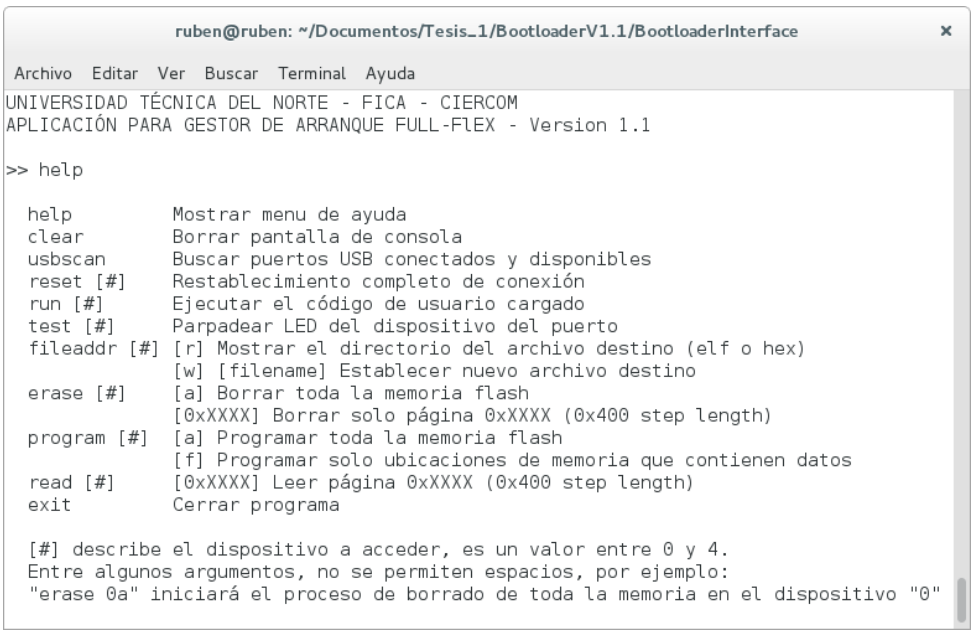


Figura 4.7: Interfaz de línea de comandos

Fuente: Elaboración Propia

Para poder identificar el proceso que realiza la MHI se realiza un diagrama de flujo donde muestra el funcionamiento de las partes más importantes como podemos evidenciar en la figura 4.8.

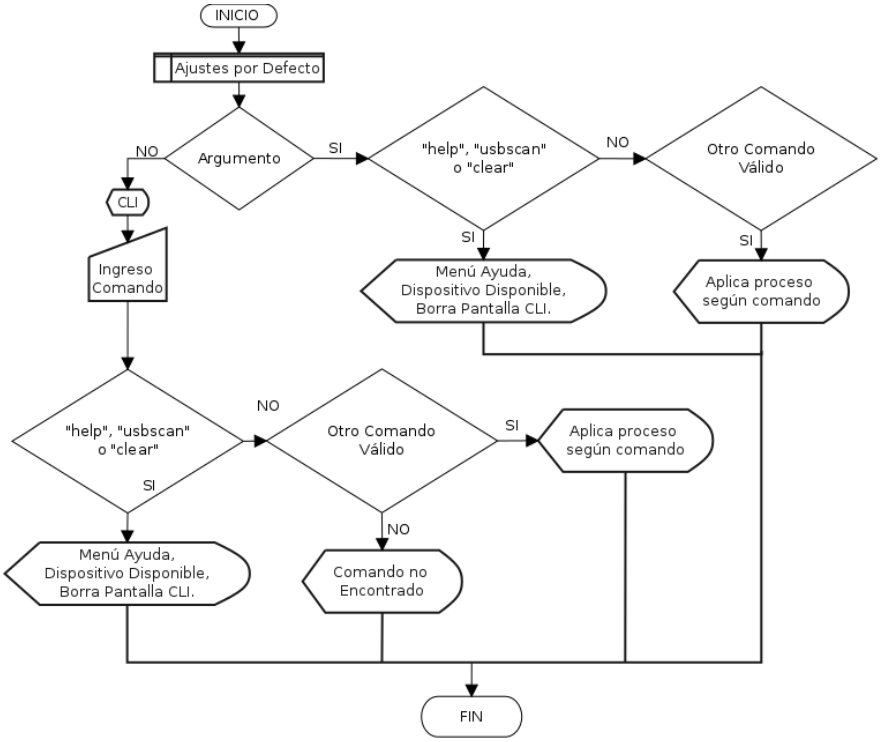


Figura 4.8: Diagrama de Flujo MHI

Fuente: Elaboración Propia

Según la figura 4.8, el primer proceso que realiza es cargar ajustes en caso de que se hayan almacenado con anterioridad, en caso de serlo carga un script donde se almacenan los directorios de puertos con la respectiva ubicación de los archivos *.elf o .hex* que hayan sido programados con anterioridad para ser grabados en la FLEX-Full. Si la MHI comprueba que existen argumentos pasa a ejecutarse los procesos según la tabla 4.1 caso contrario se queda en un blucle infinito hasta ingreso de argumentos por teclado.

Capítulo 5

Pruebas de Funcionamiento

En este capítulo se culmina las últimas subsecciones de la metodología para el desarrollo del gestor de arranque. Se inicia con la descripción del funcionamiento completo de los desarrollos realizados para alcanzar el objetivo de ayudar al gestor de arranque programe la memoria del dsPIC33F. Se evidencia con la programación de un archivo *.elf* o *.hex* en la memoria flash del mismo. Finalmente, para el producto final y mediante un ejemplo se podrá verificar el correcto funcionamiento del gestor de arranque.

5.1. Integración

En esta subsección se realiza la explicación de las configuraciones tanto de hardware (tarjeta Full-Flex) como de software (MPLAB IDE) para poder realizar la programación del código de usuario en el dsPIC33F, en la figura 5.1 se muestra el diagrama de arquitectura del dsPIC33F en la sección con la que interactúa la interfaz humano máquina.

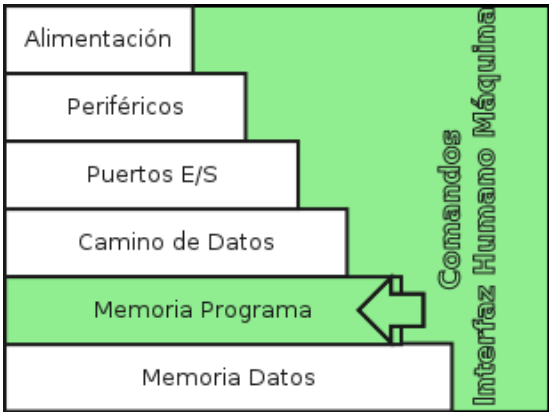


Figura 5.1: Arquitectura básica dsPIC33FJ para acceso de MHI

Fuente: http://bit.ly/30iJ2Ay p.40

5.1.1. Visión General

El gestor de arranque para proporcionar la programación en el dsPIC33F se apoya de dos partes fundamentales, la MHI y puente USB/UART como se puede evidenciar en la figura 5.2.

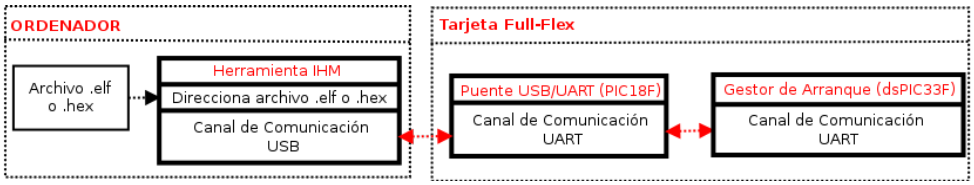


Figura 5.2: Diagrama de Básico Proceso Programación

Fuente: Elaboración Propia

Para que todo esto suceda se utiliza las siguientes herramientas de software y sistema operativo que garantizan el correcto funcionamiento del conjunto de desarrollos con sus respectivas versiones:

Tabla 5.1: Versiones de Herramientas de Software

Herramienta Software	**Versión**
Python	2.7.9
DEBIAN	Jessie 8
MPLAB X	2.0.0
MPLAB XC16 C Compiler	1.20

Fuente: Elaboración Propia

Los usuarios mediante la terminal de consola deben escribir los comandos para ejecutar la MHI y posterior la programación del dsPIC33F para ello se debe realizar configuraciones previas tanto en hardware como en software.

5.1.2. Configuración de Hardware

Esta tarjeta para el fin descrito tiene la particularidad de tener ciertas conexiones para que el gestor de arranque pueda ser programado y a su vez permita la comunicación entre el PIC18F y el dsPIC33F en el momento de la programación de una aplicación de usuario siendo transparente para el desarrollador. La figura 5.3 trata de evidenciar la conexión de los puentes que permite la comunicación de puertos UART del dsPIC33F y PIC18F. También la conexión de los jumper (color naranja) que permita la configuración correcta según el datasheet de la tarjeta FLEX-Full (EVIDENCE, p.18). Entre las principales configuraciones que permiten los jumpers son:

- LEDs indicadores de energía cuando hay una fuente de alimentación conectada.
- LED indicador controlado por dsPIC.
- Selecciona fuente de alimentación interna para PIC18.
- Conecta Master Clear a dsPIC.
- Puerto USB conectado al dsPIC.

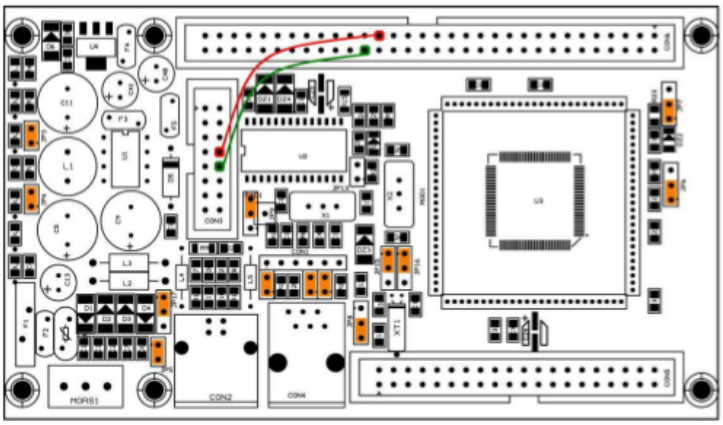


Figura 5.3: Configuración UART PIC18F y dsPIC33F

Fuente: Elaboración Propia

Si se desea saber los pines correspondientes a cada ”CON” de la tarjeta FLEX-Full dirigirse al Anexo 3.

5.1.3. Configuración de Software

5.1.3.1. Gestor de Arranque

Se debe aclarar que esta acción se la debe realizar por una única vez antes de pasar a la configuración del archivo enlazador de aplicaciones (.gld), consiste en que los archivos que contienen los códigos del gestor de arranque y del puente USB/UART (*.hex*) se debe programar de la forma convencional (uso de hardware externo) y directamente en cada puerto de la MCU correspondiente a la tarjeta FLEX-Full.

5.1.3.2. Aplicaciones de Usuario

Según el datasheet menciona los requisitos que se aplican a cualquier aplicación de usuario que el gestor de arranque realice la programación, el código de las aplicaciones de usuario deben caber en el espacio de memoria designado y no pueden ser colocadas en el espacio de

memoria que está reservado para el gestor de arranque y el retraso del cargador de arranque se debe especificar para las posteriores ejecuciones del gestor de arranque, para ello, la secuencia de comandos del enlazador de la aplicación (archivo .gld) debe modificarse para especificar la dirección de la aplicación de usuario y designar el periodo de retraso del gestor de arranque. Para los dispositivos dsPIC33F, el archivo *.gld* se modifica para colocar la aplicación del usuario en la dirección *0x0C02* y proporcione un valor de tiempo de espera para el cargador de arranque.

```
program (xr) : ORIGIN = 0xC00, LENGTH = 0x29E00
__CODE_BASE = 0xC00; /* Handles, User Code, Library Code */

/*
** User Code and Library Code
*/
.text __CODE_BASE :
{
   SHORT(0x0A); /* Bootloader timeout in sec */
   *(.handle);
   *(.libc) *(.libm) *(.libdsp);
   *(.lib*);
   *(.text);
} >program
```

Figura 5.4: Configuración Archivo *.gld* MPLAB IDE

Fuente: http://bit.ly/30kIIRF p.6

Para ello debemos dirigirnos y editar el archivo *p33FJ256MC710.gld* que se encuentra en el siguiente directorio */opt/microchip/xc16/v1.20/support/dsPIC33F/gld/* y cambiar por las líneas de código según la figura 5.4.

5.2. Pruebas

En esta subsección una vez que se haya configurado de manera correcta las partes de todas las secciones 5.1 se puede proseguir hacer uso del conjunto de desarrollos para realizar las pruebas correspondientes de la lectura, escritura y borrado de archivos *.elf* de la memoria flash del dsPIC33F, esto ayuda a poner en evidencia como el gestor de arranque actúa de manera conjunta para dicho objetivo. En la figura 5.5 se presenta la conexión directa entre la tarjeta

FLEX-Full y un ordenador después de seguir las configuraciones antes mencionadas.

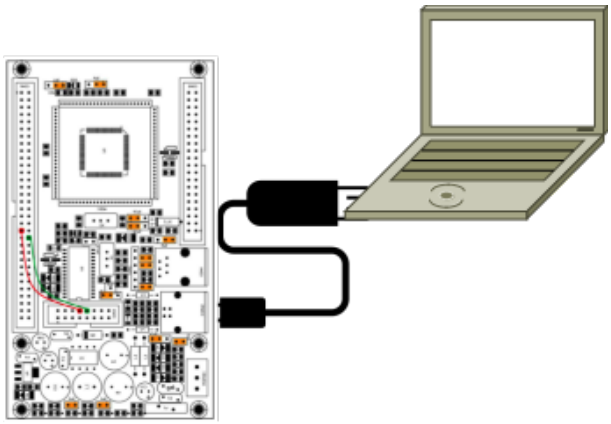


Figura 5.5: Conexión Tarjeta FLEX-Full-Ordenador

Fuente: Elaboración Propia

5.2.1. Interpretación de Comandos

La serie de comandos compuestos por argumentos hace posible que las aplicaciones de usuario se graben en la memoria flash del dsPIC33F a continuación se da una explicación detallada del uso de los comandos de la MHI.

5.2.2. Ingreso a MHI

Mediante la terminal de consola existen dos maneras de acceder a la MHI: La primera es a través del ingreso de los comandos con sus respectivos argumentos en la cual se necesita ejecutar el script principal llamado *serialBoot.py* con el siguiente formato *[programa] [script][-abreviación comando]* como se muestra en la figura 5.6.

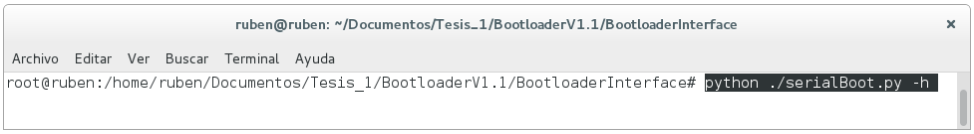


Figura 5.6: Método abreviado de ejecución MHI

Fuente: Elaboración Propia

De esta manera se ejecuta lo solicitado (despliegue menú de ayuda) ingresada y una vez procesado el script se cierra como se puede evidenciar en la figura 5.7.

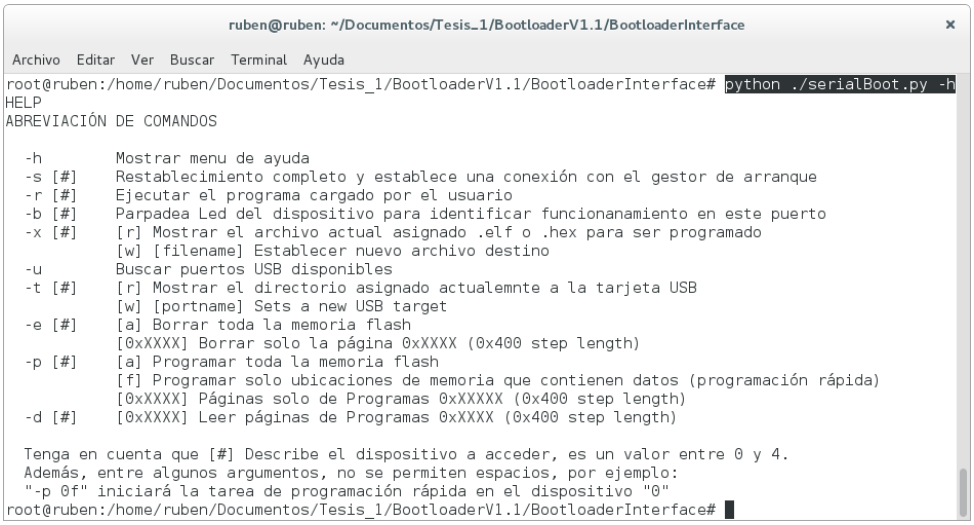


Figura 5.7: Menú de opciones método abreviado de ejecución MHI

Fuente: Elaboración Propia

La segunda manera de ejecución denominada método bucle indeterminado la que es mediante el formato de comandos *[programa] [script]* para la terminal de consola, se mantiene en ejecución hasta introducir la sentencia apropiada de comandos para salir de la MHI. En la figura 5.8 se muestra la ejecución de la sentencia del comando para ingresar al script y ejecutar un ”bucle” indeterminado para el procesamiento de los comandos después ingresados por consola las veces que el usuario lo considere.

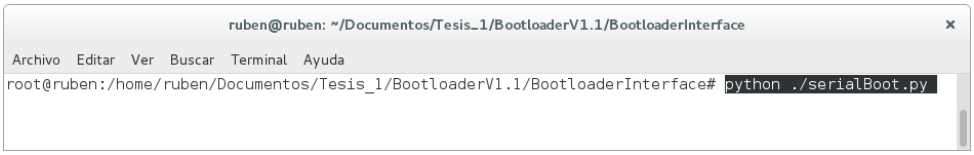


Figura 5.8: Método Bucle Indeterminado

Fuente: Elaboración Propia

La figura 5.9 detalla la salida de pantalla del script principal una vez ejecutado el comando *help*.

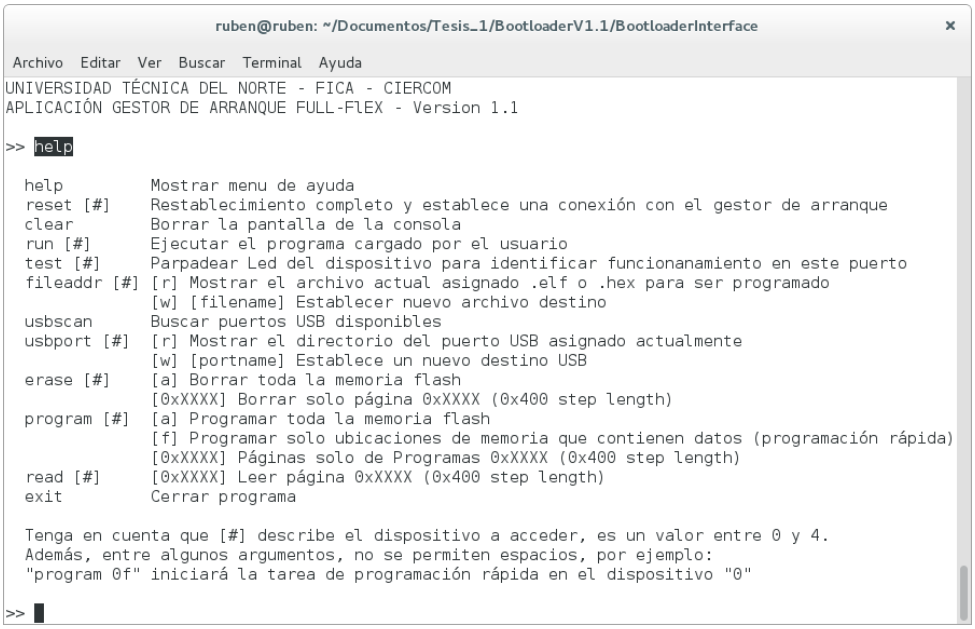


Figura 5.9: Menú Ayuda Método Bucle Indeterminado MHI

Fuente: Elaboración Propia

A continuación, se dará una explicación del uso de los comandos existentes con sus respectivas abreviaciones:

- ***clear:*** borra toda la pantalla del terminal de consola.
- ***exit***: termina la ejecución de la MHI.
- ***h:****help*, muestra el menú de ayuda.

Para los comandos *usbscan,reset,test* el método abreviado tiene el formato *[programa] [-comando] [argumento]* y el método bucle indeterminado tiene formato *[comando] [argumento]* como se puede evidenciar en las figuras 5.10 y 5.11 respectivamente.

Figura 5.10: Método abreviado para escaneo de puertos

Fuente: Elaboración Propia

- ***s:****reset*, restablece por completo la conexión con el gestor de arranque.
- ***r:****run*, ejecuta el programa cargado por el usuario.
- ***b:****test*, parpadea Led del dispositivo para identificar funcionamiento en el puerto.
- ***u:****usbscan*, busca puertos USB disponibles, muestra cuántas tarjetas Full-Flex están conectadas al ordenador (máximo 5) y en qué puerto USB se encuentra. En este caso, existe una placa en */dev/ttyACM0* como se muestra en la figura 5.11.

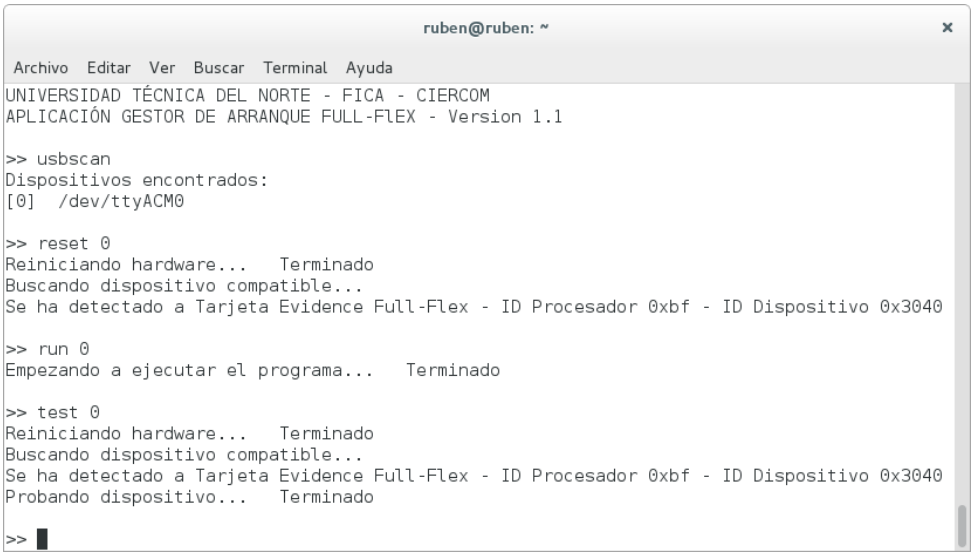


Figura 5.11: Método abreviado comandos básicos

Fuente: Elaboración Propia

- ***x:****fileaddr*, [r] muestra el directorio del archivo *.elf* o *.hex* para ser programado, [w] [filename] establece nuevo directorio del archivo *.elf* o *.hex*.

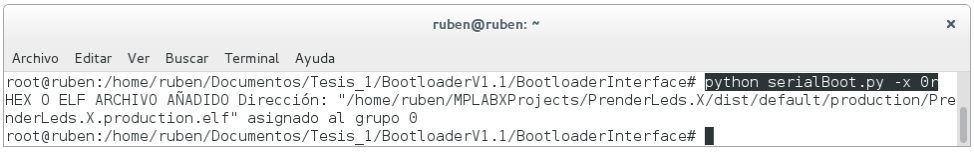


Figura 5.12: Método abreviado para mostrar directorio

Fuente: Elaboración Propia

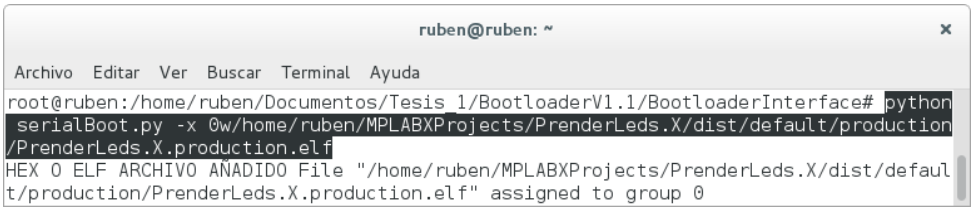


Figura 5.13: Método abreviado para nuevo directorio

Fuente: Elaboración Propia

- ***e:****erase*, [a] borra toda la memoria flash, [0xXXXX] borra solo la página 0xXXXX (0x400 step length).

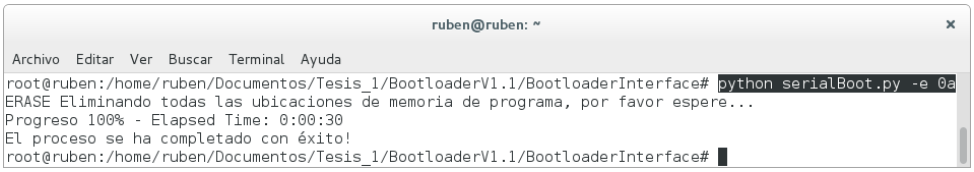


Figura 5.14: Método abreviado borrado de memoria Flash

Fuente: Elaboración Propia

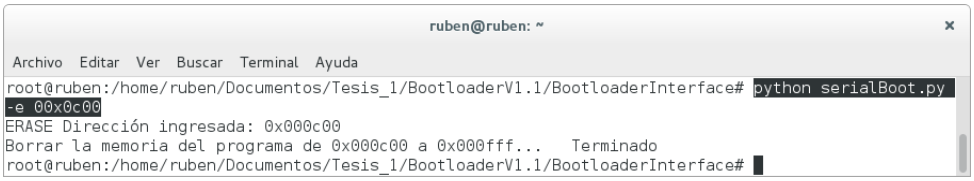


Figura 5.15: Método abreviado borrado de páginas memoria Flash

Fuente: Elaboración Propia

- ***p:****program*, [a] Programar toda la memoria flash.

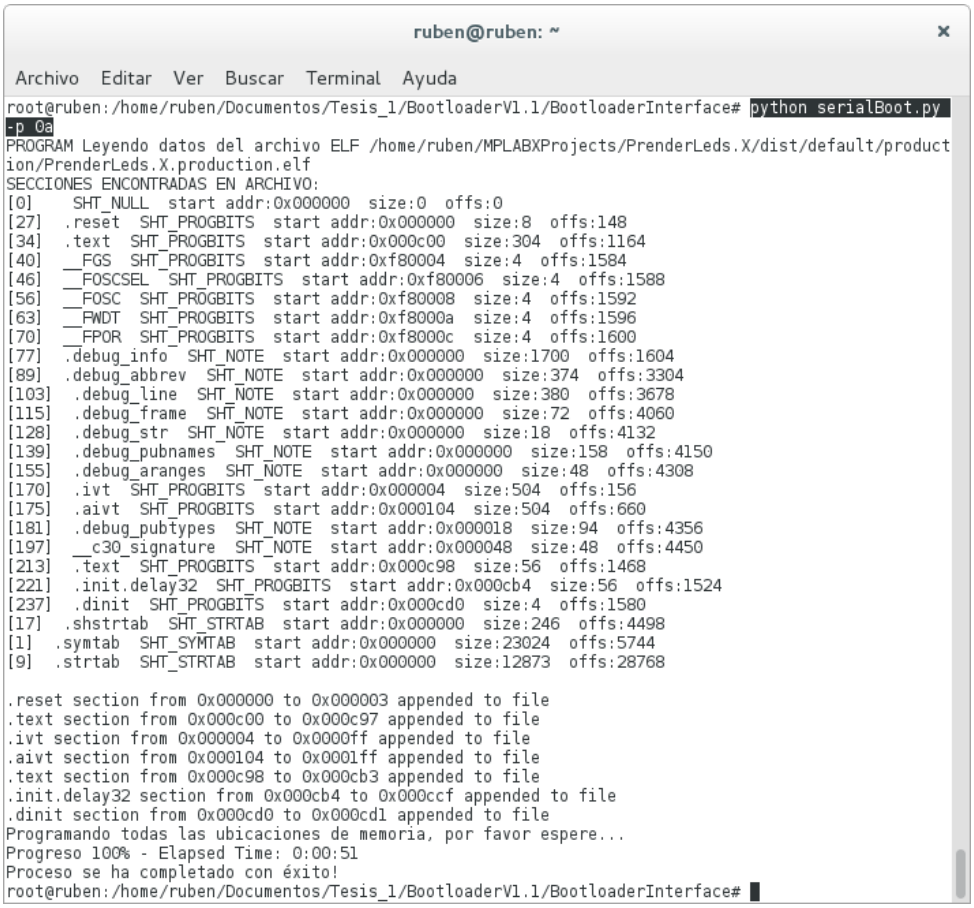


Figura 5.16: Método abreviado para programación de memoria flash

Fuente: Elaboración Propia

- ***d:****read*, lee páginas de programas 0xXXXX (0x400 step length).

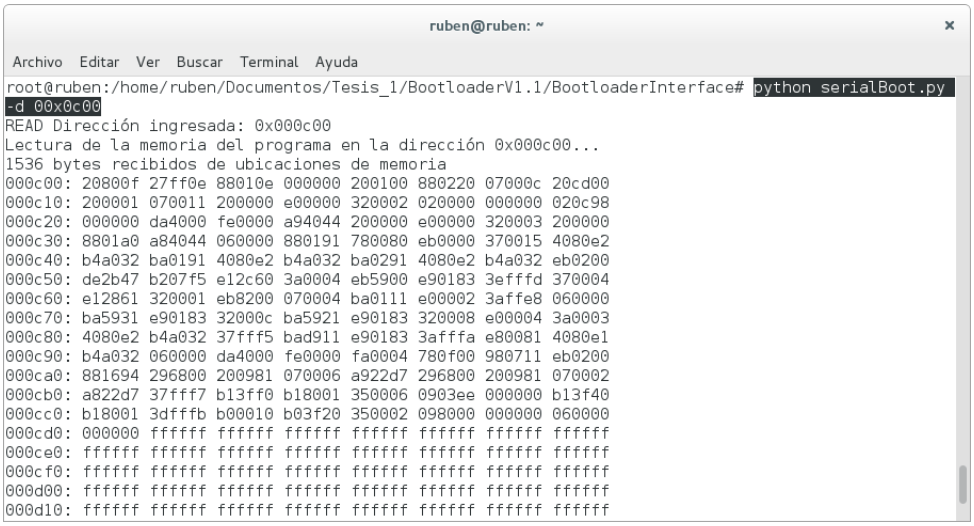


Figura 5.17: Método abreviado para lectura de memoria Flash

Fuente: Elaboración Propia

5.3. Producto de Ingeniería

Para esta parte se tiene ejemplos de aplicaciones de usuario para poder hacer uso de la MHI conjuntamente con el gestor de arranque sobre los microcontroladores de la tarjeta FLEX-Full con lo que también ayudará a consolidar las pruebas a la que fue sometida. Para ello tenemos los siguientes ejemplos los cuales no tienen el objetivo de mostrar cual es el proceso para la programación de microcontroladores en lenguaje C sobre la plataforma MPLAB IDE si no el proceso de programación para el dsPIC33F, en la figura 5.18 se muestra la orientación y distribución de pines de la tarjeta FLEX-Full.

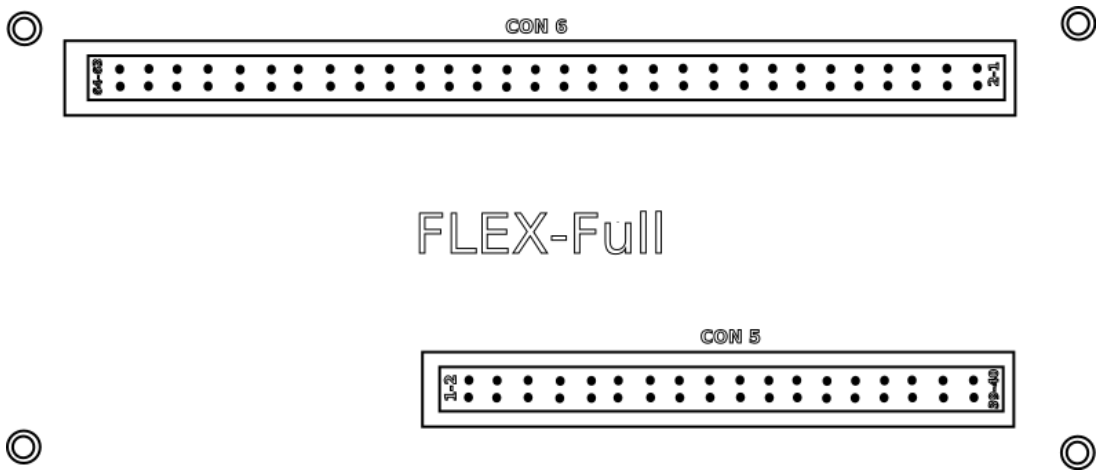


Figura 5.18: Distribución de pines CON5 - CON6 FLEX-Full

Fuente: Elaboración Propia

5.3.1. Ejemplos de aplicaciones de usuario

Para cualquier aplicación de usuario y para los ejemplos que se muestra en la sección 5.2.1 se debe obtener los archivos .elf o .hex generados por MPLAB IDE una vez realizado la configuración que se menciona en la sección 5.1.4. para dicha plataforma.

5.3.1.1. Ejemplo 1: Manejo de Interrupciones Externas

El código se encuentra disponible en el Anexo 4 en el cual se propone encender/apagar un LED mediante un bucle indeterminado con un retardo de 5000ms, el LED debe estar conectado en el Pin11 del CON5 y mediante el uso de la interrupción externa *INT0* que corresponde al Pin43 del CON6, se cambiar el retardo de encendido/apagado del LED a 500ms, se restablece mediante un RESET que se conecta en el Pin13 del CON6 tomando como guía el siguiente diagrama de la figura 5.19:

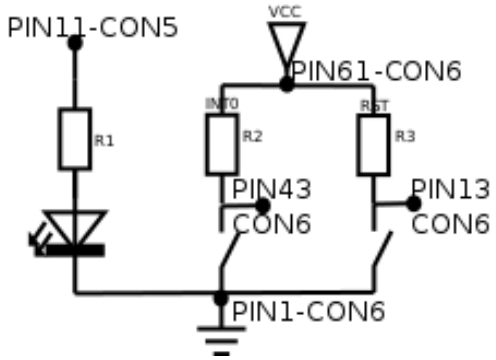


Figura 5.19: Diagrama para interrupciones externas

Fuente: Elaboración Propia

5.3.1.2. Ejemplo 2: Manejo de LCD

En primera instancia como información adicional a este ejercicio se hará un ejemplo del manejo básico de un LCD 16x2 usando el dsPIC33f. El manejo del LCD con el dsPIC se centra en estáblecer la comunicación entre el microprocesador Hitachi 44780 o compatibles, este microprocesador se encarga de la escritura en pantalla del LCD. Para la comunicación con el LCD se ha desarrollado la librería *lcd.h* en el Anexo 5 En la tabla 5.2. se muestra los pines de la tarjeta FLEX-Full correspondientes para la LCD.

Tabla 5.2: Tabla Conexiones Ejemplo 2

Puerto, PIN	**LCD**
CON5, 2	VCC
CON5, 3	GND
CON6, 36	RS
CON5, 20	RW
CON5, 17	E
CON5, 27	D0
CON5, 30	D1
CON5, 34	D2
CON5, 33	D3
CON5, 36	D4
CON5, 38	D5
CON5, 37	D6
CON5, 40	D7

Fuente: Elaboración Propia

5.3.2. Resultados

Para determinar resultados del uso para el gestor de arranque se realizó un taller con parte demostrativa y práctica a un grupo de 16 estudiantes pertenecientes entre quinto a séptimo nivel de la Facultad FICA y carrera CIERCOM los que en cuyos niveles toman las cátedras de Sistemas Microprocesados y Sistemas Embebidos para lo cual ya tienen conocimientos teóricos y prácticos que ayuda a que el taller tenga mejor sustento y dinámica para poder obtener resultados muy cercanos a las necesidades en los laboratorios donde se utiliza microcontroladores. Se utilizó una hoja guía para este taller mismo que se puede evidenciar en el Anexo 6.

Posteriormente al taller práctico se realizó una en cuenta con 10 preguntas cerradas y 2 mixtas a los estudiantes que participaron en el taller con el fin de generar información concreta sobre el tema planteado y respaldar a que gran parte de la problemática pueda ser cubierta y tener mejor sustento en los resultados y concluir acerca del mismo. Si se desea observar la encuesta y los resultados (tabulación) dirigirse al Anexo 7 y 8 respectivamente.

De manera general podemos mencionar que mediante las preguntas realizadas en la encuesta se hace notorio que el 81 % de los estudiantes encuestados tienen conocimientos acerca el uso de hardware externo para la programación de microcontroladores los cuales el 38 % han tenido inconvenientes con el software, 56 % con el hardware de programación y el 6 % no han tenido ningún inconveniente. Por otro lado, el 100 % de los estudiantes encuestados creen el factor económico es determinante en las universidades públicas para lo cual el 56 % cree que cada estudiante debe asumir los gastos que se generen en laboratorios donde se haga uso de microcontroladores y por ende de programadores con lo cual el 81 % creen que por esa razón el desempeño académico del estudiante se ve afectado.

Tabla 5.3: Resultados de la Encuesta (Preguntas 1-5)

Pregunta	**Resultado (%)**
1	SI=81 %, NO=19 %
2	a=38 %, b=56 %, c=0 %, d=0 %, e=6 %
3	SI=100 %, NO=0 %
4	SI=56 %, NO=44 %
5	SI=81 %, NO=19 %

Fuente: Elaboración Propia

El 25 % de los encuestados mencionan que tienen o han tenido en cada semana de clases regulares entre 1 a 2 horas de prácticas de laboratorio referente a microcontroladores, el 44 % entre 2 a 4 horas y otro 25 % más de 4 horas en las cuales el 56 % coincide que para un mismo día de laboratorio pueden tener entre 1 a 2 prácticas y así el 100 % de encuestados está de acuerdo que la cantidad de elementos que podrían conformar el armado de una práctica es un limitante para completar el objetivo de la misma.

Tabla 5.4: Resultados de la Encuesta (Preguntas 6-8)

Pregunta	**Resultado (%)**
6	a=25 %, b=44 %, c=25 %, d=6 %
7	a=56 %, b=38 %, c=6 %, d=0 %
8	SI=100 %, NO=0 %

Fuente: Elaboración Propia

Para la última parte de la encuesta que hace referencia a los gestores de arranque el 81 % de los encuestados no han escuchado ni tiene conocimiento acerca este tipo de soluciones de programación para MCUs pero el 100 % cree que este método alternativo ayudaría para que la disponibilidad de programadores de MCUs aumente al igual reduzca costos de inversión del estudiante para este tipo de prácticas y el mismo porcentaje de estudiantes está de acuerdo que este método también ayudaría al desarrollo académico en laboratorios de microcontroladores como tambíen el 94 % de estudiantes cree que ayudaría a disminuir tiempos de armando para

las prácticas.

Tabla 5.5: Resultados Encuesta (Preguntas 9-12)

Pregunta	**Resultado (%)**
9	SI=19 %, NO=81 %
10	SI=100 %, NO=0 %
11	SI=94 %, NO=6 %
12	SI=100 %, NO=0 %

Fuente: Elaboracón Propia

Conclusiones

Los requerimientos y funcionalidades consideradas para este desarrollo fueron concretas para no perder el enfoque del objetivo principal de leer, borrar y escribir en la memoria flash de una MCU.

El uso de la metodología Botton-Up fue adaptada para el desarrollo consecutivo del gestor de arranque de forma que abarcó desde la toma de requisito y funcionalidades hasta la presentación del producto final.

Con la utilización de librerías de Python en la MHI se pudo lograr la decodificación de los archivos *elf* y *hex* y la comunicación serial hacia el dsPIC33F haciendo actuar al PIC18F2550 como puente USB/UART de manera transparente para el usuario.

El taller práctico realizado a los estudiantes encuestados arrojó como resultado principal que la forma de programación propuesta cubre en gran medida la falta de grabadores y ayuda a reducir tiempos en armados de prácticas para ser más factible lograr los objetivos de clase.

Recomendaciones

Para el uso de esta alternativa de programación el usuario debe tener conocimientos básicos en el uso de comandos para la terminal de consola en GNU/LINUX, programación en lenguaje C y Python para una adecuada interpretación del proceso de programado de la MCU.

El gestor de arranque puede servir como pauta para realizar adaptaciones para MCUs de otras gamas que satisfagan las necesidades de usuario.

Se debe tomar en cuenta que los desarrollos realizados fueron bajo el concepto de GNU/-LINUX lo que abre la posibilidad de crear una MHI de interfaz gráfica como también extender este desarrollo para sistemas operativos licenciados.

Se una la tarjeta Full-Flex para optimizar y enfocar el tiempo de desarrollo en el gestor de arranque, MHI y puente USB/UART, lo que deja abierto la posibilidad de crear una tarjeta con un conjunto de electrónica que incluya la misma lógica (dos MCUs).

Bibliografía

[1] Cucho, Z., Orihuela, F., Sánchez, R., y Rodríguez, L. (Dic. 2006). *Microcontroladores Atmega8*. Pontificia Universidad Católica del Perú. Facultad de Ciencias e Ingeniería. Sección Electricidad y Electrónica. Área de Circuitos y Sistemas Electrolíticos, Visitado: May. 2018. Recuperado de: https://goo.gl/vEmRmL, pp.3-4.

[2] González, J. y Moreno, A. (Ene. 2004). *Herramientas hardware y software para el desarrollo de aplicaciones con Microcontroladores PIC bajo plataformas GNU/Linux*. Universidad Autónoma de Madrid. Ifara Tecnologías, Visitado: May. 2018. Recuperado de: https://goo.gl/Sf5t4t, pp. 2-3.

[3] Villalobos, F., Rodríguez, H., Molina, J., y Trejo, R. (Dic. 2006). *Boot Loader para Microcontroladores PIC serie 18*. Instituto Tecnológico de Aguascalientes. Conciencia Tecnológica, Visitado: May. 2018. Recuperado de: https://goo.gl/vYGwfX.

[4] Guadrón, R. y Guevara, J. *Implementación de Bootloaders en Microcontroladores PIC16 y PIC18 de Microchip Inc*. Escuela Especializada en Ingeniería ITCA-FEPADE. Revista Tecnológica, 7(1), Visitado: Jun, 2018. Recuperado de: https://goo.gl/WbPzYk, pp. 1-2.

[5] Torres, M. (Abr. 2007). *Tutorial Microcontroladores PIC*. Visitado: Jun. 2018. Recuperado de: https://goo.gl/yBQe2a, p.4.

[6] Herrán, A. (2011). *Técnicas didácticas para una enseñanza más formativa*. Estrategias y metodologías para la formación del estudiante en la actualidad. Visitado: Jun, 2018. Recuperado de: https://goo.gl/UYwJp6, p.31.

[7] Martínez, M., Marceleño, J., Govea, E., y Medina, R. (Sep. 2013). *Interfaz Gráfica con Microcontrolador PIC18f4550 para Automatización de Procesos*. Universidad Politécnica de San Luis. Visitado: Jun, 2018. Recuperado de: https://goo.gl/PG7Gos, p.784.

[8] Alley, P.(May. 2011). *Introductory Microcontroller Programming.* WORCESTER POLYTECHNIC INSTITUTE. Tesis PhD. Visitado: Jul, 2018. Recuperado de: https://goo.gl/mbE8Xb, pp. 2-3.

[9] UNED. (Ene. 2009). *Controladores industriales de diseño de alto nivel*. Departamento de Ingeniería de Eléctrica, Electrónica y Control. Visitado: Jul, 2018. Recuperado de: https://goo.gl/unYwxN, pp. 4-8.

[10] Zamora, F. (Sep. 2017). *Las mejores herramientas para empezar a desarrollar con PIC Micros*. Visitado: Jun, 2018. Recuperado de: https://goo.gl/BaVcvD, pp. 3-4.

[11] Camacho, A. (Dic. 2009). *Análisis, diseño e implementación de sistemas de control en red mediante un uso eficiente del ancho de banda*. Universidad Politécnica de Catalunya. Tesis de Grado. Visitado: Jul, 2018. Recuperado de: https://goo.gl/WxicLU, p.p 88-90

[12] Lewis, J. (Ene. 2016). *Arduino Bootloader, What is it? This simple code does important things*. BALDENGINEER. Visitado: Ene, 2019. Recuperado de: https://goo.gl/jQTah9.

[13] Savannah. (Jun. 2010). *AVRDUDE - AVR Downloader/UploaDEr*. Visitado: Ene, 2019. Recuperado de: http://bit.ly/2VUccUU.

[14] Electronicwings. (Ene. 2018). *Basics to Developing Bootloader for Arduino*. Visitado: Ene, 2019. Recuperado de: https://goo.gl/1of8u1.

[15] Angulo, J., Etxebarría, A., Angulo, I. y Trueba, I. (Jun, 2016). *dsPIC Diseño Práctico de Aplicaciones*. Ed(1), McGraw-Hill. Visitado: Feb, 2019. Recuperado de: https://goo.gl/iwcBYt. ISBN: 84-481-5156-9. p.p 6-10.

[16] Flex (Dic, 2012). *Modular soultion for embedded applications*. EVIDENCE. Visitado: Abr, 2019. Recuperado de: https://short1.link/Rh4bcE.

[17] Caicedo, C y Álvarez, E. (Jun, 2018). *PROTOTIPO ENFOCADO AL APRENDIZAJE DE LÓGICA DE PROGRAMACIÓN EN NIÑOS DE EDADES COMPRENDIDAS ENTRE 4 A 10 AÑOS*. Visitado: Jul, 2019. Recuperado de: http://bit.ly/2Gcd15k. p.p 50-54.

Anexos

.0.3. DataSheet Tarjeta de Desarrollo FLEX-Full

FLEX

Modular solution for embedded applications

version: 0.30
February 18, 2008

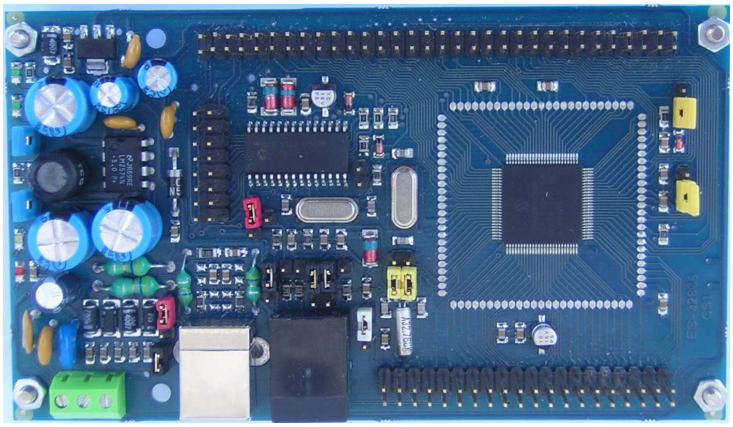


Figure 3.13: A typical jumper setting for programming the PIC18 on the FLEX Full.

JP5 (closed) - ground is connected to the PE reference;

JP6 and JP7 (1-2) - A/D converters use VDD and VSS as Vref+ and Vref-;

JP8 (1-2) - the voltage on the ICSP connector is 5 V;

JP9 (2-3) - the PIC18 power supply selects the internal power supply;

JP10 (2-3) - the Master Clear signal is connected to the PIC18;

JP11 (2-3) - the programming data line is connected to the PIC18;

JP12 (2-3) - the programming clock line is connected to the PIC18;

JP13 (open) - no RX serial port pull-up resistor;

JP15 and JP16 (2-3) - Real-Time Clock enabled;

JP17 (1-2) USB shield connected to GND.

3.2 FLEX pinout mapping

The next two tables shows the FLEX pinout mappings[1]

3.2.1 FLEX CON5 mappings

[1]Thanks to Andrea Bertelli for the Latex tables!

Pin	CON5 Mappings
1	$Vout$
2	$5V\ out$
3	Gnd_{out}
4	$3V\ out$
5	$INT3/RA14$
6	Gnd
7	$IC1/RD8$
8	$INT4/RA15$
9	$IC3/RD10$
10	$IC2/RD9$
11	$OC1/RD0$
12	$IC4/RD11$
13	$OC3/RD2$
14	$OC2/RD1$
15	$IC5/RD12$
16	$OC4/RD3$
17	$OC5/CN13/RD4$
18	$IC6/CN19/RD13$
19	$OC7/CN15/RD6$
20	$OC6/CN14/RD5$
21	$C1RX/RF0$
22	$OC8/UPDNCN16/RD7$
23	$C2TX/RG1$
24	$C1TX/RF1$
25	$AN22/CN22/RA6$
26	$C2RX/RG0$
27	$PWM1L/RE0$
28	$AN23/CN23/RA7$
29	$CSCK/RG14$
30	$PWM1H/RE1$
31	$CSD0/RG13$
32	$CSDI/RG12$
33	$PWM2H/RE3$
34	$PWM2L/RE2$
35	$COFS/RG15$
36	$PWM3L/RE4$
37	$PWM4L/RE6$
38	$PWM3H/RE5$
39	$AN16/T2CK/T7CK/RC1$
40	$PWM4H/RE7$

3.2.2 FLEX **CON6 mappings**

Pin	CON6 mappings
1	Gnd
2	Gnd
3	Gnd
4	Gnd
5	PGD2/EMUD2/SOSCI/CN1/RC13
6	PGC2/EMUC2/SOSCO/T1CK/CN0/RC14
7	AN17/T3CK/T6CK/RC2
8	AN18/T4CK/T9CK/RC3
9	AN19/T5CK/T8CK/RC4
10	SCK2/CN8/RG6
11	SDI2/CN9/RG7
12	SDO2/CN10/RG8
13	DSP_{MCLR}
14	SS2/CN11/RG9
15	TMS/RA0
16	AN20/FLTA/INT1/RE8
17	AN21/FLTB/INT2/RE9
18	AN5/QEB/CN7/CN7/RB5
19	AN4/QEA/CN6/RB4
20	AN3/INDX/CN5/RB3
21	AN2/SS1/CN4/RB2
22	$Vref-/RA9$
23	$Vref+/RA10$
24	$AVDD_{ext}$
25	$AVSS_{EXT}$
26	AN8/RB8
27	AN9/RB9
28	AN10/RB10
29	AN11/RB11
30	TCK/RA1
31	U2CTS/RF12
32	U2RTS/RF13
33	AN13/RB13
34	AN12/RB12
35	IC7/U1CTS/CN20/RD14
36	AN15/OCFB/CN12/RB15
37	U2RX/CN17/RF4

38	IC8/U1RTS/CN21/RD15
39	U1TX/RF3
40	U2TX/CN18/RF5
41	SDO1/RF8
42	U1RX/RF2
43	SCK1/INT0/RF6
44	SDI1/RF7
45	SCL1/RG2
46	SDA1/RG3
47	SDA2/RA3
48	SCL2/RA2
49	**TD0/RA5**
50	TDI/RA4
51	PGD3/EMUD3/AN0/CN2/RB0
52	DSP_{PCLK}
53	PGC3/EMUC3/AN1/CN3/RB1
54	DSP_{PDATA}
55	$5V_{out}$
56	$5V_{out}$
57	$3V_{out}$
58	$3V_{out}$
59	Gnd
60	Gnd
61	V_{out}
62	V_{out}
63	GND_{out}
64	GND_{out}

3.3 Daughter boards

A FLEX Daughter Board is a board with specialized features that can be inserted on top of a FLEX Base Board (piggybacking) or connected another Daughter Board, to obtain complex devices for all possible applications.

Evidence S.r.l. and Embedded Solutions S.r.l. propose a set of general purpose Daughter Boards for some most common applications.

The development of custom, home-made daughter boards is made easy since the FLEX Base Board connectors use the standard 2.54 mm pitch. Therefore, the features of the FLEX platform can be extended with virtually no limits.

.0.4. Manejo Interrupciones Externas (main.c)

Programa 1: Código implementado sobre microcontrolador dsPIC33FJ256MC

```
/*
 * File:   main.c
 * Author: ruben
 *
 * Created on 30 de julio de 2019, 10:09 AM
 */

#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#if defined(__dsPIC33F__)
#include "p33Fxxxx.h"
#elif defined(__PIC24H__)
#include "p24Hxxxx.h"
#endif

//######################Archivos Cabecera#################
#define FCY             20000000
#define BRGVAL          10      //((FCY/BAUDRATE)/16)-1
#define delayBegin      2       //reset after reboot in seconds
#include "libpic30.h"

//#########################DEFINIR MACROS###################
_FOSCSEL(FNOSC_FRC & FNOSC_PRIPLL); // Clock Switching and Fail Safe Clock Monitor is disabled
                        // Primary (XT, HS, EC) Oscillator with PLL
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF & POSCMD_XT);  //Frecuencia de Oscilacion
_FWDT(FWDTEN_OFF);                  //WatchDog time
_FPOR(FPWRT_PWR1 & FPWRT_PWR16); //Reinicio
_FGS(GWRP_OFF);             //Seguridad
```

```
/*
 *
 */
//Declaracin de la subfuncin interrupcin 0
/* Global Variables and Functions */
void INTx_IO_Init(void);
void __attribute__((__interrupt__)) _INT0Interrupt(void); /*Declare external interrupt ISRs*/

//*************** PROGRAMA PRINCIPAL*******************//
int main (void)
{
// Configure Oscillator to operate the device at 40Mhz
// Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
// Fosc= 8M*40/(2*2)=80Mhz for 8M input clock
        PLLFBD=38;                      // M=40
        CLKDIVbits.PLLPOST=0;           // N1=2
        CLKDIVbits.PLLPRE=0;            // N2=2
        OSCTUN=0;                       // Tune FRC oscillator, if FRC is used

// Disable Watch Dog Timer
    RCONbits.SWDTEN=0;

// Configure the Analog functional pins as digital
        AD1PCFGL=0xFFFF;
        AD1PCFGH=0xFFFF;

// Clock switch to incorporate PLL
        __builtin_write_OSCCONH(0x03);  // Initiate Clock Switch to Primary
                                        // Oscillator with PLL (NOSC=0b011)
        __builtin_write_OSCCONL(0x01);  // Start clock switching
        while (OSCCONbits.COSC != 0b011);       // Wait for Clock switch to occur

// Wait for PLL to lock
        while(OSCCONbits.LOCK!=1) {};
INTx_IO_Init();
```

```
TRISD=0x0000;
//Habilita la INT0
//INTCON2bits.INT0EP=1;
//Habilita la INT0
//IEC0bits.T1IE=1;

while(1) //Ciclo infinito while para programa principal
{
LATDbits.LATD8=0;
// Retardo de 50000 ciclos de maquina
__delay_ms(5000);
LATDbits.LATD8=1;
// Retardo de 50000 ciclos de maquina
__delay_ms(5000);
LATDbits.LATD8=0;
}
}

void INTx_IO_Init(void)
{
        INTCON2 = 0x001E;       /*Setup INT1, INT2, INT3 & INT4 pins to interupt */
                                /*on falling edge and set up INT0 pin to interupt */
                                /*on rising edge */
        IFS0bits.INT0IF = 0;    /*Reset INT0 interrupt flag */
        IEC0bits.INT0IE = 1;    /*Enable INT0 Interrupt Service Routine */
}

void __attribute__((interrupt, no_auto_psv)) _INT0Interrupt(void)
{
    IFS0bits.INT0IF = 0;    //Clear the INT0 interrupt flag or else, Limpia la bandera de la
        INT0
                            //the CPU will keep vectoring back to the ISR
    while (1)
    {
```

```
        LATDbits.LATD8=0;
        __delay_ms(500);      // Retardo de 500 ciclos de maquina
        LATDbits.LATD8=1;
        __delay_ms(500);
        LATDbits.LATD8=0;
    }
}
```

.0.5. Manejo LCD 16x2 (main.c)

Programa 2: Código implementado sobre microcontrolador dsPIC33FJ256MC

```
/************************************************************************
*   2007 PicMana By RedRaven
*        http://picmania.garcia-cuervo.net
*
* FileName:         main_LCD_Hello_World.c
* Dependencies:     p33FJ256GP710.h
* Processor:        dsPIC33F
* Compiler:         MPLAB C30 v2.01 or higher
*
* Present "Hello World" message
*
* C30 Optimization Level: -O1
*
************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <xc.h>
#include "lcd.h"
//#include "delay.h
#if defined(__dsPIC33F__)
```

```
#include "p33Fxxxx.h"
#elif defined(__PIC24H__)
#include "p24Hxxxx.h"
#endif

//######################Archivos Cabecera#################
#define FCY             20000000
#define BRGVAL          10      //((FCY/BAUDRATE)/16)-1
#define delayBegin      2       //reset after reboot in seconds
#include "libpic30.h"

//#########################DEFINIR MACROS###################
_FOSCSEL(FNOSC_FRC & FNOSC_PRIPLL); // Clock Switching and Fail Safe Clock Monitor is disabled
                        // Primary (XT, HS, EC) Oscillator with PLL
_FOSC(FCKSM_CSECMD & OSCIOFNC_OFF & POSCMD_XT);  //Frecuencia de Oscilacion
_FWDT(FWDTEN_OFF);                  //WatchDog time
_FPOR(FPWRT_PWR1 & FPWRT_PWR16); //Reinicio
_FGS(GWRP_OFF);             //Seguridad


int main(void){

    char mytext1[] = " dsPIC33F - LCD ";
    char mytext2[] = "  Hello World   ";

    char *pText1=&mytext1[0];
    char *pText2=&mytext2[0];

    //The settings below set up the oscillator and PLL for 16 MIPS
    PLLFBD = 0x00A0;
    CLKDIV = 0x0048;
    // Initialize LCD Display
    Init_LCD();
    // Hellow World message
    home_clr();
```

```
    puts_lcd(pText1, sizeof(mytext1)-1);
    line_2();
    puts_lcd(pText2, sizeof(mytext2)-1);
    // Infinite Loop
    while(1){};
}
```

.0.6. Taller Práctico

UNIVERSIDAD TÉCNICA DEL NORTE
F ACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
Carrera de Ingeniería en Electrónica y Redes de Comunicación

GUÍA DE PRACTICA-TALLER
Ciclo: Abril - Agosto 2019

Responsable: Rubén Padilla **e-mail:** rubenutn@gmail.com **Telf:**0984539214

1. Introducción

1.1.Nombre de la práctica:

Método de programación para un microcontrolador (dsPIC33F) utilizando un gestor de arranque y una aplicación por interfaz de linea de comandos (IHM).

1.2.Objetivo(s) de la práctica:

1.2.1. General

Dar a conocer un método alternativo de programación de aplicaciones de usuario para microcontroladores mediante el uso de un gestor de arranque y una IHM.

1.2.2. Específicos

- Explicar de manera breve el método convencional y el método propuesto de programación de microcontroladores.
- Verificar los requisitos tanto de hardware como de software previos pera realizar el presente taller.
- Realizar una demostración practica del funcionamiento del gestor de arranque.
- Verificar el correcto funcionamiento del gestor de arranque mediante la puesta en marcha de una aplicación de usuario.

1.3. Marco teórico

1.3.1. Microcontroladores (MCUs)

Es considerado como circuito integrado programable embebido que contiene una memoria para datos, una CPU (Unidad Central de Procesamiento), periféricos de E/S y varios recursos adicionales que ayudan al desarrollo de aplicaciones de usuario, tiene la capacidad de ejecutar instrucciones que están colocadas en parte de la memoria en codificación hexadecimal.

1.3.2. Distribución de Memoria dsPIC33F

Aunque el gestor de arranque requiere un mínimo de memoria, la arquitectura de la MCU restringe un rango de memoria para ser dedicada para esta pequeña aplicación. En la fig 1 se muestra la organización de la memoria del dsPIC33F.

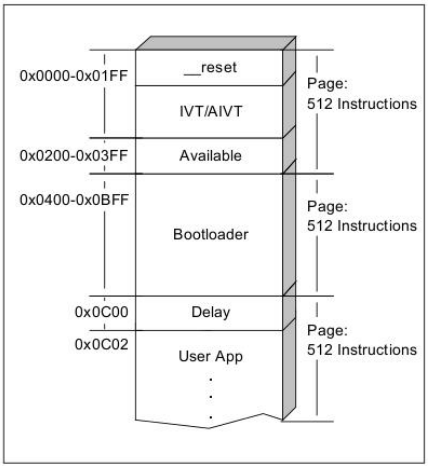


Fig 1: Distribución Memoria dsPIC33F

1.3.3. Gestor de Arranque

Es tomado como una pequeña aplicación que es colocada en una parte de memoria de la MCU la que se ejecuta al instante de inicializada la misma. Se utiliza para ayudar a cargar el archivo HEX y sea colocado en parte de la memoria flash de la MCU, para que esto ocurra debe existir dos partes, la primera el gestor de arranque del lado de la MCU que es programada por única vez con un programador externo y la segunda el uso de una herramienta de software (IHM) del lado del ordenador que tienen la función de codificar el archivo HEX y enviar a través de un puerto estándar de comunicación (USB UART) para que el gestor de arranque de la MCU reciba y coloque el archivo codificado en una sección de su memoria.

1.3.4. Comandos

Comando	Formato	Objetivo	Ejemplo
help	*help*	Muestra menú opciones.	*help*
exit	*exit*	Termina ejecución de la IHM	*exit*
clear	*clear*	Limpia pantalla de la IHM	*clear*
usbscan	*usbscan*	Busca puertos USB asignados.	*usbscan*
reset	*reset [#tarjeta]*	Para y restablece la conexión con el gestor de arranque.	*reset 0*
run	*run [#tarjeta]*	Ejecuta el programa cargado por el usuario.	*run 0*
test	*test [#tarjeta]*	Parpadea LED de tarjeta.	*test 0*
fileaddr	*- fileaddr [#tarjeta][argumento]* *- fileaddr [#tarjeta][argumento][NombreArchivo]*	[r], Muestra directorio del archivo .elf o .hex a ser programado y [w], establece nuevo directorio del archivo .elf o .hex.	*- fileaddr 0r* *- fileaddr 0w/home/ruben/pre nderleds.elf*
erase	*erase [#tarjeta][argumento]*	[a] Borra toda la memoria flash.	*erase 0a*

program	*program [#tarjeta][argumento]*	[a] Programa toda la memoria flash.	*program 0a*
read	*read [#tarjeta][DirMemoria]*	Lee paginas de programas con saltos de 0x400.	*read 00xc00*

Tabla 1: Comandos para IHM

1.4.Materiales y equipos

1.4.1. Materiales Hardware

- 8 Cables para protoboard “Macho-Hembra”, 4 LEDs , 4 Resistencias 330 Ohms, Protoboard, Cable USB tipo B.

1.4.2. Materiales Software

- MPLAB X 2.0.0, Python 2.7.9, MPLAB XC16 C Compiler 1.20.

1.4.2. Equipos

- Laptop con sistema operativo Linux, Placa de desarrollo Full-Flex Evidence

1.5.Procedimiento

1.5.1. Diagrama circuito para práctica

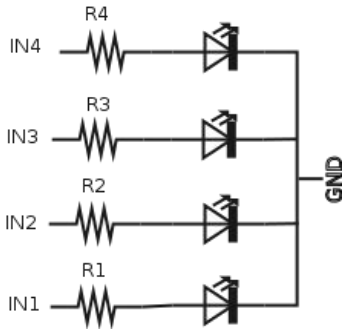


fig 2: Diagrama de circuito para práctica

1.5.1. Bloques de proceso para programación con gestor de arranque de MCU

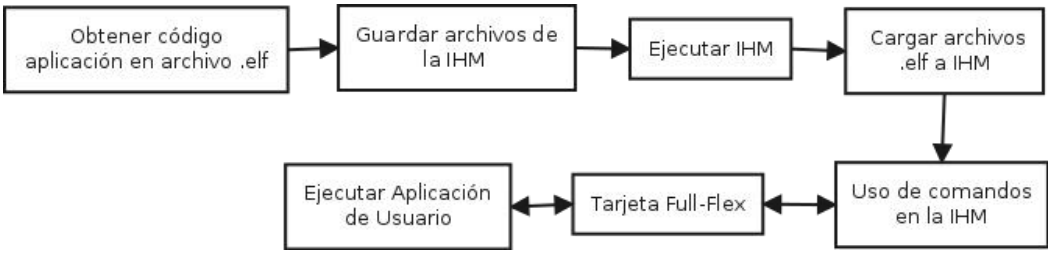


fig 2: Diagrama de bloques proceso programación

1.5.2. Ejecutar IHM

- Abrir terminal como super usuario.
- Dirigirse al directorio del IHM.
- Ejecutar la siguiente sentencia: ***python ./serialboot.py***

1.5.3. Cargar Archivo .elf para IHM

- *help*
- *fileaddr 0r*
- *fileaddr 0w/home/ruben/prenderleds.elf*

1.5.4. Uso de comandos varios

- *read 00xc00*
- *erase 0a*
- *program 0a*
- *run 0*

1.6.Resultados

- Para ejecución de la IHM:

Fig 3: Ejecución IHM

- Para ejecución menú IHM:

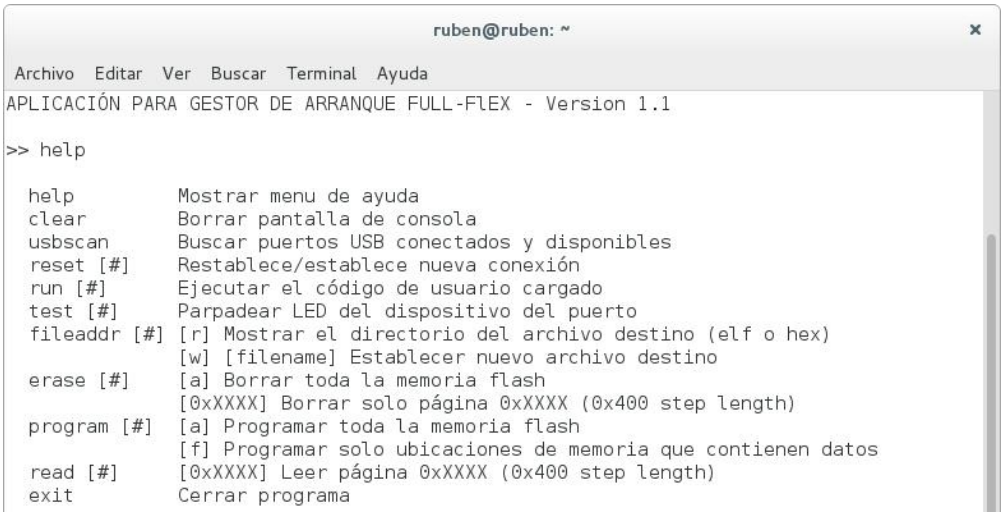


Fig 4: Menú principal IHM

- Para ejecución de comandos básicos IHM:

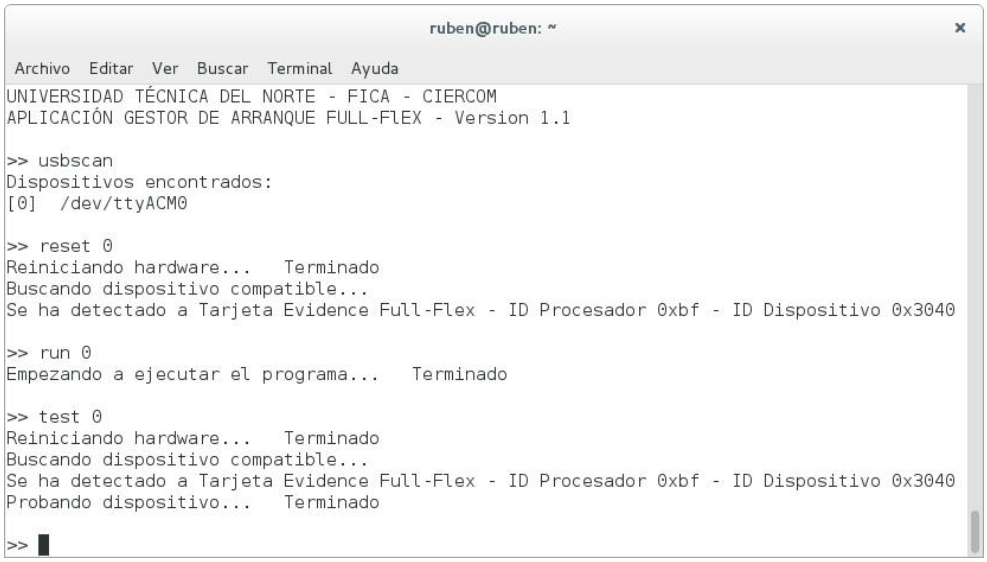


Fig 5: Comandos básicos

- Para ejecución de asignación de directorio del archivo .elf para la IHM:

Fig 6: Asignación directorio .elf a IHM

- Para ejecución de programación comandos básicos IHM:

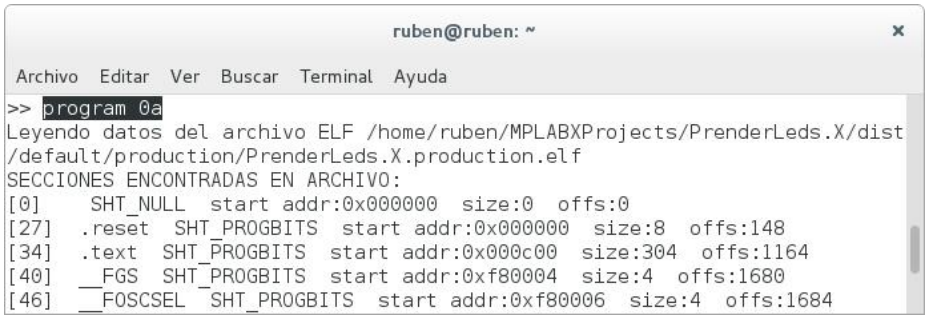


Fig 7: Programacion .elf

- Para ejecución de lectura de páginas de memoria del dsPIC33F

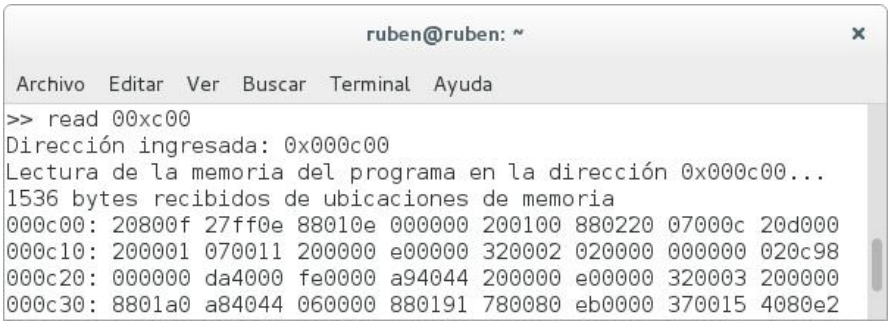


Fig 8: Lectura Memoria dsPIC33F

- Para ejecución de borrado de páginas de memoria del dsPIC33F

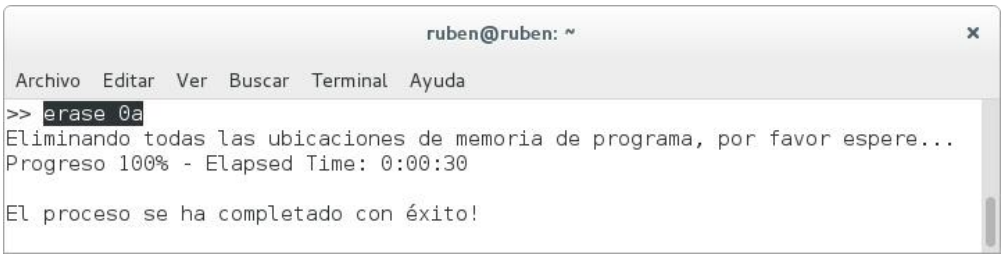


Fig 9: Lectura Memoria dsPIC33F

1.7.Consideraciones

1.7.1. Interfaz Humano Máquina

- Esta aplicación debe ejecutarse bajo el sistema operativo DEBIAN 8 JESSIE para garantizar el funcionamiento ya que en el mismo fueron sometidos las pruebas.
- Debe estar previamente instalado Python para su ejecución bajo el mismo sistema operativo.
- **Opcional:** MPLAB IDE X no necesariamente debe estar bajo el mismo sistema operativo ya que la IHM solo hace uso de los archivos *.elf o .hex* generados por el mismo. También, existen otra configuración previa para MPLAB IDE para el cambio de dirección de memoria para ejecución en primera instancia del gestor de arranque pero no se mencionara ya que no es el objetivo del taller.

1.7.2. Gestor de Arranque

- El uso de un grabador externo es obligatorio para grabar los firmware a las MCUs tanto el gestor de arranque y el puente USB/UART.

1.7.3. Aplicación de Usuario

- La aplicación de usuario (cualquier ejemplo) debe ser orientada para el microcontrolador dsPIC33F y debe estar previamente generado el archivo .elf o .hex ya que no es el objetivo de la practica realizarlo pudiendo ser proporcionada por el responsable de la misma.

.0.7. Encuesta

UNIVERSIDAD TÉCNICA DEL NORTE
F ACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
Carrera de Ingeniería en Electrónica y Redes de Comunicación

ENCUESTA
MÉTODOS DE PROGRAMACIÓN DE MCUs

Encuesta dirigida a las personas que han hecho uso de programadores (grabadores) para MCUs o a su vez tienen conocimiento sobre la misma.

Objetivo: Obtener información de interés directamente de los estudiantes, acerca del uso de programadores de MCUs convencionales en laboratorios y dar a conocer un método alternativo para el mismo proceso.

Indicaciones: Lea detenidamente las preguntas y encierre la opción que corresponda según sea el caso.

1. Usted a utilizado o a escuchado sobre la forma convencional de programar microcontroladores (MCUs).
SI NO

2. Usted a tenido o evidenciado problemas en la programación de MCU.
a) Software b) Hardware c) Daños Físicos (MCU) d) Otros e)Ninguno

3. Cree usted que el factor económico es determinante para los laboratorios de universidades públicas.
SI NO

4. SI la respuesta anterior fue SI cree usted que el estudiante debe asumir inversiones económicas en materiales de laboratorio de microcontroladores.
SI NO

5. Cree que la falta de disponibilidad de grabadores (hardware) en laboratorios es un factor que afecte al desempeño académico de los estudiantes.
SI NO

6. Cuántas horas de laboratorio de microcontroladores tiene o tuvo a la semana.
a)1 a 2 horas b)2 a 4 horas c) Más de 4 horas d) Ninguno

7. Cuántas practicas por lo general pueden existir para un mismo laboratorio.
a)1 a 2 b)2 a 4 c) Más de 4 d) Ninguna

8. Cree usted que debido a la cantidad de elementos que podrían conformar una practica se dificulta el armado siendo este un limitante para completar el objetivo de desarrollo.

SI NO

9. Usted a escuchado sobre gestores de arranque (bootloaders) para microcontroladores.

SI NO

10. Cree usted que este método (gestor de arranque) podría ayudar para mejorar la disponibilidad de grabadores y reducir costos de inversión como alternativa para programado de microcontroladores.

SI NO

PORQUÉ:_____

_____

11. Cree usted que con este método de programación de microcontroladores podría reducir tiempos para armado de practicas en laboratorios.

SI NO

12. Cree usted que este tipo de soluciones (gestores de arranque) ayudaría al desarrollo académico de los estudiantes en laboratorios de microcontroladores.

SI NO

PORQUÉ:_____

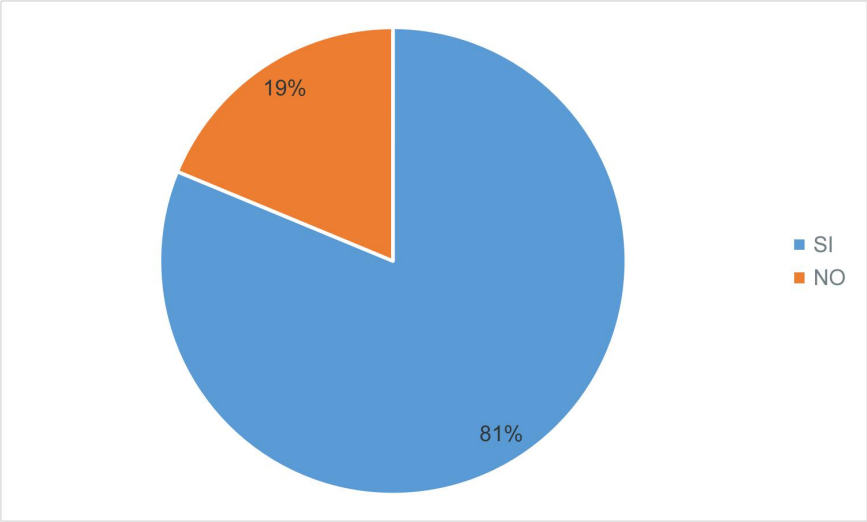
_____

.0.8. Tabulación

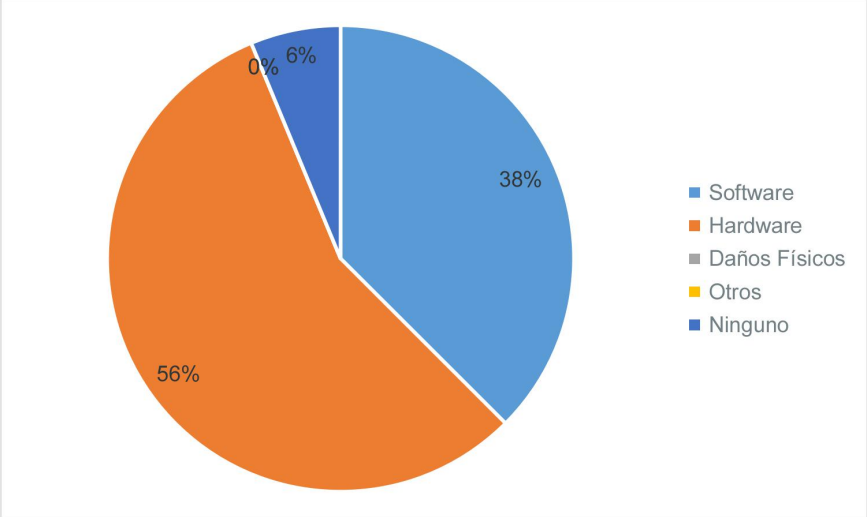
UNIVERSIDAD TÉCNICA DEL NORTE
F ACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
Carrera de Ingeniería en Electrónica y Redes de Comunicación

ENCUESTA
MÉTODOS DE PROGRAMACIÓN DE MCUs

1. Usted a utilizado o a escuchado sobre la forma convencional de programar microcontroladores (MCUs).
SI=13 NO=3

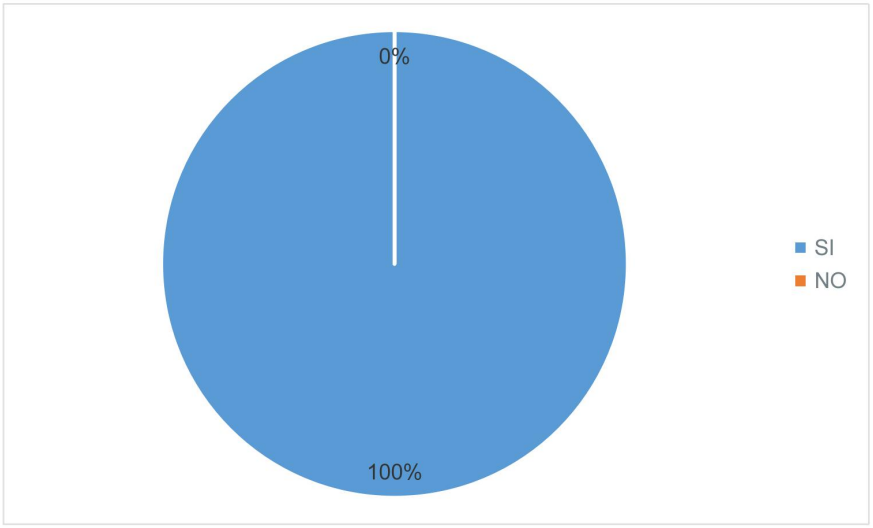


2. Usted a tenido o evidenciado problemas en la programación de MCU.
a) Software=6 b) Hardware=9 c) Daños Físicos (MCU)=0
d) Otros=0 e)Ninguno=1



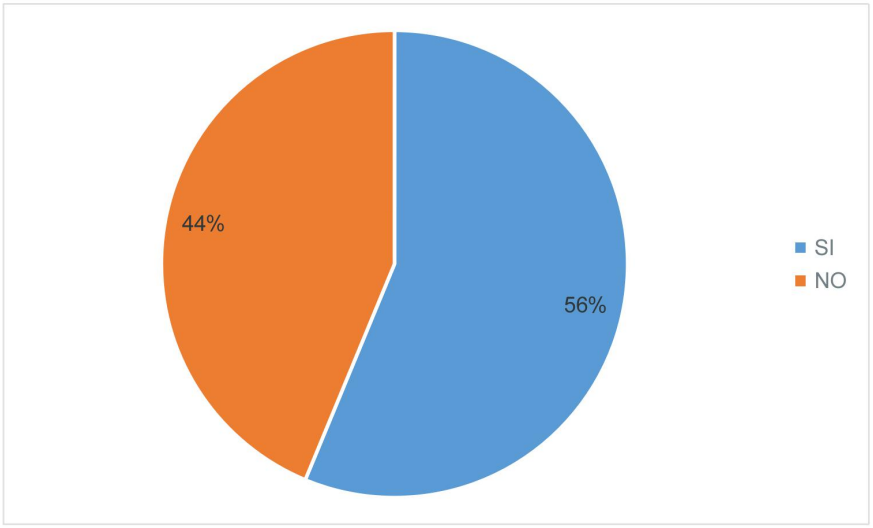
3. Cree usted que el factor económico es determinante para los laboratorios de universidades públicas.

SI=16 NO=0



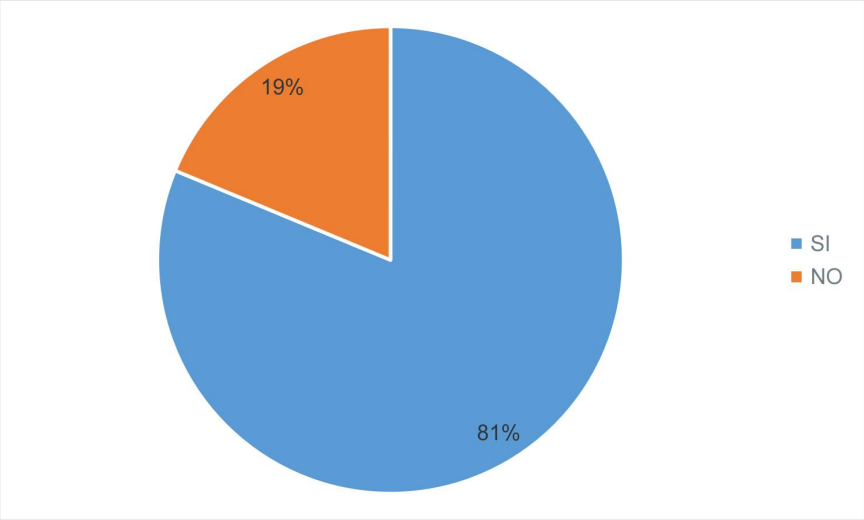
4. SI la respuesta anterior fue SI cree usted que el estudiante debe asumir inversiones económicas en materiales de laboratorio de microcontroladores.

SI=9 NO=7



5. Cree que la falta de disponibilidad de grabadores (hardware) en laboratorios es un factor que afecte al desempeño académico de los estudiantes.

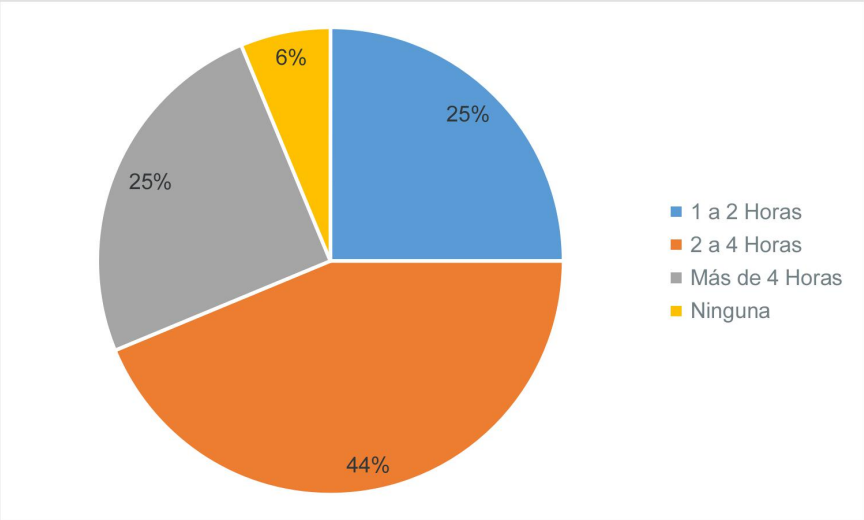
SI=13 NO=3



6. Cuántas horas de laboratorio de microcontroladores tiene o tuvo a la semana.

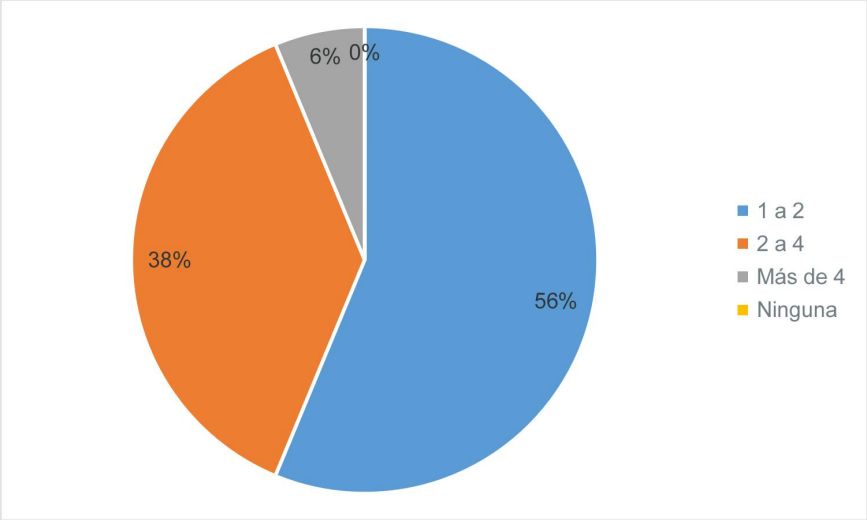
a) 1 a 2 horas=4 b) 2 a 4 horas=7 c) Más de 4 horas=4

d) Ninguno=1



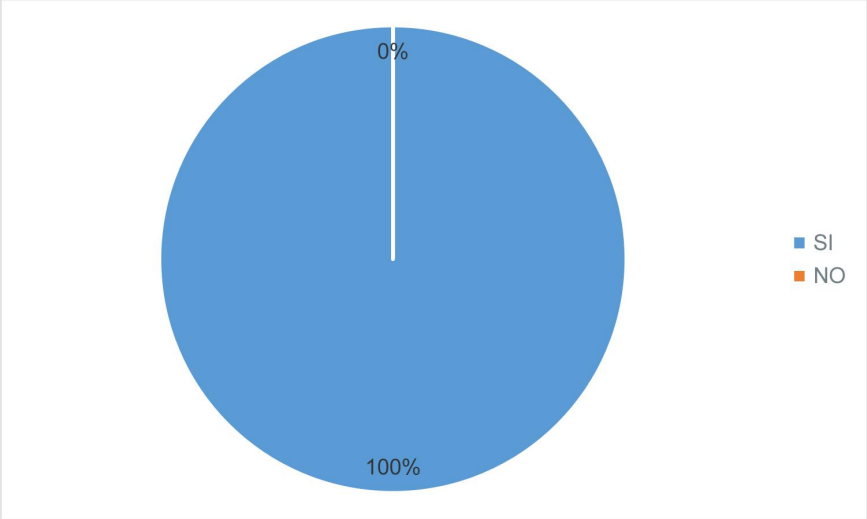
7. Cuántas practicas por lo general pueden existir para un mismo laboratorio.

a) 1 a 2=9 b) 2 a 4=6 c) Más de 4=1 d) Ninguna=0



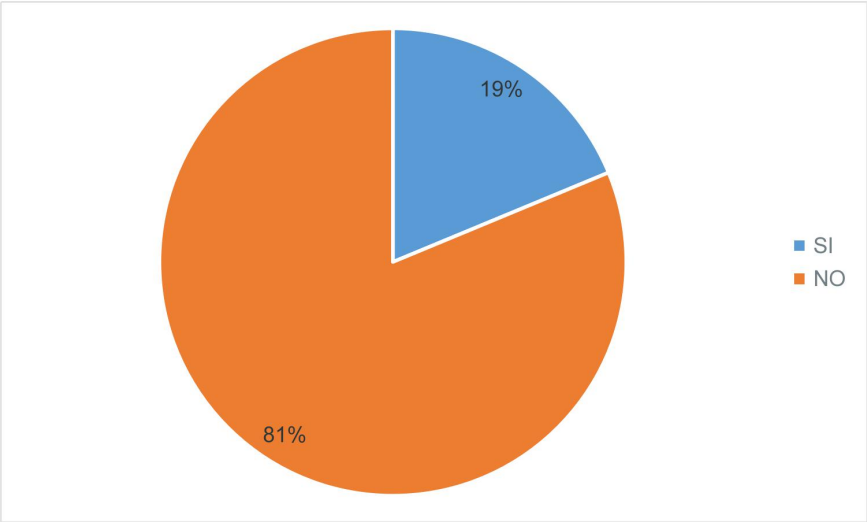
8. Cree usted que debido a la cantidad de elementos que podrían conformar una practica se dificulta el armado siendo este un limitante para completar el objetivo de desarrollo.

SI=16 NO=0



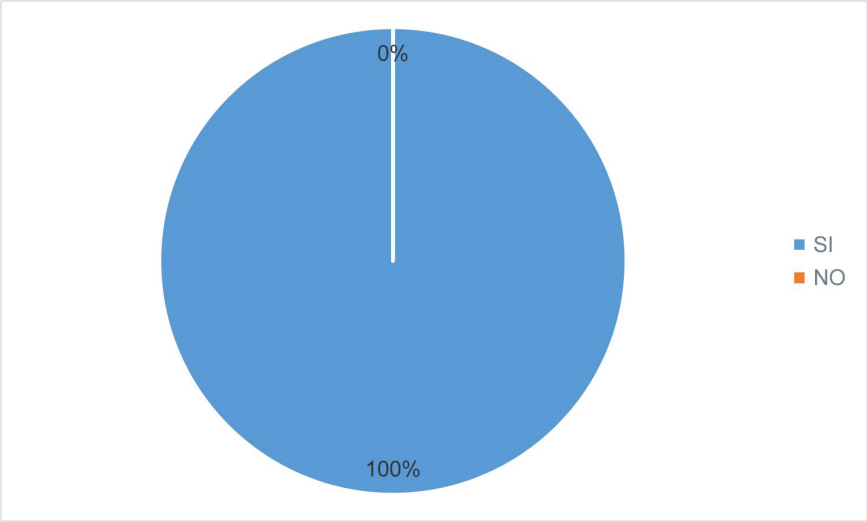
9. Usted a escuchado sobre gestores de arranque (bootloaders) para microcontroladores.

SI=3 NO=13



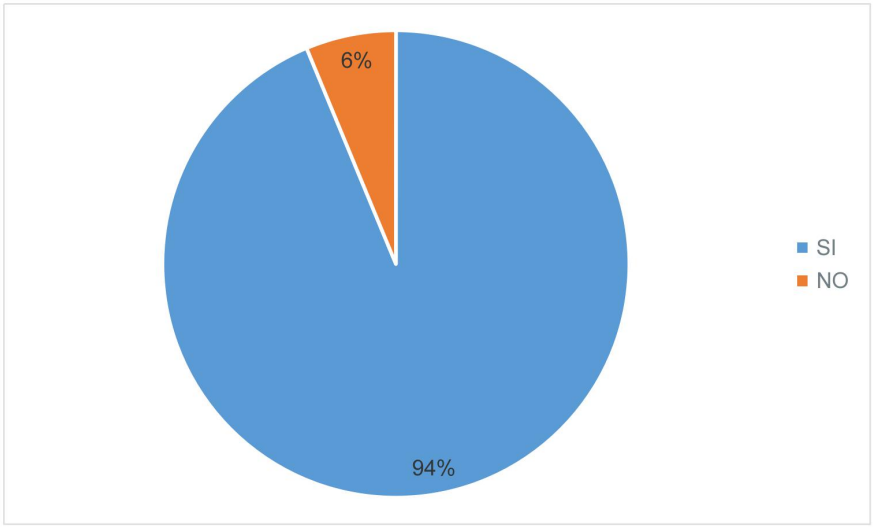
10. Cree usted que este método (gestor de arranque) podría ayudar para mejorar la disponibilidad de grabadores y reducir costos de inversión como alternativa para programado de microcontroladores.

SI=16 NO=0



11. Cree usted que con este método de programación de microcontroladores podría reducir tiempos para armado de practicas en laboratorios.

SI=15 NO=1



12. Cree usted que este tipo de soluciones (gestores de arranque) ayudaría al desarrollo académico de los estudiantes en laboratorios de microcontroladores.

SI=16 NO=0