UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE INGENIERÍA ELECTRÓNICA

“Diseño e Implementación de los módulos didácticos complementarios para entrenamiento de microcontroladores de la familia PIC18F4550”

Trabajo de titulación previo a la obtención del

Título de **Ingeniero Electrónico**

AUTORES:

- **DENNYS LEÓN MARTILLO**
- **CARLOS VÁSQUEZ PARRA**

TUTOR:

ING. ORLANDO BARCIA A. MSC

GUAYAQUIL – ECUADOR

2023

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros **Dennys León Martillo** con cédula de identificación N° **0925946378** y **Carlos Vásquez Parra** con cédula de identificación N° **0915328678;** manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 11 de Marzo del año 2024.

(f)Dennys León Martillo
C.I: 0925946378

(f)Carlos Vásquez Parra
C.I: 0915328678

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros **Dennys León Martillo** con cédula de identificación N° **0925946378** y **Carlos Vásquez Parra** con cédula de identificación N° **0915328678**, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del **Proyecto técnico: "Diseño e implementación de los módulos didácticos complementarios para entrenamiento de microcontroladores de la familia PIC18F4550"**, el cual ha sido desarrollado para optar por el título de: **INGENIERO ELECTRÓNICO**, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 11 de Marzo del año 2024.

(f)Dennys León Martillo
C.I: 0925946378

(f)Carlos Vásquez Parra
C.I: 0915328678

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo Orlando Barcia Ayala con documento de identificación N° 1309445714, docente de la **Universidad Politécnica Salesiana**, declaro que bajo mi autoría fue desarrollado el trabajo de titulación: **"Diseño e Implementación de los módulos didácticos complementarios para entrenamiento de microcontroladores de la familia PIC18F4550"**, realizado por **Dennys León Martillo** con cédula de identificación N° **0925946378** y por **Carlos Vásquez Parra** con cédula de identificación N° **0915328678** obteniendo como resultado final el trabajo de titulación bajo la opción **Proyecto técnico**, que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 11 de Marzo del año 2024.

Atentamente,

Ing. Orlando Barcia Ayala, MSC
C.I: 1309445714

DEDICATORIA

Dedico esta meta a mi familia, a mis padres que siempre me apoyaron y me inculcaron el deseo por estudiar, para concretar mis metas, a mis hijos que han sido la fuente constante de inspiración para culminar este logro y a mi esposa que ha sido el apoyo y la compañía posible en todo este largo y arduo camino que ha llegado a su feliz conclusión.

Dennys León Martillo

DEDICATORIA

Quiero dedicar este logro a Dios, quien ha sido la guía y fortaleza en cada paso de mi vida, también quiero expresar mi más profundo agradecimiento a mis padres, quienes siempre estuvieron apoyándome y alentándome a seguir adelante, el empujón de cada día; sin importar donde estén siempre tengo presente que contare con su apoyo.

Carlos Vásquez Parra

AGRADECIMIENTO

Primero y sobre todo agradezco a Dios, ya que sin su ayuda no hubiera logrado llegar hasta aquí; gracias a mis padres por ser mi punto de partida y origen del deseo de superación constante; agradezco a mi esposa por ser el soporte diario y vital para culminar este proceso académico con éxito, también agradezco a cada uno de los docentes que he tenido quienes aportaron lo necesario y suficiente para crecer académicamente.

Dennys León Martillo

AGRADECIMIENTO

Quiero agradecer a mi madre y mi padre por ser los guerreros de mi vida, quienes sin bajar los brazos dieron hasta el último apoyo para seguir cada meta propuesta en mi vida, me otorgaron la sabiduría para enfrentar los desafíos en cada camino y el empujón necesario para no rendirme.

Carlos Vásquez Parra

RESUMEN

AÑO	ALUMNOS	DIRECTOR DE PROYECTO	TEMA DE PROYECTO DE TITULACIÓN
2023	LEÓN MARTILLO DENNYS VÁSQUEZ PARRA CARLOS	ING. ORLANDO BARCIA AYALA, MSC	"DISEÑO E IMPLEMENTACIÓN DE LOS MÓDULOS DIDÁCTICOS COMPLEMENTARIOS PARA ENTRENAMIENTO DE MICROCONTROLADORES DE LA FAMILIA PIC18F4550"

Este proyecto se enfoca en abordar una brecha crítica en la formación de ingenieros electrónicos al mejorar y ampliar las prácticas complementarias en la programación de microcontroladores PIC 18F4550 en la Universidad. El objetivo principal es brindar a los estudiantes una experiencia educativa más completa y aplicada, centrándose en la programación de periféricos como pantallas LCD, teclados matriciales, pantallas gráficas GLCD y la implementación de sistemas de control PID (Proporcional-Integral-Derivativo) con interfaces gráficas en tiempo real.

El proyecto reconoce la importancia de cerrar la brecha entre la teoría y la aplicación práctica, y se propone lograrlo a través de prácticas enriquecidas que permitan a los estudiantes adquirir habilidades esenciales para enfrentar los desafíos tecnológicos de la actualidad. La programación de microcontroladores es esencial en la ingeniería electrónica moderna, y los microcontroladores PIC 18F4550 son el enfoque central debido a su versatilidad y aplicabilidad.

El proyecto no solo se limita a la programación de microcontroladores, sino que también se adentra en áreas avanzadas como la implementación de sistemas de control PID con interfaces gráficas. Esto permitirá a los estudiantes adquirir habilidades en el diseño y ajuste de sistemas de control en tiempo real, un aspecto crucial en diversas aplicaciones industriales y tecnológicas.

Además, se llevarán a cabo prácticas adicionales que reflejan aplicaciones prácticas del conocimiento adquirido. Esto incluirá el control de cerraduras electrónicas y la regulación de la temperatura mediante sistemas PID, lo que permitirá a los estudiantes aplicar sus habilidades en situaciones del mundo real.

Es importante destacar que las interfaces gráficas para los sistemas de control PID se implementarán utilizando herramientas ampliamente utilizadas en la industria, como MATLAB y LabVIEW. Esto proporcionará a los estudiantes experiencia en el uso de herramientas de ingeniería del mundo real, preparándolos para futuras aplicaciones profesionales.

En resumen, este proyecto tiene como objetivo cerrar la brecha entre la teoría y la práctica en la programación de microcontroladores y sistemas de control. Mediante prácticas enriquecidas y aplicaciones prácticas, los estudiantes adquirirán habilidades esenciales para enfrentar los desafíos tecnológicos actuales y futuros en la ingeniería electrónica. La implementación de interfaces gráficas con software como MATLAB y LabVIEW añade un componente aplicado y profesional, mejorando aún más la preparación de los estudiantes para su futura carrera en la industria.

Palabras Claves: PIC, C, Microprocesador.

ABSTRACT

YEAR	STUDENTS	PRJ. DIRECTOR	SUBJECT
2023	LEÓN MARTILLO DENNYS VÁSQUEZ PARRA CARLOS	ING. ORLANDO BARCIA AYALA, MSC	"DESIGN AND IMPLEMENTATION OF COMPLEMENTARY TEACHING MODULES FOR TRAINING OF MICROCONTROLLERS OF THE PIC18F4550 FAMILY"

This project focuses on addressing a critical gap in the training of electronic engineers by improving and expanding complementary practices in programming PIC 18F4550 microcontrollers at the University. The main objective is to provide students with a more complete and applied educational experience, focusing on the programming of peripherals such as LCD screens, matrix keyboards, GLCD graphic displays and the implementation of PID (Proportional-Integral-Derivative) control systems with graphical interfaces. in real time.

The project recognizes the importance of bridging the gap between theory and practical application and aims to achieve this through enriched practices that allow students to acquire essential skills to meet today's technological challenges. Microcontroller programming is essential in modern electronic engineering, and the PIC 18F4550 microcontrollers are the central focus due to their versatility and applicability.

The project is not only limited to microcontroller programming, but also delves into advanced areas such as the implementation of PID control systems with graphical interfaces. This will allow students to acquire skills in the design and adjustment of control systems in real time, a crucial aspect in various industrial and technological applications.

In addition, additional practices will be carried out that reflect practical applications of the knowledge acquired. This will include control of electronic locks and temperature regulation using PID systems, allowing students to apply their skills in real world situations.

It is important to highlight that the graphical interfaces for the PID control systems will be implemented using tools widely used in the industry, such as MATLAB and LabVIEW. This will provide students with experience using real-world engineering tools, preparing them for future professional applications.

In short, this project aims to bridge the gap between theory and practice in programming

microcontrollers and control systems. Through enriched practice and practical applications, students will acquire essential skills to meet current and future technological challenges in electronic engineering. Implementing graphical interfaces with software such as MATLAB and LabVIEW add an applied and professional component, further improving students' preparation for their future career in industry.

Keywords: PIC, C, Microprocessor.

ABREVIATURAS

GUI: Graphical User Interface.

PCB: Printed Circuit Board.

LCD: Liquid Crystal Display.

GLCD: Graphic Liquid Crystal Display.

PIC: Peripheral Interface Controller.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INTRODUCCIÓN

En el mundo actual, la tecnología se ha convertido en un pilar fundamental en todas las esferas de la sociedad. La educación no es una excepción, y en particular, la formación en ingeniería electrónica y programación de microcontroladores se ha vuelto cada vez más esencial. En este contexto, las universidades desempeñan un papel crucial al proporcionar a los estudiantes las herramientas y conocimientos necesarios para enfrentar los desafíos tecnológicos en constante evolución.

La programación de microcontroladores es una habilidad fundamental en la ingeniería electrónica moderna, permitiendo la creación de sistemas inteligentes y automatizados. Entre los microcontroladores ampliamente utilizados, los dispositivos PIC 18F4550 destacan por su versatilidad y capacidad de procesamiento. Estos dispositivos han encontrado aplicaciones en una variedad de campos, desde sistemas embebidos hasta el control de dispositivos electrónicos y maquinaria.

En este contexto, el presente tema de titulación se enfoca en la ampliación y mejora de las prácticas complementarias ofrecidas en la Universidad, específicamente dirigidas a la programación de microcontroladores PIC 18F4550. Estas prácticas se diseñarán con el propósito de fortalecer la comprensión teórica y práctica de los estudiantes en la programación de microcontroladores, abordando aspectos esenciales como la interfaz con dispositivos de entrada/salida y la implementación de controladores PID (Proporcional-Integral-Derivativo) con una interfaz gráfica amigable.

El contenido de las prácticas se ampliará para incluir la programación de diversos periféricos electrónicos, como pantallas LCD, teclados matriciales y pantallas gráficas GLCD, etc. Estos componentes desempeñan un papel fundamental en la creación de sistemas interactivos y en tiempo real, y su dominio es esencial para los futuros ingenieros electrónicos.

La adición de prácticas de control PID con interfaz gráfica es un elemento destacado de esta propuesta. El control PID es un método ampliamente utilizado en la ingeniería de control para regular y estabilizar sistemas dinámicos. La incorporación de una interfaz gráfica permitirá a los estudiantes visualizar y ajustar parámetros en tiempo real, brindando una experiencia más inmersiva y aplicada en el desarrollo de sistemas de control.

En resumen, este tema de titulación tiene como objetivo principal enriquecer las prácticas educativas en la Universidad, proporcionando a los estudiantes una plataforma de aprendizaje integral para la programación de microcontroladores PIC 18F4550 y la implementación de sistemas de control PID con interfaz gráfica. El resultado esperado es una mejora significativa en la comprensión y capacidad de los estudiantes para abordar desafíos tecnológicos en un mundo cada vez más automatizado y digitalizado.

1. EL PROBLEMA

1.1. Descripción del problema.

A pesar de la importancia creciente de la programación de microcontroladores y la electrónica en la formación de ingenieros, existe una brecha entre los conocimientos teóricos adquiridos en las aulas universitarias y la aplicación práctica en entornos reales. Los estudiantes a menudo enfrentan dificultades al transferir sus conocimientos teóricos a la programación efectiva de microcontroladores, en particular de dispositivos como los PIC 18F4550. A pesar de los esfuerzos actuales en las prácticas complementarias, se ha identificado unas pocas prácticas en áreas cruciales como la programación de pantallas LCD, teclados matriciales y, más significativamente, la implementación de sistemas de control PID con una interfaz gráfica.

Esta brecha entre la teoría y la aplicación práctica limita la capacidad de los estudiantes para abordar desafíos tecnológicos del mundo real, donde la programación de microcontroladores es esencial para el diseño y desarrollo de sistemas electrónicos avanzados. Además, la falta de experiencia en la programación de periféricos y controladores puede resultar en soluciones subóptimas o ineficientes en proyectos que requieren interacción con el entorno físico y sistemas de control precisos.

En este contexto, la problemática que este tema de titulación busca resolver se centra en la falta de prácticas educativas completas y aplicadas en la programación de microcontroladores PIC 18F4550, específicamente en relación con la programación de pantallas LCD, teclados matriciales, pantallas gráficas GLCD y, más importante aún, en la implementación de sistemas de control PID con una interfaz gráfica. Esta carencia de experiencia práctica limita la formación integral de los estudiantes en el campo de la ingeniería electrónica y puede obstaculizar su capacidad para enfrentar exitosamente los desafíos tecnológicos actuales y futuros.

1.2. Antecedentes.

La formación en ingeniería electrónica y programación de microcontroladores ha sido un pilar esencial en la educación técnica durante décadas. Los avances tecnológicos han llevado a la proliferación de microcontroladores en una amplia gama de aplicaciones, desde dispositivos médicos hasta sistemas de control industrial. En este contexto, la enseñanza práctica y aplicada de la programación de microcontroladores se ha convertido en un componente crítico para preparar a los futuros ingenieros para los desafíos del mundo real.

Los microcontroladores PIC (Peripheral Interface Controller) de Microchip Technology han sido ampliamente utilizados en la industria y en el ámbito educativo debido a su versatilidad y potencia de procesamiento. Los dispositivos PIC 18F4550, en particular, han ganado popularidad por su capacidad de interfaz con una variedad de periféricos y su capacidad para implementar sistemas de control en tiempo real.

Los enfoques tradicionales de enseñanza en programación de se centran en la teoría y conceptos fundamentales. Sin embargo, ha habido una creciente conciencia de la necesidad de equilibrar esta formación teórica con aplicaciones prácticas y ejercicios que imiten escenarios del mundo real. A pesar de los esfuerzos en esta dirección, se ha encontrado que muchos programas académicos carecen de prácticas completas que aborden aspectos específicos como la programación de pantallas LCD, teclados matriciales, pantallas gráficas GLCD, etc. y sistemas de control PID con interfaz gráfica.

La implementación de sistemas de control PID con interfaz gráfica es un campo en rápido crecimiento en la ingeniería de control. Los controladores PID son esenciales para mantener la estabilidad y precisión en una variedad de aplicaciones, desde sistemas de climatización hasta robótica industrial. La incorporación de una interfaz gráfica para el ajuste y supervisión en tiempo real añade un nivel de interacción y control que es altamente valioso en la formación de ingenieros en este campo.

En resumen, los antecedentes destacan la importancia de brindar a los estudiantes una educación integral en programación de microcontroladores, con un enfoque específico en la programación de periféricos y sistemas de control avanzados. Los avances tecnológicos y las aplicaciones cada vez más complejas requieren que los programas educativos se adapten para garantizar que los futuros ingenieros estén bien preparados para enfrentar los desafíos del mundo real.

1.3. Importancia y alcances.

El presente tema de titulación sobre la implementación y mejora de prácticas complementarias en la programación de microcontroladores PIC 18F4550, con un enfoque en la programación de periféricos como pantallas LCD, teclados matriciales, pantallas gráficas GLCD, etc. y sistemas de control PID con interfaz gráfica, posee una significativa importancia y abarca varios aspectos esenciales:

Importancia:

Formación Integral de Estudiantes: El proyecto aborda una brecha importante en la formación de ingenieros electrónicos al proporcionar una experiencia educativa más completa y aplicada en la programación de microcontroladores. Esto permitirá a los estudiantes adquirir habilidades prácticas que son directamente relevantes para enfrentar los desafíos tecnológicos actuales y futuros.

Relevancia en la Industria: La programación de microcontroladores y la implementación de sistemas de control son habilidades altamente valoradas en la industria. Los graduados con experiencia en estas áreas estarán mejor preparados para contribuir de manera efectiva en el diseño y desarrollo de sistemas electrónicos avanzados y automatizados.

Innovación Educativa: La incorporación de prácticas centradas en la programación de periféricos y sistemas de control avanzados enriquece las metodologías educativas tradicionales. Esto promueve la innovación en la enseñanza al brindar a los estudiantes la oportunidad de interactuar con tecnologías de vanguardia y desarrollar soluciones prácticas.

Preparación para Investigación: Al ofrecer a los estudiantes la oportunidad de trabajar en proyectos más avanzados y aplicados, el proyecto fomenta la preparación para futuras investigaciones en áreas relacionadas con la electrónica y la ingeniería de control. Los estudiantes estarán mejor equipados para abordar desafíos de investigación y desarrollar soluciones innovadoras.

Alcances:

Programación de Periféricos Electrónicos: Los alcances del proyecto incluyen la implementación de prácticas que aborden la programación de componentes como pantallas LCD, teclados matriciales y pantallas gráficas GLCD. Los estudiantes ganarán experiencia en la interfaz con dispositivos de entrada/salida, lo que es fundamental para el desarrollo de sistemas interactivos.

Implementación de Control PID: El proyecto se centrará en la enseñanza de los conceptos y la implementación práctica de sistemas de control PID (Proporcional-Integral-Derivativo) utilizando microcontroladores. Los estudiantes aprenderán a estabilizar sistemas y mejorar su rendimiento a través del ajuste de parámetros.

Interfaz Gráfica para Control: El tema de titulación ampliará los alcances al incluir la creación de interfaces gráficas para los sistemas de control PID. Esto permitirá a los estudiantes

visualizar y ajustar parámetros en tiempo real, mejorando su comprensión y experiencia en la configuración de controladores.

Aplicación en Proyectos Prácticos: Los alcances se extenderán a la implementación de proyectos prácticos que utilicen las habilidades aprendidas en las prácticas. Esto permitirá a los estudiantes aplicar sus conocimientos en contextos reales y fomentará la creatividad y la resolución de problemas.

En resumen, la importancia y los alcances de este tema de titulación radican en la mejora de la formación de ingenieros electrónicos al abordar deficiencias en la programación de microcontroladores y sistemas de control. Esto preparará a los estudiantes para enfrentar los desafíos tecnológicos actuales y futuros, y contribuirá al avance de la educación y la investigación en ingeniería electrónica y sistemas de control.

1.4. Delimitación del problema.

1.4.1. Temporal.

La implementación de este proyecto se realizó en el periodo en el año 2023 y hasta marzo 2024

1.4.2. Espacial.

El proyecto se realizó para ser usado en las instalaciones de la Universidad Politécnica Salesiana. Sede Guayaquil, campus Barrio Cuba, en Chambers 227 y 5 de junio. En la Figura 1 se puede apreciar la vista superior de la institución.

Figura 1. Vista superior del Campus Guayaquil.

1.4.3. Académica.

Para la implementación del presente proyecto se refuerzan los conocimientos de acuerdo a las siguientes materias de la nueva malla académica: Programación, Programación Orientada a Objetos, Electrónica Digital, Electrónica Analógica, Sistemas Micro procesados, Diseño Electrónico, Teoría de control I y II, Redes Industriales y SCADA.

1.5. Objetivos.

1.5.1 Objetivo General.

Desarrollar módulos didácticos complementarios para realizar prácticas de laboratorio utilizando un microcontrolador de la familia PIC 18F4550.

1.5.2 Objetivo Específicos.

- Comprender el diseño y programación en el lenguaje del microcontrolador PIC 18F4550.
- Diseñar un módulo de microcontroladores para el fortalecimiento sobre el proceso de enseñanza y aprendizaje del elemento PIC 18F4550.
- Verificar las diferentes funciones y partes del módulo del microcontrolador como las herramientas didácticas para la enseñanza de la asignatura.
- Esquematizar un módulo para la codificación de doce prácticas de laboratorio.
- Realizar pruebas prácticas para comprobar que los estudiantes comprendieron sobre la codificación.

2. FUNDAMENTOS TEÓRICOS

En el presente proyecto se utiliza los módulos creados en el trabajo de titulación “Diseño e Implementación de los módulos didácticos de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II” el cual estaba enfocado en un módulo entrenador para aplicaciones con microcontroladores 18F4550 de la familia microchip, integrando diferentes periféricos como teclado matriciales, pantallas liquidas de 16x20 pixeles, botoneras de pulso alto y pulso bajo, salidas mediante leds, un bloque de salidas para las conexiones con el controlador con espadines macho, matrices led con sus respectivos registros de desplazamientos teniendo como medio de conexión cables tipo jumper para las diferentes secciones de la placa permitiendo como lenguaje de programación C mediante el programa PIC C Compiler.

De la misma manera se utiliza como referencia la tesis “Contruccíon de un módulo didáctico para el control de microcontroladores de un GLCD, motor PAP-DC, Sensor T° y teclado Hexadecimal, desde un computador personal utilizando el puerto USB para el laboratorio de microprocesadores y redes de la información”, donde sirve para tomar referencias y apoyo al momento de desarrollar las practicas con temperatura que favorecen al conocimiento teóricos. (Ramírez Alex, 2010).

También se utiliza como referencia la tesis “Diseño y construcción de cinco entrenadores didácticos con sistemas micro procesados y desarrollo de una aplicación de control de velocidad para un motor de corriente alterna”, en el cual se implementaron cinco entrenadores didácticos con sistemas micro procesados, que favorecieron la interiorización eficiente de los conocimientos teóricos impartidos por los docentes. (Franco & Montesdeoca, 2014).

2.1. Hardware de procesamiento.

2.1.1. Microprocesador.

Un microprocesador o computadora digital se compone de tres secciones básicas:

CPU, E/S y memoria, con la adición de algunos circuitos de soporte. Cada sección puede variar en complejidad desde lo básico hasta todas las campanas y silbatos. Veamos cada una por separado:

La entrada/salida (E/S) puede comprender funciones digitales, analógicas y especiales y es la sección que se comunica con el mundo exterior.

La unidad procesadora central (CPU) es el corazón del sistema y puede trabajar en formatos de datos de 4, 8 o 16 bits para realizar los cálculos y la manipulación de datos.

La memoria puede ser RAM, ROM, EPROM, EEPROM o cualquier combinación de éstas y se utiliza para almacenar el programa y los datos.

Se necesita un oscilador para controlar el microprocesador. Su función es introducir datos e instrucciones en la CPU, calcular los resultados y, a continuación, emitir la información. El oscilador puede fabricarse a partir de componentes discretos o ser un módulo prefabricado.

Otros circuitos asociados al microprocesador son los siguientes para evitar que el sistema se bloquee, el búfer para los buses de direcciones y datos para permitir que varios chips se conecten entre sí sin deteriorarse los niveles lógicos y la lógica de decodificación de direcciones y E/S para seleccionar uno de los números de circuitos conectados en el mismo bus.
Es normal referirse a un Microprocesador como un producto que es principalmente la CPU del sistema. La E/S y la memoria estarían formadas por chips separados y requieren un Bus de Datos, un Bus de Direcciones y una Decodificación de Direcciones para permitir un correcto funcionamiento. (An introduction to programing The Microchip PIC in CCS C, 2002)

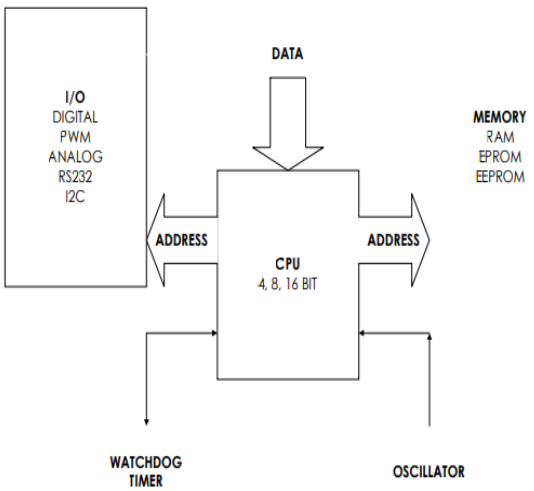


***Figura 2**. Sistema típico de un microprocesador* (An introduction to programing The Microchip PIC in CCS C, 2002)

2.1.2. Microcontrolador.

El PICmicro (MCU), por otro lado, es un Microcontrolador y tiene toda la CPU, memoria, oscilador, watchdog y E/S incorporados dentro del mismo chip. Esto ahorra espacio, tiempo de diseño y problemas de sincronización y periféricos externos, pero en algunas circunstancias puede limitar el diseño a un determinado tamaño de memoria y capacidades de E/S. La familia de microcontroladores PIC ofrece una amplia gama de funciones de E/S, memoria y funciones especiales para satisfacer la mayoría de los requisitos del ingeniero de desarrollo. Encontrará muchos libros generales en las estanterías de las bibliotecas que exploran el diseño de microcontroladores, microprocesadores y ordenadores, por lo que el tema no se ampliará ni duplicará aquí, salvo para explicar las diferencias básicas. (An introduction to programing The Microchip PIC in CCS C, 2002)

Por qué usar el PIC

Eficiencia del código El PIC es un microcontrolador de 8 bits basado en la arquitectura Harvard, lo que significa que hay buses internos separados para la memoria y los datos. Por lo tanto, la tasa de rendimiento aumenta debido al acceso simultáneo a la memoria de datos y de programa. Los microcontroladores convencionales suelen tener un único bus interno para datos y programa. Esto ralentiza el funcionamiento al menos 2 veces en comparación con el PICmicro (MCU).

Seguridad Todas las instrucciones caben en una palabra de memoria de programa de 12 o 14 bits. No existe la posibilidad de que el software salte a la sección DATA de un programa e intente ejecutar los DATOS como instrucciones. Esto puede ocurrir en un Harvard que utilice buses de 8 bits.

Conjunto de Instrucciones Hay 33 instrucciones que tiene que aprender para escribir software para la familia 16C5x y 14 bits para la familia 16Cxx. Cada instrucción, con la excepción de CALL, GOTO o las instrucciones de comprobación de bits (BTFSS, INCFSZ), se ejecuta en un ciclo.

Velocidad El PIC tiene un divisor interno por 4 conectado entre el oscilador y el bus de reloj interno. Esto hace que el tiempo de instrucción sea fácil de calcular, especialmente si usas un cristal de 4 MHz. Cada ciclo de instrucción es de 1 uS. El PIC es un micro muy rápido para trabajar, por ejemplo, un cristal de 20 MHz pasa a través de un programa a 5 millones de instrucciones por segundo. Programa a 5 millones de instrucciones por segundo. - casi el doble que un ¡386SX 33!

Funcionamiento estático El PIC es un microprocesador totalmente estático; en otras palabras, si usted reloj, todos los registros se mantienen. En la práctica el PIC en modo de reposo, lo que detiene el reloj y activa varios relojes y activa varias banderas dentro del PIC para permitirte saber en qué estado estaba antes del estaba antes del reposo. En reposo, el PIC toma sólo su corriente de espera que puede ser inferior a 1uA.

Capacidad de control El PIC tiene una alta capacidad de control de salida y puede controlar directamente Leds, triacs, etc. directamente Leds y triacs, etc. Cualquier pin de E/S puede consumir 25mA o 100mA para todo el dispositivo.

Opciones Una gama de velocidad, temperatura, paquete, líneas de E/S, funciones de temporizador, comunicaciones serie, A/D y tamaños de memoria de la familia PIC para satisfacer prácticamente todas sus necesidades. Versatilidad El PIC es un micro versátil y en volumen es una solución de bajo coste para sustituir incluso unas pocas puertas lógicas, especialmente cuando el espacio es reducido. (An introduction to programing The Microchip PIC in CCS C, 2002)

2.1.3. PIC18F4550.

El Pic18F4550 posee una arquitectura tipo Harvard, ya que dispone de diferentes buses para acceder a la memoria de programa o a la memoria de datos. Esto nos da la opción de acceder a la memoria de datos para ejecutar una instrucción, mientras se lee de la memoria de programa la siguiente instrucción. Es decir, podemos acceder de forma simultánea a ambas memorias. El Bus de memoria de programa: Está formado por 21 líneas de dirección, 16 líneas para instrucciones y 8 líneas para datos. El Bus de memoria de datos: Compuesto por 12 líneas de dirección y 8 líneas de datos.(Diseño e Implementación de un módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II)

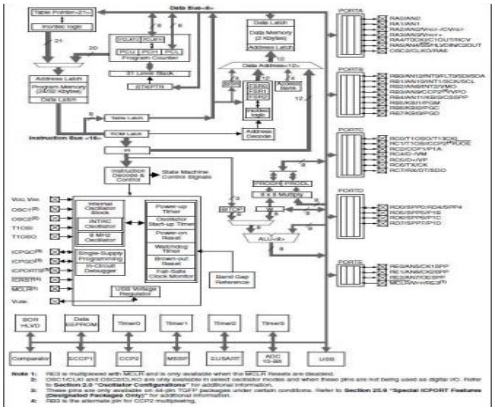


Figura 3. Estructura interna del PIC18F4550 (Diseño e Implementación de un módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II)

2.1.4. Lenguaje de programación C.

C es un lenguaje portátil destinado a tener una modificación mínima al transferir programas de una computadora a otra. Esto está bien cuando se trabaja con PC y mainframes, pero los microcontroladores y los microprocesadores son diferentes. El flujo del programa principal permanecerá básicamente sin cambios, mientras que las distintas configuraciones y el control de puertos/periféricos serán micro específico. Un ejemplo de esto es que los registros de dirección del puerto en un PICmicro (MCU) están configurados 1=Entrada 0=Salida, mientras que H8 es 0=Entrada y 1=Salida. El uso de C en aplicaciones de microcontroladores ha sido provocado por fabricantes que proporcionan áreas de memoria RAM y programas más grandes además a velocidades de operación más rápidas. (An introduction to programing The Microchip PIC in CCS C, 2002)

En este proyecto se usa el lenguaje de alto nivel C para realizar la programación de las prácticas en los microcontroladores PIC18F4550 utilizando el compilador CCS Compiler.

2.2. GLCD.

Una Pantalla Gráfica de Cristal Líquida o GLCD es una pantalla plana formada por una matriz de píxeles monocromos colocados delante de una fuente de luz o reflectora. Es utilizada en proyectos pequeños ya que su consumo de energía es muy pequeño, hay versiones de pantallas con diferentes controladores embebidos, como el Samsung KS0107, Samsung KS0108 o el Toshiba T6963.

Las GLCD disponen de una memoria RAM interna del mismo tamaño de la capacidad que dispone la pantalla, por ejemplo, si una pantalla tiene un tamaño de 128 pixeles de largo por 64 pixeles de alto (128x64) tiene una memoria RAM interna de la misma capacidad (128x64).(Diseño y Construcción de cinco entrenadores didácticos con sistemas microprocesados y desarrollo de una aplicación de control de velocidad para un motor de corriente alterna)

Figura 4. Foto del GLCD en el Módulo.

2.3. Pantalla LCD 20x4.

Una pantalla de cristal líquido o LCD (sigla del inglés liquid-crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de corriente.(Diseño e Implementación de un módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II)

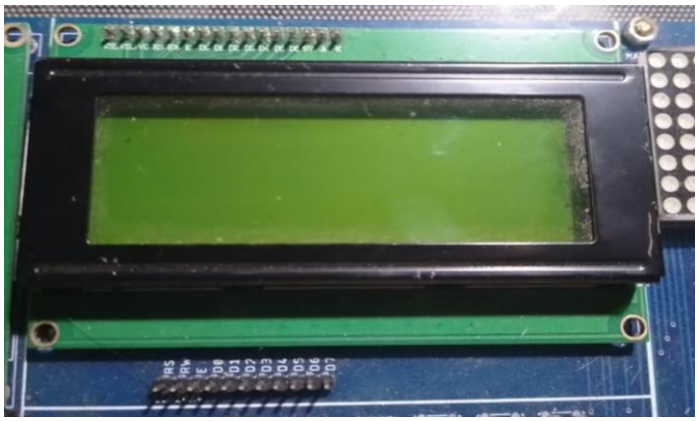


Figura 5. Foto del LCD en el módulo.

2.4. Teclado matricial 4x4.

El teclado matricial de botones plástico formado por 4 filas y 4 columnas para un total de 16 teclas permite agregar una entrada de usuario a tus proyectos. El teclado matricial 4x4 está formado por una matriz de pulsadores dispuestos en filas (L1, L2, L3, L4) y columnas (C1, C2, C3, C4), con la intención de reducir el número de pines necesarios para su conexión. Las 16 teclas necesitan sólo 8 pines del microcontrolador en lugar de los 16 pines que se requerirían para la conexión de 16 teclas independientes. Para poder leer que tecla ha sido pulsada se debe de utilizar una técnica de barrido y no solo leer un pin de microcontrolador. (Diseño e Implementación de un módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II)

Figura 6. Foto del Teclado Matricial 4x4 en el módulo.

2.5. Amplificador de termopar tipo K de salida analógica – AD8495.

Los AD8494/AD8495/AD8496/AD8497 son amplificadores de instrumentación de precisión con compensadores de unión fría de termopar en un circuito integrado. Producen una salida de alto nivel (5 mV/°C) directamente desde una señal de termopar al combinar una referencia de punto de hielo con un amplificador pre calibrado. Se pueden usar como termómetros independientes o como controladores de punto de ajuste de salida conmutada usando un control de punto de ajuste fijo o remoto.

El AD8494/AD8495/AD8496/AD8497 se puede alimentar desde un suministro de un solo extremo (menos de 3 V) y puede medir temperaturas por debajo de 0 °C compensando la entrada de referencia. Para minimizar el auto calentamiento, un AD849x descargado normalmente funciona con una corriente de suministro total de 180 µA, pero también es capaz de entregar más de ±5 mA a una carga.(Precision Thermocouple Amplifierswith Cold Junction Compensation)

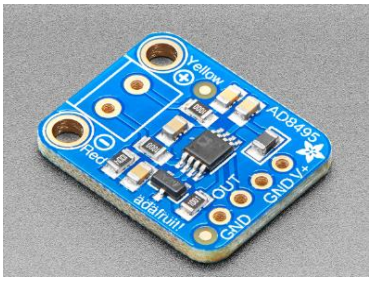


Figura 7. Tarjeta AD8495 (https://www.adafruit.com/product/1778)

2.6. Interfaz USB.

Es un estándar de conectividad simple, topología en estrella, con protocolos que pueden descubrir y configurar automáticamente los dispositivos conectados. Consta de dos pares de cables, uno para transmisión de datos y otro para transmisión de energía. La velocidad de transferencia de datos depende del tipo de USB, el 2.0 se hizo popular en 2000. (S. Carballas, 2018).

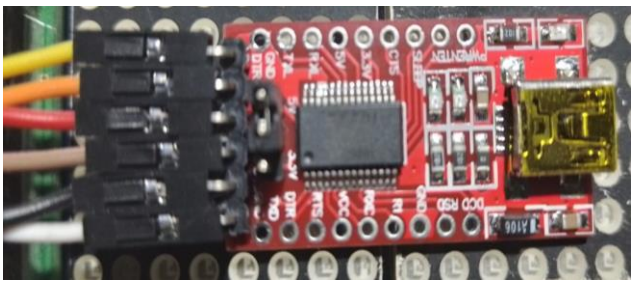


Figura 8. Foto de la interfaz USB en el módulo.

2.7. Convertidor de voltaje DC-DC Step-Down LM2596.

Los reguladores de la serie LM2596 son circuitos integrados monolíticos que brindan toda la funcionalidad activa de los reguladores de conmutación reductores que pueden impulsar cargas 3A con una excelente regulación de línea y carga.

Estos dispositivos están disponibles en voltajes de salida fijos de 3,3 V, 5 V y 12 V y versiones de salida ajustable. Estos controles requieren componentes externos mínimos, son fáciles de usar y tienen compensación de frecuencia interna y un oscilador de frecuencia fija. En la Figura 13 se observa el componente que se empleó (Texas instruments, 1999).

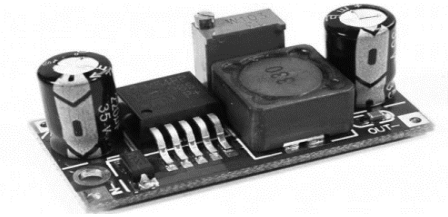


Figura 9. Convertidor Buck Step-Down LM2596 (Naylamp Mechatronics, 2021).

2.8. Controlador PID.

El controlador PID (Proporcional, Integral y Derivativo) es un controlador de retroalimentación diseñado para producir un error de estado estable de cero asintótico entre la señal de referencia y la salida del dispositivo a lo largo del tiempo logrado de manera integrada. Además, el auditor tiene la capacidad de predecir el futuro a través de operaciones derivadas que llevan a predecir el resultado del proceso. El controlador PID es suficiente para resolver problemas de control en muchas aplicaciones industriales, especialmente cuando la dinámica de los procesos lo permite, con requerimientos de desempeño moderados y respuesta rápida a los cambios en la señal de referencia. En la Figura 10 podemos visualizar cómo funciona un sistema de control.(Améstegui Moreno, 2001).

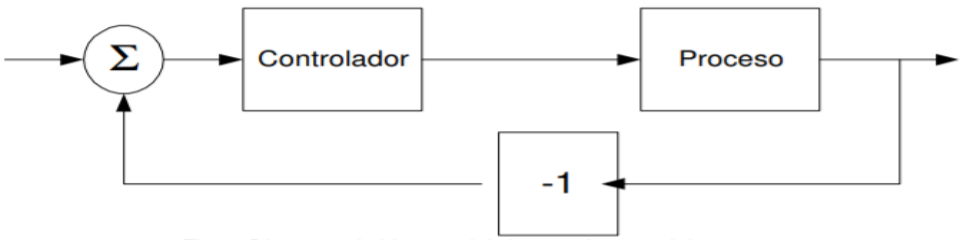


Figura 10. Diagrama de bloques del sistema de control de un proceso (Améstegui Moreno, 2001).

2.8.1. Método de Sintonía Lambda.

El método de sintonía lambda es ampliamente usado en la industria de pulpa y papel, donde hay una conexión entre la uniformidad del papel y la eficiencia manufacturera.

Cuando fue introducido por Dahlin en 1968, el ajuste de lambda ofreció una nueva forma de coordinar la sintonía de los bucles de la fábrica de papel para ganar estabilidad del proceso junto con un producto uniforme.

Para la obtención del diseño PID, considere la función de transferencia PID serie o interactuante:

$$C'(s) = K'_c\left(\frac{(1+sT'_i)(1+sT'_d)}{sT'_i}\right)$$

Dónde:

K'_c= Ganancia Proporcional

T'_i= Tiempo Integral

T'_d= Tiempo Derivativo

Este método consiste en cancelar los polos del proceso con los ceros del controlador. A diferencia de los demás métodos tradicionales, el método de sintonía lambda logra una estabilización sin presentar sobre impulsos, a pesar de que conlleva más tiempo, pero es mucho más eficiente que los otros métodos como López (IAE), Haalman y Ziegler y Nichols. A continuación, en la Figura 27 se muestra un ejemplo de un sistema con diferentes tipos de respuesta con cada método. (Pruna, Edison, & Mullo, 2017).

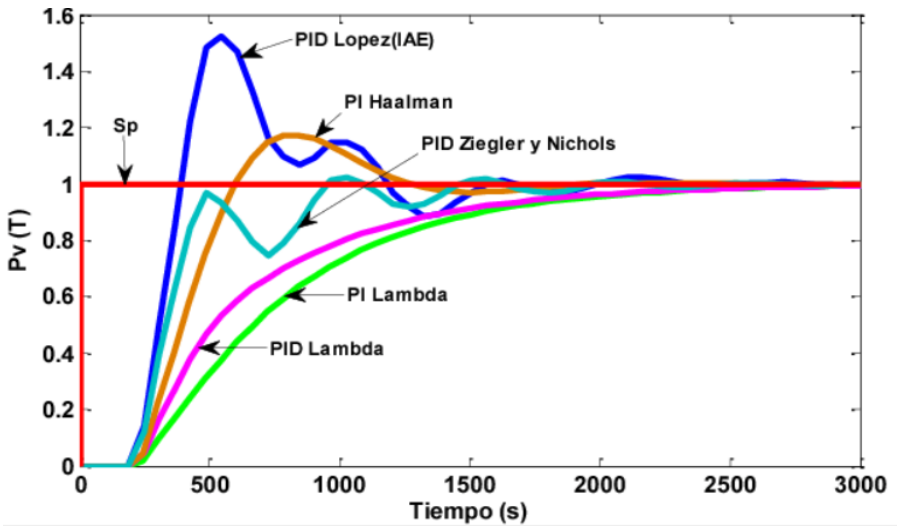


Figura 11. Comparación de respuesta de un sistema con diferentes métodos de sintonía (Pruna, Edison, & Mullo, 2017).

3. MARCO METODOLÓGICO.

3.1. Funcionalidad.

Se diseñaron prácticas complementarias a las ya existentes en el módulo; El producto se entrega con guías de cada práctica y procedimiento con sus respectivos códigos, así mismo el enfoque fue mayoritario al desarrollo de las prácticas con PID y sus respectivas interfaces en Matlab y LabVIEW.

Para poder desarrollar el PID se utilizó un recipiente de acero inoxidable que posee el principio de funcionamiento de una autoclave, además de la termocupla colocada en este.

Figura 12. Esquema del proyecto

Este es el paso a paso de la metodología que se usó para el desarrollo del proyecto:

- Preparación Inicial:

- Familiarización: Comienza por comprender el entorno de desarrollo del compilador CCS y las características del microcontrolador PIC 18F4550.

- Instalación del Software: Asegúrate de que el software del compilador CCS y el software asociado al Pickit3 estén instalados en tu computadora.

- Desarrollo del Código:

- Diseño del Programa: Crea el programa en lenguaje C utilizando el compilador CCS. Asegúrate de tener un claro entendimiento de los objetivos de la práctica y cómo deseas que funcione el microcontrolador.

- Compilación y Generación de Hex:

- Compilación: Utiliza el compilador CCS para convertir el código fuente en un archivo ejecutable. Este archivo suele tener la extensión "hex".

- Conexión del Pickit3 y del PIC 18F4550:

- Conexión Física: Conecta el programador Pickit3 a tu computadora mediante USB y asegúrate de que esté correctamente conectado al microcontrolador PIC 18F4550 en tu placa de desarrollo.

- Programación del PIC:

- Configuración del Pickit3: Abre el software de programación asociado con el Pickit3. Asegúrate de seleccionar el dispositivo PIC 18F4550 y establece las opciones de programación necesarias.

- Carga del Programa: Carga el archivo "hex" generado en el paso de compilación en el programador. Esto suele implicar seleccionar el archivo, configurar opciones adicionales si es necesario y comenzar el proceso de programación.

- Verificación y Depuración:

- Verificación Visual: Una vez que el programa se haya cargado exitosamente en el microcontrolador, verifica que los LED, LCD u otros periféricos involucrados en la práctica respondan según lo esperado.

- Depuración: Si encuentras problemas, realiza una revisión exhaustiva del código para identificar posibles errores y ajusta según sea necesario. Puedes utilizar las herramientas de depuración proporcionadas por el compilador CCS.

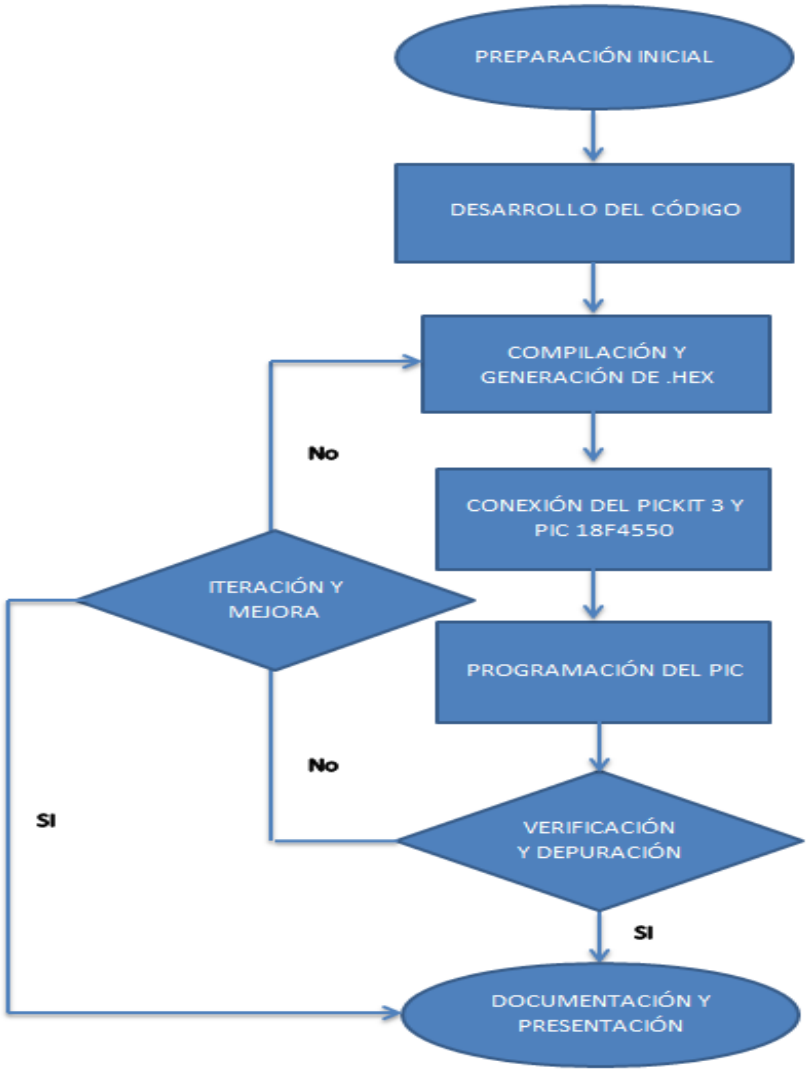
- Iteración y Mejora:

- Pruebas Iterativas: Realiza pruebas adicionales para asegurarte de que el programa funcione correctamente en diferentes escenarios. Realiza ajustes si es necesario para mejorar el rendimiento o la funcionalidad.

- Documentación y Presentación:

- Documentación: Crea documentación detallada sobre la práctica, incluyendo el propósito, el diseño del programa, los pasos de implementación y cualquier desafío encontrado.

- Presentación: Prepara una presentación que describa la práctica, sus objetivos, cómo se implementó y los resultados obtenidos.



Este paso a paso general te proporciona una guía para el desarrollo de las prácticas utilizando el compilador CCS y el programador Pickit3. Sin embargo, es importante tener en cuenta que los detalles específicos pueden variar según el proyecto, la configuración del hardware y el software. Asegúrate de consultar la documentación proporcionada por el compilador CCS y el Pickit3 para obtener instrucciones detalladas sobre su uso.

Cada ítem o Práctica se fundamenta con objetivos, marco teórico, descripción, materiales y software requeridos para su desarrollo paso a paso.

La práctica # 0: Compilación de códigos al PIC18F4550 para el modulo mediante CCS Compiler.

La práctica #1: Inicio de una secuencia de encendido de leds mediante pulsadores utilizando bucles for.

La práctica #2: Manejo de ADC-DAC con potenciómetro.

La práctica #3: Implementación de interrupciones internas y externas usando pulsadores.

La práctica #4: Uso de Timmers para generar retardos de encendido de leds.

La práctica #5: Lectura de temperatura con una termocupla usando comunicación SPI para optimización de pines.

La práctica #6: Conexión de LCD usando comunicación I2C para la optimización de pines.

La práctica #7: Control de acceso (cerradura electrónica) usando LCD y teclado matricial.

La práctica #8: Comunicación serial USP entre dos PIC.

La práctica #9: Control de Velocidad de un motor mediante un potenciómetro y teclado matricial.

La práctica #10: Control PID de una resistencia Calefactora.

La práctica #11: SCADA en LabVIEW para el control de PID de una resistencia calefactora.

La práctica #12: Interfaz gráfica en Matlab para el control PID de una resistencia calefactora.

3.2. Materiales necesarios.

Para llevar a cabo el proyecto se necesitaron de los siguientes materiales. A continuación, se adjuntan los valores de cada uno de los componentes en la Tabla 1.

Elemento	Cantidad	Precio Unitario $	Precio Total $
Termocupla	1	$ 10	$ 10
Módulo SPI para Termocupla	1	$ 4	$ 4
Modulo Analógico para Termocupla	1	$ 12	$ 12
Resistencia Calefactora.	1	$ 20	$ 20
Relé de estado sólido	1	$ 25	$ 25
Pantalla LCD con interfaz I2C	1	$ 8	$ 8
Interfaz USB serial a TTL	1	$ 5	$ 5
MOSFET de potencia IRF540N	1	$ 4	$ 4
		Total $	$ 88

Tabla 1. Lista de Materiales.

Elementos y Software
SOFTWARE PROTEUS
SOFTWARE CCS COMPILER
SOFTWARE LABVIEW
SOFTWARE MATLAB
PROGRAMADOR PICKIT 3
SOFTWARE PICKIT 3
PANTALLA GLCD EN MODULO
PANTALLA LCD EN MODULO
TECLADO MATRICIAL 4X4 EN MODULO

Tabla 2. Lista de elementos y programas.

4. GUIA DE PRÁCTICAS DE LABORATORIO

Práctica #1: Inicio de una secuencia de encendido de leds mediante pulsadores, utilizando bucles for.

<table>
<tr><td colspan="3">FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN</td></tr>
<tr><td colspan="2">CARRERA: Ingeniería Electrónica</td><td>ASIGNATURA: Microcontroladores</td></tr>
<tr><td>NRO. PRÁCTICA:</td><td>1</td><td>TÍTULO PRÁCTICA: Inicio de una secuencia de encendido de leds mediante pulsadores, utilizando bucles for.</td></tr>
<tr><td colspan="3">OBJETIVO:
- OBJETIVO GENERAL.
Implementar un programa para reforzar los conocimientos de bucles.
- OBJETIVOS ESPECÍFICOS:
- Implementar un programa con un bucle for.
- Implementar en el mismo programa el encendido de salidas digitales.</td></tr>
<tr><td colspan="2">INSTRUCCIONES:</td><td>1. Analizar la solución propuesta de la práctica#1 ubicada en los anexos de la memoria técnica.
2. Conectar el módulo a una fuente de alimentación de 5-12VDC
3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 1.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1 Definir las variables y realizar el direccionamiento correcto.</td></tr>
<tr><td colspan="3">2 Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.</td></tr>
<tr><td colspan="3">RESULTADO(S) OBTENIDO(S):
Se visualiza la secuencia de encendido de los leds conectados al puerto B.</td></tr>
<tr><td colspan="3">CONCLUSIONES:
Con esta práctica se refuerza los conocimientos de bucles y activación de entradas y salidas digitales.</td></tr>
<tr><td colspan="3">RECOMENDACIONES:
Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.</td></tr>
</table>

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Práctica #2: Manejo de ADC-DAC con potenciómetros.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	2	**TÍTULO PRÁCTICA**: Manejo de ADC-DAC con potenciómetro.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para reforzar los conocimientos de Convertidores Analógico-Digital Digital-Analógico.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa con ADC.
- Implementar un programa con DAC.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#2 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 2.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza la conversión de una señal analógica a digital y de digital a analógico PWM.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de conversores ADC y DAC.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #3: Implementación de interrupciones internas y externas usando pulsadores.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	3	**TÍTULO PRÁCTICA**: Implementación de interrupciones internas y externas usando pulsadores.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para reforzar los conocimientos acerca de las interrupciones.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa con una interrupción externa por pulsador.
- Implementar un programa con interrupción interna por comunicación serial.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#3 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 3.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza el inicio de la secuencia de LEDS de la práctica#1 pero esta vez activadas por la interrupción externa de RB0.

Así mismo se visualizan las 3 diferentes acciones cuando se envían caracteres por el puerto serial activando la interrupción interna.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de interrupciones internas y externas.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #4: Uso de Timmers para generar retardos de encendido de leds.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	4	**TÍTULO PRÁCTICA**: Uso de Timmers para generar retardos de encendido de leds.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para reforzar los conocimientos acerca de los Timmers.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa con el Timmer 1 del PIC que genera un retardo de encendido de 8 segundos.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#4 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 4.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza el encendido y apagado de un led con una frecuencia de 8 segundos.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de los Timmers en el PIC.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #5: Lectura de temperatura con una termocupla usando comunicación SPI para la optimización de pines.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN**	
CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	5	**TÍTULO PRÁCTICA**: Lectura de temperatura con una termocupla usando comunicación SPI para la optimización de pines.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa que realice la lectura de temperatura.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa que utilice un módulo SPI para termocuplas.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#5 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 5.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza la temperatura utilizando una termocupla como sonda que a su vez es conectado al PIC por medio de un módulo SPI.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de comunicación SPI.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #6: Conexión del LCD usando I2C para la optimización de pines.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN**	
CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	6	**TÍTULO PRÁCTICA**: Conexión del LCD usando comunicación I2C para la optimización de pines.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para usar comunicación I2C.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa para conectar una pantalla LCD por medio de comunicación I2C.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#6 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 6.

ACTIVIDADES POR DESARROLLAR

1 Definir las variables y realizar el direccionamiento correcto.

2 Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza en la pantalla LCD caracteres que son enviados del PIC a la pantalla por medio de comunicación I2C.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de comunicación I2C.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #7: Control de acceso (cerradura electrónica) usando LCD y teclado matricial.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	7	**TÍTULO PRÁCTICA**: Control de acceso (cerradura electrónica) usando LCD y teclado matricial.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa de control de acceso.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa que utilice la pantalla LCD y el teclado matricial para realizar una cerradura electrónica.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#7 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 7.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza que cuando se digita la clave correcta que esta guardada en la ROM del PIC se realiza el encendido de un LED que indica la apertura de la cerradura y cuando se coloca una clave incorrecta no se acciona.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de guardar y usar datos almacenados en la memoria, así como el uso del LCD en conjunto con el teclado matricial.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #8: Comunicación serial USB entre dos PICs.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN**	
CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA: 8	**TÍTULO PRÁCTICA**: Comunicación serial USB entre dos PIC.	

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para poder realizar una comunicación serial entre dos PIC.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa en el PIC que envía los datos.
- Implementar un programa en el PIC que recibe los datos.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#8 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 8.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza el envío y recepción de caracteres entre dos PIC.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de comunicación serial.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #9: Control de velocidad de un motor mediante un potenciómetro y teclado matricial.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN**	
CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	9	**TÍTULO PRÁCTICA**: Control de velocidad de un motor mediante un potenciómetro y teclado matricial.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa para poder controlar la velocidad de un motor.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa en el cual, mediante el teclado matricial, un potenciómetro y PWM se controla la velocidad de un motor.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#9 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 9.

ACTIVIDADES POR DESARROLLAR

1 Definir las variables y realizar el direccionamiento correcto.

2 Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se regula la velocidad de un motor DC ya sea con un potenciómetro o digitando en el teclado matricial.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de PWM.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #10: Control PID de una resistencia calefactora.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	10	**TÍTULO PRÁCTICA**: Control PID de una resistencia calefactora.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar un programa que realice un control PID

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa en el PIC que realice el control PID de un sistema que consiste en una resistencia calefactora y una termocupla como sensor, los datos se visualizan en el LCD.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#10 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 10.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza el control de la temperatura en el sistema que tiene el principio de funcionamiento de una autoclave.

CONCLUSIONES:

Con esta práctica se aprende a implementar un control PID en un microcontrolador.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

Práctica #11: SCADA en Labview para el control PID de una resistencia calefactora.

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN

CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	11	**TÍTULO PRÁCTICA**: SCADA en LabVIEW para el control PID de una resistencia calefactora.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar una Interfaz en LabVIEW para la practica 11.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa en LabVIEW en el cual se lean los datos de temperatura que llegan del PIC por medio de comunicación serial.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#11 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 11.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza la recepción de datos de temperatura provenientes del PIC por medio de comunicación serial.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de interfaces gráficas con LabVIEW.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Práctica #12: Interfaz gráfica en Matlab para el control PID de una resistencia calefactora.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN**	
CARRERA: Ingeniería Electrónica		**ASIGNATURA**: Microcontroladores
NRO. PRÁCTICA:	12	**TÍTULO PRÁCTICA**: Interfaz gráfica en Matlab para el control PID de una resistencia calefactora.

OBJETIVO:

- OBJETIVO GENERAL.

Implementar una Interfaz en Matlab para la practica 11.

- OBJETIVOS ESPECÍFICOS:

- Implementar un programa en Matlab en el cual se lean los datos de temperatura que llegan del PIC por medio de comunicación serial.

INSTRUCCIONES:	1. Analizar la solución propuesta de la práctica#12 ubicada en los anexos de la memoria técnica.
	2. Conectar el módulo a una fuente de alimentación de 5-12VDC
	3. Realizar las conexiones correctas del módulo de acuerdo con el esquemático planteado en la resolución del anexo de la PRÁCTICA 12.

ACTIVIDADES POR DESARROLLAR

1	Definir las variables y realizar el direccionamiento correcto.
2	Elaborar la programación y el cableado comprobando el funcionamiento de la práctica.

RESULTADO(S) OBTENIDO(S):

Se visualiza la recepción de datos de temperatura provenientes del PIC por medio de comunicación serial.

CONCLUSIONES:

Con esta práctica se refuerza los conocimientos de interfaces gráficas con Matlab.

RECOMENDACIONES:

Revisar el correcto conexionado de todos los elementos que intervienen en la práctica, para evitar averías.

***Docente*: __MSC. Orlando Barcia Ayala____**

Firma: _____

Resolución CS N° 076-04-2016-04-20

5. RESULTADOS

5.1. Diseño y elaboración de las prácticas.

Para la realización de estas prácticas se utilizaron diversos software, como Proteus, CCS, Hercules, Matlab, LabVIEW. Los resultados de las prácticas se detallan a continuación:

- En la práctica uno se pudo comprobar la utilidad de los bucles for para realizar secuencias, en este caso se realizó la secuencia de encendido de luces Leds, esta secuencia es iniciada por un pulsador.

- En la práctica dos se verificó que el uso de un conversor de analógico al digital tiene mucha utilidad ya que en esta práctica realizamos la lectura del voltaje que ingresa en el puerto RA0.

- En la práctica tres se usó la interrupción externa de RB0 para generar el mismo resultado de la práctica uno, así mismo se realizó también una interrupción interna por comunicación serial para poder leer el voltaje de RA0 o encender un Led.

- En la práctica cuatro se comprobó que con el uso de los Timmers del PIC se pueden generar señales de reloj de diferentes frecuencias dependiendo de cómo se configure y cual Timmer se utilice, en este caso específico se utilizó el Timmer 1 con un periodo de 8 segundos.

- En la práctica cinco se verificó el uso de la comunicación SPI como una alternativa para la optimización de pines se comprobó velocidad de comunicación y utilidad.

- Así mismo en la práctica seis se verificó el uso de la comunicación I2C como una alternativa para la optimización de pines se comprobó velocidad de comunicación y utilidad esta vez con una pantalla LCD.

- En la práctica siete se constató la utilidad de los periféricos en usos reales en este caso para el uso de una cerradura electrónica, se utilizó la memoria del PIC para guardar una clave y leer esta para comparar si la combinación numérica ingresada mediante el teclado corresponde.

- En la práctica ocho se verificó como mediante comunicación serial dos PIC se pueden enviar y recibir caracteres, así mismo se constató que se pueden realizar dos comunicaciones seriales asignando diferentes pines.

- En la práctica nueve se comprobó cómo se puede utilizar el PWM entre otras cosas para el control de velocidad de motores DC, así mismo se desarrollaron destrezas para configurar este PWM.

- En la práctica diez se volvió a usar el PWM, pero esta vez con un controlador PID comprobando la versatilidad del uso del PWM así mismo comprobando la utilidad del PIC 18F4550 para realizar el control y monitoreo de diferentes procesos.

- En la práctica once se comprobó cómo se pueden crear diferentes tipos de interfaces para comunicación con el PIC18F4550 en este caso con el software LabVIEW.

- En la práctica doce se comprobó el uso de interfaces, pero esta vez con el software de Matlab.

6. CRONOGRAMA

ACTIVIDAD	MES											
	1	2	3	4	5	6	7	8	9	10	11	12
DISEÑO DE MODULOS												
DISEÑO DE PCB												
ENSAMBLAJE DE PCB												
TEST DE PCB												
DESARROLLO DE PRASCTICAS												
PRUEBA DE PRACTICAS												
ENTREGA DE TESIS												

7. PRESUPUESTO

DESCRIPCION	CANTIDAD	VALOR UNITARIO $	VALOR TOTAL $
DISEÑO DE LA TARJETA	8	19	152
MOTOR DC DE 12 VOLTIOS	8	20	160
PANTALLA NEXTION	8	47	376
TRANSISTORES IRF XXX	16	2	32
RESISTENCIA CALEFACTORA DE 12 VOLTIOS	8	30	240
MODULO I2C PARA PANTALLA LCD	8	3,50	28
TERMOCUPLA TIPO K + MODULO SENSOR MAX6675	8	9,40	75,20
ELEMENTOS VARIOS	1	30	30
		TOTAL $	1093,20

8. CONCLUSÍONES

El proyecto de mejora y ampliación de prácticas complementarias en la programación de microcontroladores PIC 18F4550 con enfoque en periféricos y sistemas de control PID con interfaces gráficas representa un avance significativo en la formación de ingenieros electrónicos. A partir de los antecedentes identificados y la problemática reconocida, se han alcanzado varias conclusiones importantes:

- La brecha entre la teoría y la aplicación práctica en la programación de microcontroladores ha sido reconocida como una limitación en la educación. La implementación de prácticas enriquecidas y aplicadas, que incluyen la programación de periféricos y la implementación de sistemas de control PID, es crucial para una formación integral y efectiva de los estudiantes.

- Los conocimientos adquiridos a través de estas prácticas enriquecidas prepararán a los estudiantes para enfrentar desafíos en la industria de la electrónica y la automatización. La capacidad de programar microcontroladores, interactuar con periféricos y diseñar sistemas de control PID será una habilidad altamente valiosa en el mercado laboral.

- La incorporación de aplicaciones prácticas, como el control de cerraduras electrónicas y la regulación de la temperatura mediante sistemas PID, demuestra la relevancia directa de los conocimientos adquiridos en situaciones del mundo real. Los estudiantes podrán aplicar sus habilidades en proyectos concretos y resolver problemas reales.

- La implementación de interfaces gráficas utilizando herramientas como MATLAB y LabVIEW amplía la experiencia de los estudiantes en entornos tecnológicos avanzados. Esto fomenta la innovación al permitir a los estudiantes explorar soluciones creativas y aplicar tecnologías líderes en la industria.

- La implementación de prácticas enriquecidas fomenta un enfoque de aprendizaje activo, donde los estudiantes pueden experimentar directamente los conceptos teóricos en un entorno controlado. Esto mejora la retención de conocimientos y la comprensión de los conceptos fundamentales.

- En conjunto, este proyecto se presenta como un paso valioso y necesario para mejorar la formación en ingeniería electrónica. Al cerrar la brecha entre la teoría y la práctica, y al proporcionar experiencias aplicadas en la programación de microcontroladores, la implementación de sistemas de control PID y la interacción con periféricos, se prepara a los estudiantes para ser profesionales altamente competentes y listos para enfrentar los desafíos tecnológicos del futuro.

9. RECOMENDACIONES.

Basándose en la realización y conclusión del proyecto de mejora y ampliación de prácticas complementarias en la programación de microcontroladores PIC 18F4550 con enfoque en periféricos y sistemas de control PID con interfaces gráficas, se proponen las siguientes recomendaciones para maximizar el impacto y el éxito continuo de este esfuerzo:

- Es fundamental realizar una evaluación continua y sistemática de las prácticas implementadas. Recopilar retroalimentación de los estudiantes, identificar áreas de mejora y realizar ajustes según sea necesario asegurará que las prácticas sigan siendo efectivas y estén alineadas con los objetivos educativos.

- Para lograr una implementación sostenible, se recomienda la integración de las prácticas en el currículo académico de manera formal. Esto garantizará que todos los estudiantes tengan la oportunidad de adquirir estas habilidades esenciales y se beneficien de la formación mejorada.

- Proporcionar capacitación y recursos adecuados para los docentes encargados de impartir estas prácticas es esencial. Asegurarse de que estén bien preparados para guiar a los estudiantes a través de las prácticas, responder preguntas y abordar desafíos técnicos garantizará una experiencia educativa de alta calidad.

- Dado que la tecnología avanza constantemente, se recomienda mantenerse actualizado con los últimos desarrollos en microcontroladores, sistemas de control y herramientas de interfaz gráfica. Esto permitirá que las prácticas reflejen las tendencias tecnológicas actuales y futuras.

- Al implementar proyectos adicionales, como el control de cerraduras y sistemas de control PID, se alienta a fomentar la creatividad y la resolución de problemas entre los estudiantes. Proporcionar oportunidades para aplicar los conocimientos en contextos variados promoverá el pensamiento crítico y la innovación.

- Dada la naturaleza interdisciplinaria de la electrónica y la ingeniería de control, se recomienda explorar oportunidades de colaboración con otros departamentos académicos. Esto podría enriquecer aún más las prácticas y permitir a los estudiantes aplicar sus habilidades en diversas áreas.

- Los resultados y las lecciones aprendidas de este proyecto podrían ser compartidos a través de seminarios, conferencias o publicaciones académicas. Compartir las mejores prácticas y los enfoques exitosos podría inspirar a otras instituciones educativas a implementar enfoques similares.

- En última instancia, el éxito continuo de este proyecto dependerá de un compromiso constante con la mejora y la adaptación para satisfacer las necesidades cambiantes de los estudiantes y la industria. Al implementar estas recomendaciones, se puede asegurar que la formación en programación de microcontroladores y sistemas de control sea efectiva, relevante y preparatoria para el futuro.

10. REFERENCIAS BIBLIOGRÁFICAS.

Nigel Gardner. (2002). An introduction to programing The Microchip PIC in CCS C.*USA*.

José Eduardo López Guzmán, Lissette Carolina Reinoso Figueroa. (2021). “Diseño e implementación de un módulo de entrenamiento para microcontrolador 18F4550 con software didáctico para la materia de microcontroladores I y II”*, Universidad Politécnica Salesiana*.

Ramírez Guaña Alex Rodrigo. (2010). "Contrucción de un módulo didáctico para el control con microcontroladores de un GLCD, motor PAP-DC, sensor T° y teclado Hexadecimal, desde un computador personal utilizando el puerto USB para el laboratorio de microprocesadores y redes de la información", Universidad Politécnica Nacional.

Rafael Christian Franco Reina, Israel Montesdeoca Pladines. (2014). “Diseño y Construcción de cinco entrenadores didácticos con sistemas microprocesados y desarrollo de una aplicación de control de velocidad para un motor de corriente alterna”*, Universidad Politécnica Salesiana*.

Améstegui Moreno, M. (2001). *Control PID.* La Paz: Universidad Mayor de San Andres.*

Naylamp Mechatronics. (2021). Obtenido de https://naylampmechatronics.com/conversores-dc-dc/196-convertidor-voltaje-dc-dc-step-down-3a-lm2596.html#:~:text=El%20convertidor%20DC%2DDC%20LM2596,dise%C3%B1o%20de%20fuentes%20de%20alimentaci%C3%B3n.

Pruna, E., Edison, S., & Mullo , S. (2017). *PI and PID Controller Tuning Tool Based on the Lambda Method.* Sangolqui: IEEE.

S. Carballas, J. C. (2018). *Creación de un protocolo de comunicación entre un PC y un puerto USB de un microcontrolador y su integración en MATLAB.* Coruña, España. Obtenido de https://ruc.udc.es/dspace/bitstream/handle/2183/21183/CarballasChas_Sergio_TFG_2018.pdf?sequence=2&isAllowed=y

Texas instruments. (1999). *LM2596 SIMPLE SWITCHER Power Converter 150-kHz.* Texas.

ANEXO B. MANUAL DE PRÁCTICAS

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 0**
NOMBRE PRÁCTICA:	Compilación de códigos al PIC18F4550 para el módulo mediante CCS Compiler.	

2. Objetivo General.

Implementar una guía para la programación del código en el PIC18F4550.

3. Objetivos Específicos.

Implementar una guía para el estudiante donde pueda tener el conocimiento para la creación y grabación del archivo .hex al PIC18F4550.

4. Marco Teórico.

PIC C COMPILER CCS.:

Es un Software diseñado para la programación de microcontroladores PIC. El software es compatible con una amplia variedad de microcontroladores PIC, entre el que se usara PIC18F4550.

PICkit 3:

El PICkit 3 permite la depuración y programación del código generado en el CCS Compiler en los microcontroladores PIC y es utilizado en el entorno de desarrollo integrado, el módulo PICkit 3 se conecta a la PC mediante su interfaz USB full-speed.

5. Descripción.

La práctica comienza con el inicio del software CCS Compiler para realizar el código utilizado para las practicas citadas de la 1 a la 12, una vez realizado el código; se realiza la verificación previo a la programación que se realizara en el módulo PICkit3 para hacer uso del PIC18F4550.

6. Materiales.

1. Módulo didáctico.
2. Jumpers Conectores.
3. Software CCS Compiler
4. Software PICKIT 3
5. Programador Pickit3.

7. Desarrollo.

Paso N. º 1: Para cargar las prácticas al módulo, es necesario generar un archivo .hex desde el ide CCS Compiler para lo cual primero iniciamos el software dando doble clic izquierdo sobre el icono del escritorio.

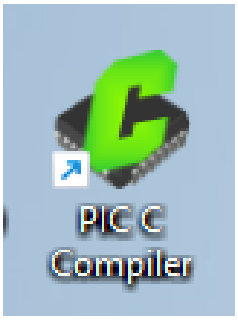


Figura 1. Software CCS Compiler.

Paso N. º 2: Una vez aparece la ventana del software, se debe crear un nuevo programa. Dirigiéndose a file >> New >> Source File

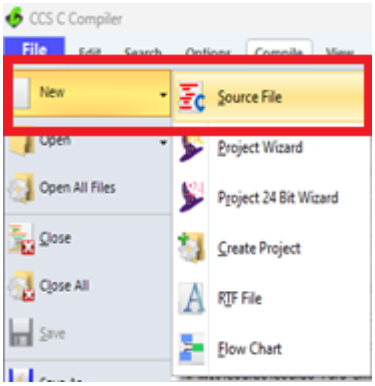


Figura 2. Creación del Nuevo archivo.

Paso N. º 3: Se debe elegir la carpeta donde se guardará el archivo, se le coloca un nombre y se da clic izquierdo en guardar.

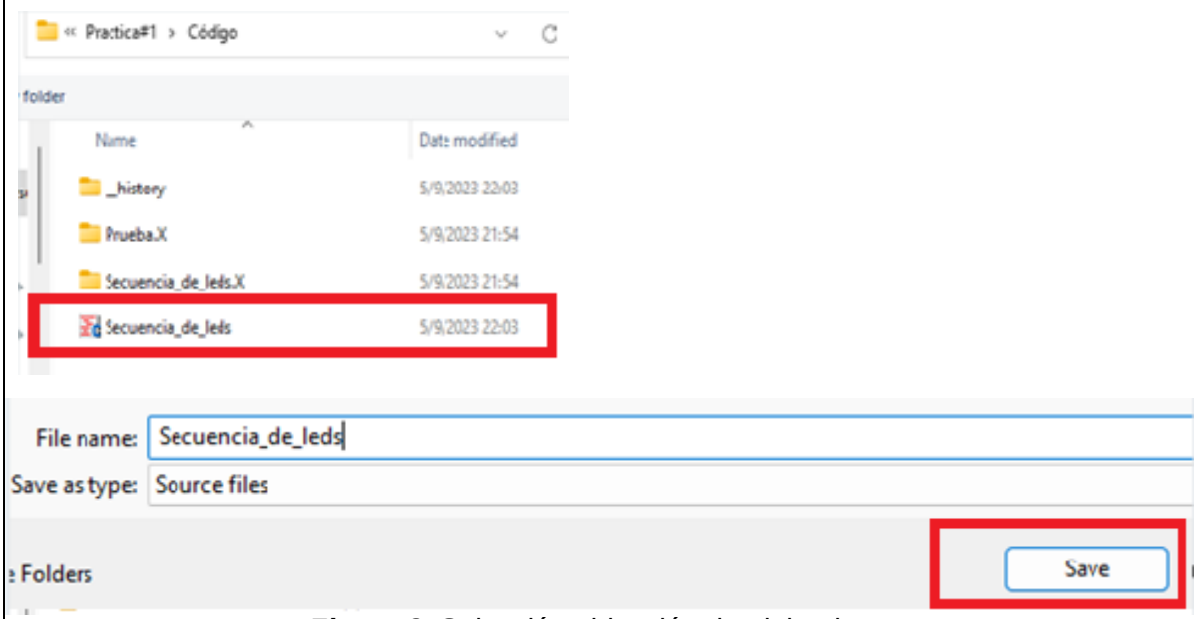


Figura 3. Selección ubicación de alojamiento.

Paso N. º 4: Una vez creado el archivo se coloca la programación (la programación de cada una de las prácticas se colocará en el Anexo B Resolución de Prácticas), una vez colocada la programación nos dirigimos a la pestaña Compile >> Compile y le damos clic izquierdo al icono de engranaje (Compile).

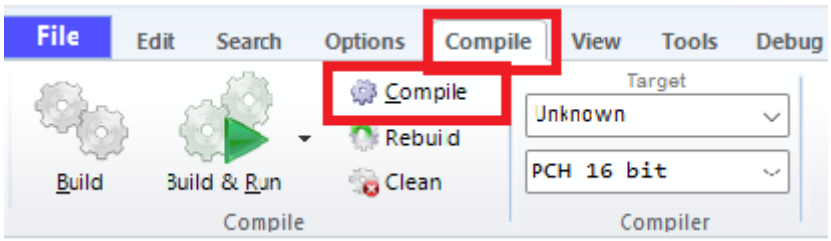


Figura 4.Compilación del código.

Paso N. ° 5: Después de que aparezca el cuadrado azul de confirmación de la compilación, se habrá creado el archivo .hex en la misma carpeta donde se encuentra el archivo fuente de la práctica.

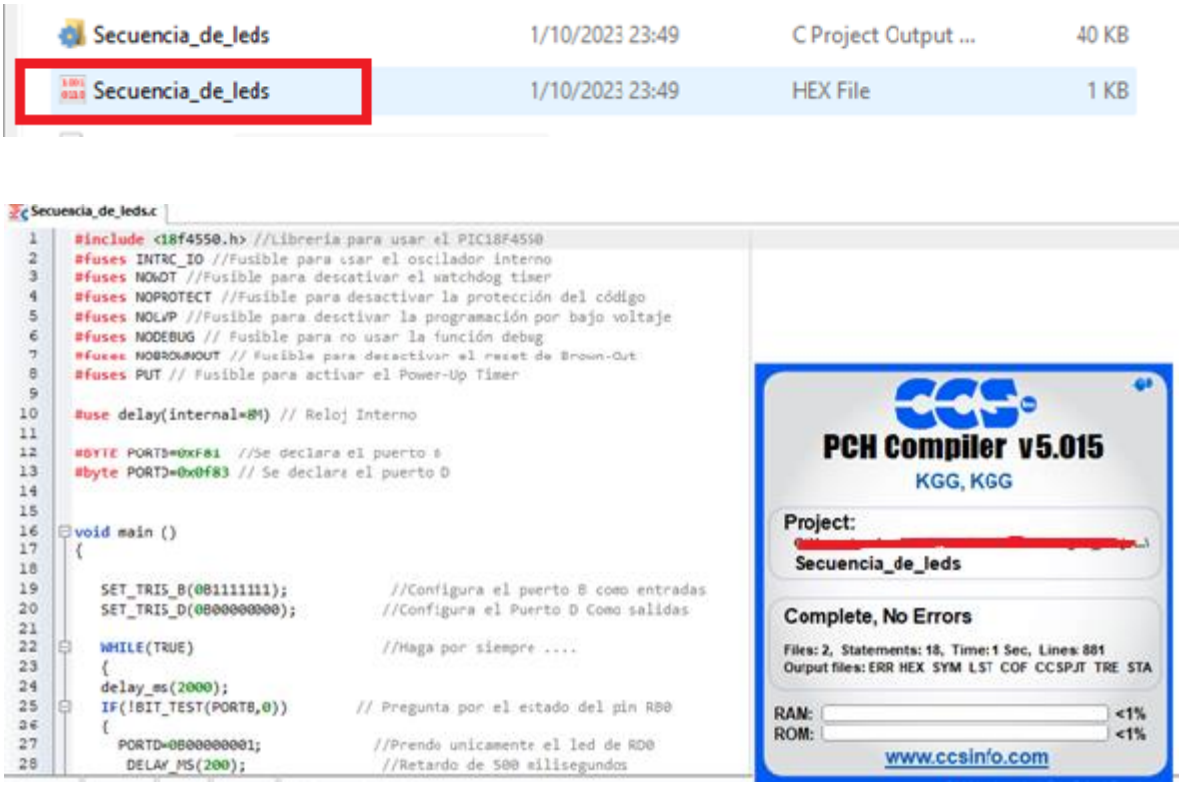


Figura 5. Compilación verificación de error.

Paso N.º 6: Posterior a eso conectamos el dispositivo picKit3 primero al módulo y luego al Computador.

Figura 6: Conexión PICkit 3.

Paso N. º 7: Una vez conectado procedemos a abrir el software dando clic izquierdo en el icono del escritorio, y si no existe ningún problema se debería visualizar en la pantalla en el apartado de device PIC18F4550.

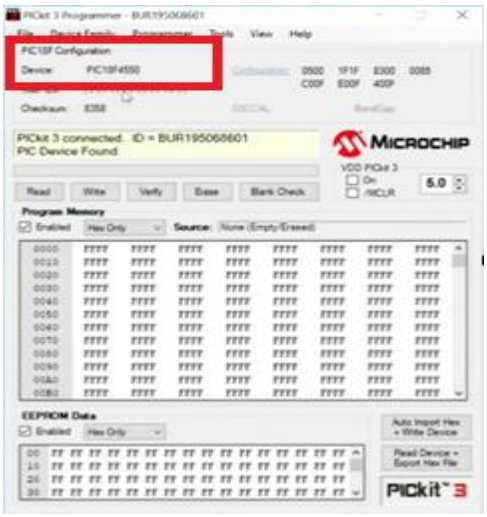
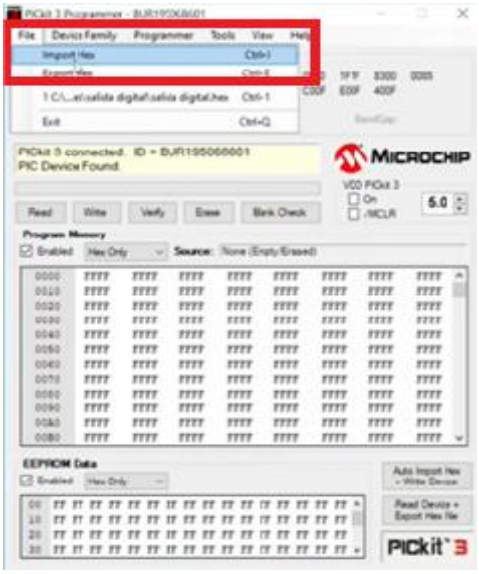


Figura 7. Selección Software PICkit 3.

Paso N. º 8: Continuamos con la importación del archivo .hex generado anteriormente, buscamos en la carpeta donde se encuentre el archivo.



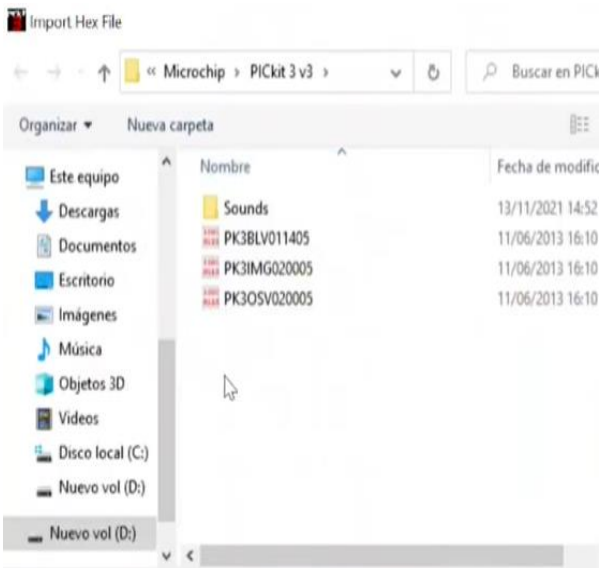


Figura 8. Búsqueda archive .hex.

Paso N. º 9: Después de seleccionar el archivo nos aparece en la ventana del software, archivo completamente importado.

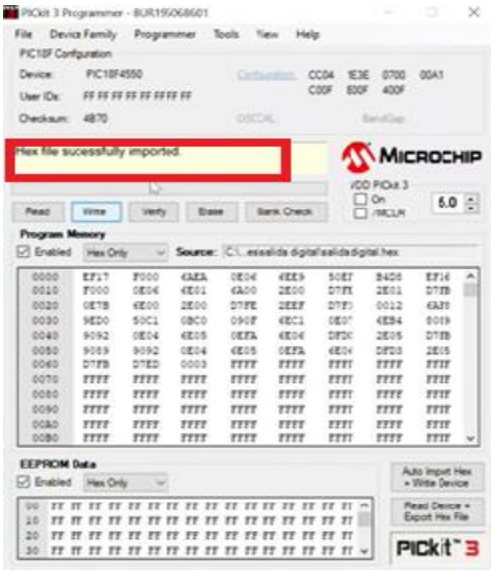


Figura 9.Importación código .hex.

Paso N. º 10: Finalmente Damos clic izquierdo en el botón Write, cuando se culmina nos muestra una pantalla verde que nos indica programación exitosa (Programming Succesful).

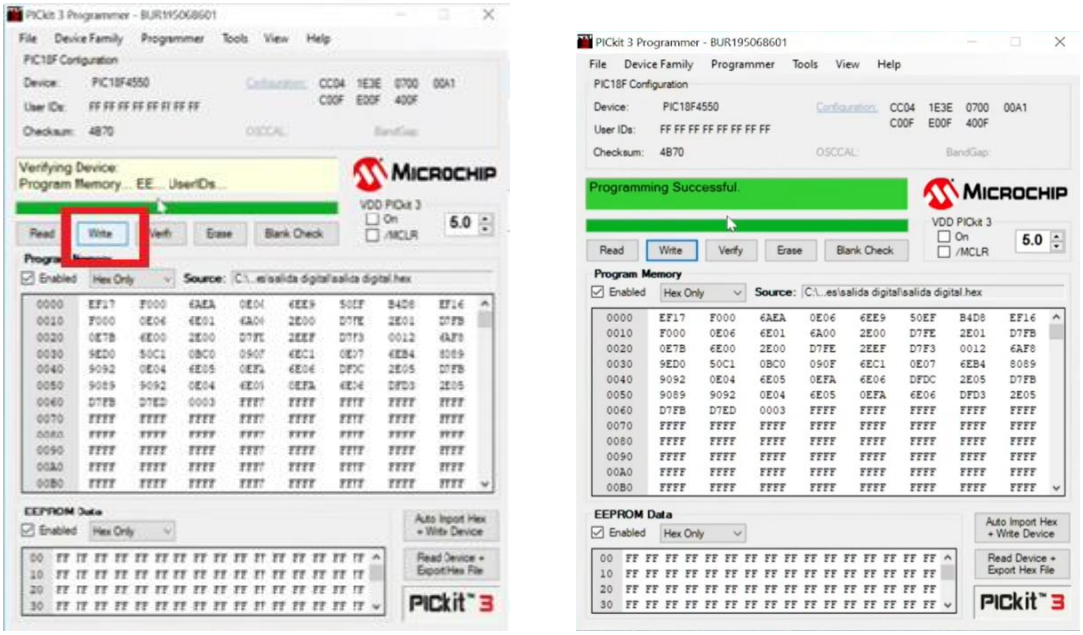


Figura 10. Programación código exitosa.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 1**
NOMBRE PRÁCTICA:	Inicio de una secuencia de encendido de leds mediante pulsadores, utilizando bucles for.	

1. **Objetivo General.**

   Implementar un programa para reforzar los conocimientos de bucles.

2. **Objetivos Específicos.**

1. Implementar un programa con bucle for.
2. Implementar en el mismo programa el encendido de salidas digitales.

3. **Marco Teórico.**

   Microcontroladores y PIC18F4550:

   Los microcontroladores son dispositivos electrónicos programables utilizados para controlar sistemas y realizar tareas específicas. El PIC18F4550 es un microcontrolador ampliamente utilizado en aplicaciones embebidas debido a su versatilidad y capacidad de procesamiento.

   Temporización y Retardos:

   La función delay_ms se utiliza para introducir retardos en milisegundos. Esto es útil para crear tiempos de espera o pausas entre acciones, como el destello de los LEDs.

   Propósito de las Secuencias de Luces:

   Las secuencias de luces tienen un propósito visual y pueden utilizarse en aplicaciones como indicadores, efectos visuales o señales de estado. En este caso, la secuencia de luces proporciona una respuesta visual al presionar el botón. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

La práctica funciona de la siguiente manera, El puerto B está conectado a un botón. Cuando se presiona el botón conectado a RB0, se enciende el LED conectado al pin RD0 del puerto D. Después de 2 segundos de encender el LED, comienza una secuencia de luces en el puerto D, donde el encendido del LED se mueve hacia la izquierda y luego hacia la derecha con retardos de 200 ms. La secuencia continúa hasta que se presiona nuevamente el botón en RB0, lo que reinicia el proceso.

5. Materiales.

- Módulo Didáctico.
- Jumpers conectores.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en las figuras.

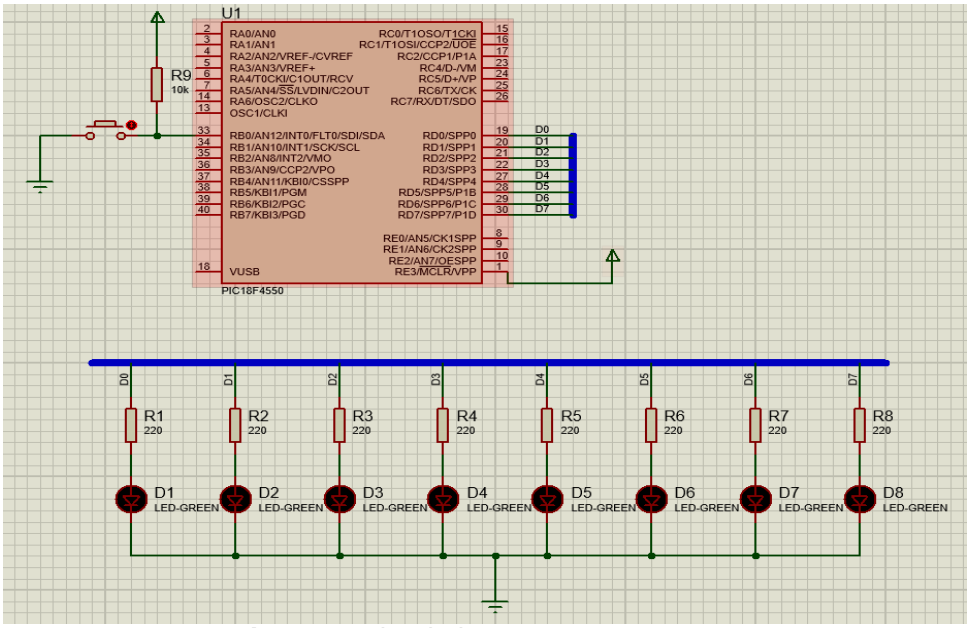


Figura 1. Simulation en Proteus.

Pines del PIC	Pines del Pulsador/LEDs
D0	R1-LED1
D1	R2-LED2
D2	R3-LED3
D3	R4-LED4
D4	R5-LED5
D5	R6-LED6
D6	R7-LED7
D7	R8-LED8
B0	R31-PULLUP7

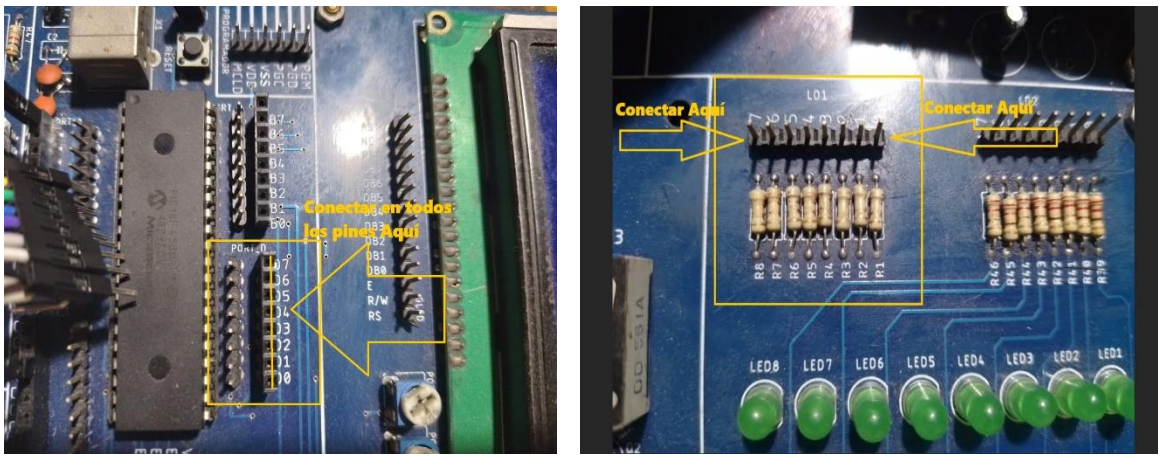


Figura 2. Conexiones del Puerto D a los LEDs.

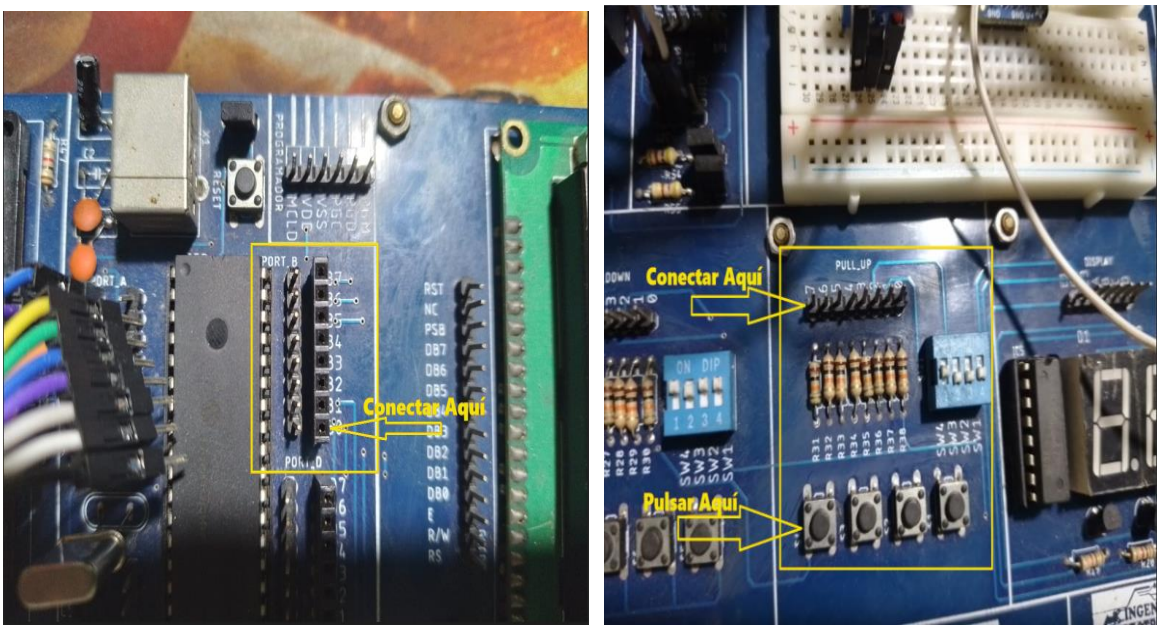


Figura 3. Conexión del Pin B0 a Botón a Pull_UP.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

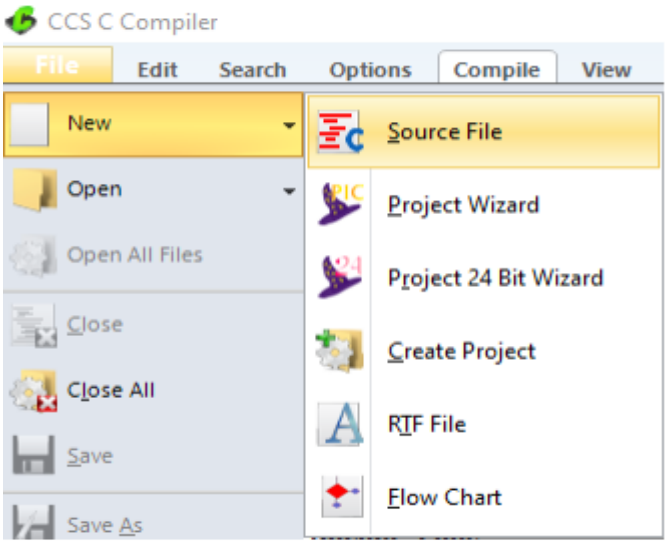


Figura 4. Crear un nuevo proyecto en el ide de CCS C Compiler.

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18f4550.h> //Libreria para usar el PIC18F4550
#fuses INTRC_IO //Fusible para usar el oscilador interno
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(internal=8M) // Reloj Interno

#BYTE PORTB=0xF81  //Se declara el puerto B
#byte PORTD=0x0f83 // Se declara el puerto D


void main ()
{

   SET_TRIS_B(0B1111111);            //Configura el puerto B como entradas
   SET_TRIS_D(0B00000000);          //Configura el Puerto D Como salidas

   WHILE(TRUE)                      //Haga por siempre ....
   {
   delay_ms(2000);
   IF(!BIT_TEST(PORTB,0))        // Pregunta por el estado del pin RB0
   {
     PORTD=0B00000001;             //Prendo unicamente el led de RD0
      DELAY_MS(200);                //Retardo de 500 milisegundos
     WHILE (!BIT_TEST(PORTD,7))    //Haga mientras el LED RB7 se encuentre apagado
     {
        PORTD=PORTD<<1;            //Rote hacia la izquierda una unidad
        DELAY_MS(200);             //Retardo de 500 milisegundos
     }
     DELAY_MS(200);                //Retardo de 500 milisegundos
     WHILE (!BIT_TEST(PORTD,0))    //Haga mientras el LED RB0 se encuentre apagado
     {
        PORTD=PORTD>>1;            //Rote hacia la derecha una unidad
        DELAY_MS(200);             //Retardo de 500 milisegundos
     }
     }
   }
}
```

Figura 5. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

- Se incluye la librería `<18f4550.h>`, que es específica para el PIC18F4550.

- Se configuran los fusibles del microcontrolador utilizando directivas `#fuses`. Estos fusibles controlan aspectos importantes como el oscilador, el watchdog timer y la protección del código.

2. Configuración de Reloj Interno:

- Se configura el reloj interno del PIC18F4550 a 8 MHz mediante `#use delay(internal=8M)`. Esto establece la frecuencia de funcionamiento del microcontrolador.

3. Declaración de Puertos:

- Se declaran los puertos B y D utilizando las directivas `#BYTE`. Esto permite acceder y manipular los registros de estos puertos de manera más conveniente.

4. Función Principal (`main`):

- La función principal del programa se inicia con `void main()`.

5. Configuración de Puertos como Entradas o Salidas:

- Se configura el puerto B como entradas (bits 0-6) y el puerto D como salidas. Esto se hace utilizando `SET_TRIS_B` y `SET_TRIS_D` para configurar los registros TRIS correspondientes.

6. Bucle Principal:

- Se inicia un bucle principal con `WHILE(TRUE)`, que se ejecutará continuamente.

7. Espera y Detección de Entrada:

- El programa espera durante 2 segundos utilizando `delay_ms(2000)` y luego verifica el estado del pin RB0 en el puerto B.

8. Control de LEDs en el Puerto D:

- Si el pin RB0 está en estado bajo (presionado), se enciende el LED en el pin RD0 del puerto D.

9. Secuencia de Luces:

- Se inicia una secuencia de luces en el puerto D: el encendido del LED se mueve hacia la izquierda y luego hacia la derecha, mientras se aplican retardos de 200 ms entre cada cambio de estado.

10. Fin del Bucle Principal:

- Una vez que el LED llega al extremo derecho y al extremo izquierdo, el programa vuelve al bucle principal y espera otra entrada en RB0.

Paso N.º 4: Compilar: **Según Practica # 0**

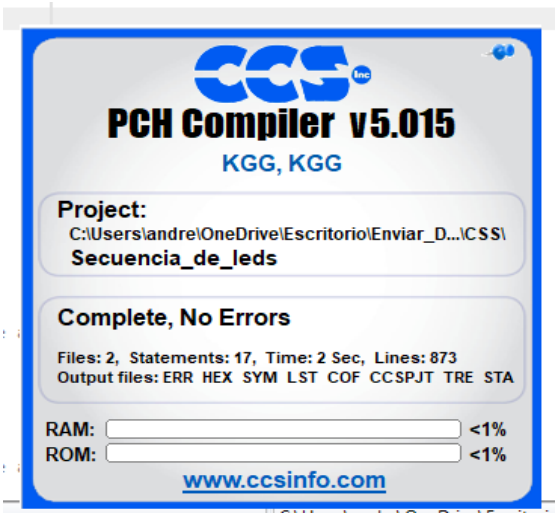
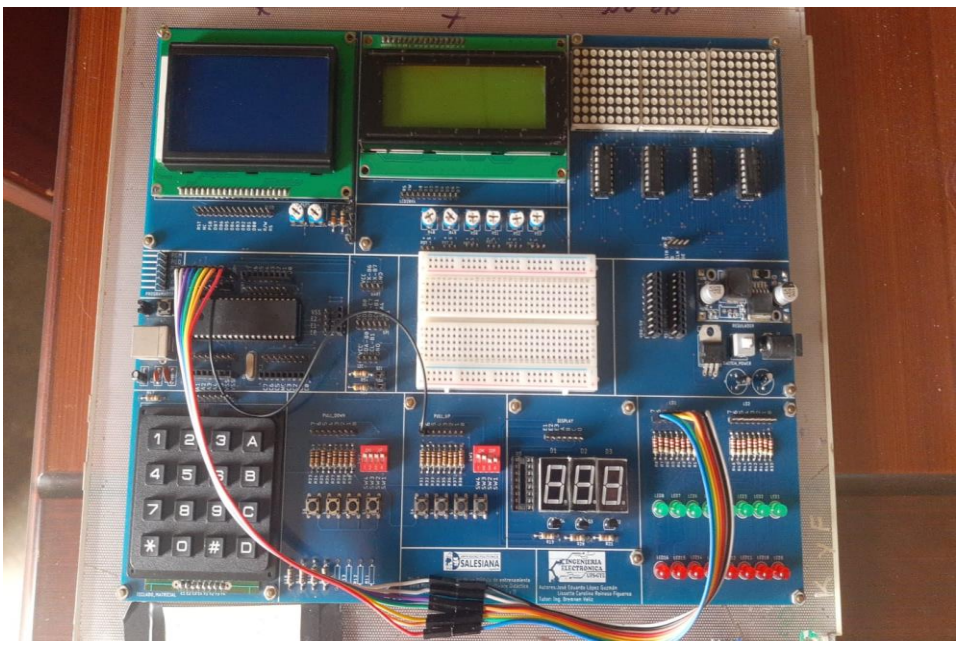


Figura 6. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 7. Conexión Pickit3 al Módulo.

Paso N. º 6: Finalización del proceso.



Resultado.

Figura 8. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 2**
NOMBRE PRÁCTICA:	Manejo de ADC-DAC con potenciómetros.	

1. Objetivo General.

Implementar un programa para reforzar los conocimientos de Convertidores Analógico-Digital Digital-Analógico.

2. Objetivos Específicos.

1. Implementar un programa con ADC.
2. Implementar un programa con DAC.

3. Marco Teórico.

Analog-to-Digital Converter (ADC):

El ADC permite al microcontrolador medir señales analógicas y convertirlas en valores digitales que pueden ser procesados por el microcontrolador. En este caso, se configura el ADC para tener una resolución de 10 bits, lo que significa que puede representar 1024 valores diferentes (2^10).

Medición de Voltaje:

El objetivo principal del código es medir voltajes analógicos utilizando el ADC. Esto es útil en muchas aplicaciones donde se necesita monitorear o controlar dispositivos que generan o responden a señales analógicas, como sensores de temperatura, presión, luz, etc.

Configuración del ADC:

El código configura el ADC para trabajar con el reloj interno del microcontrolador (`adc_clock_internal`) y establece que todo el puerto A sea analógico (`setup_adc_ports(all_analog)`). Esto prepara el ADC para medir señales analógicas en el pin RA0. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

El pin RA0 se utiliza para medir un voltaje analógico, que proviene de un potenciómetro.

El valor del voltaje medido se convierte a una representación numérica y se muestra en un LCD conectado al puerto D del microcontrolador.

La medición de voltaje se actualiza cada 100 ms, lo que permite monitorear continuamente el voltaje en tiempo real.

Este código se puede utilizar en aplicaciones de monitoreo de voltaje, como sistemas de medición, sensores de temperatura, sistemas de control, entre otros.

El LCD proporciona una interfaz de usuario para visualizar el valor del voltaje en un formato legible.

5. Materiales.

- Módulo didáctico.
- Jumpers Conectores.
- Potenciómetro de 10 KΩ.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

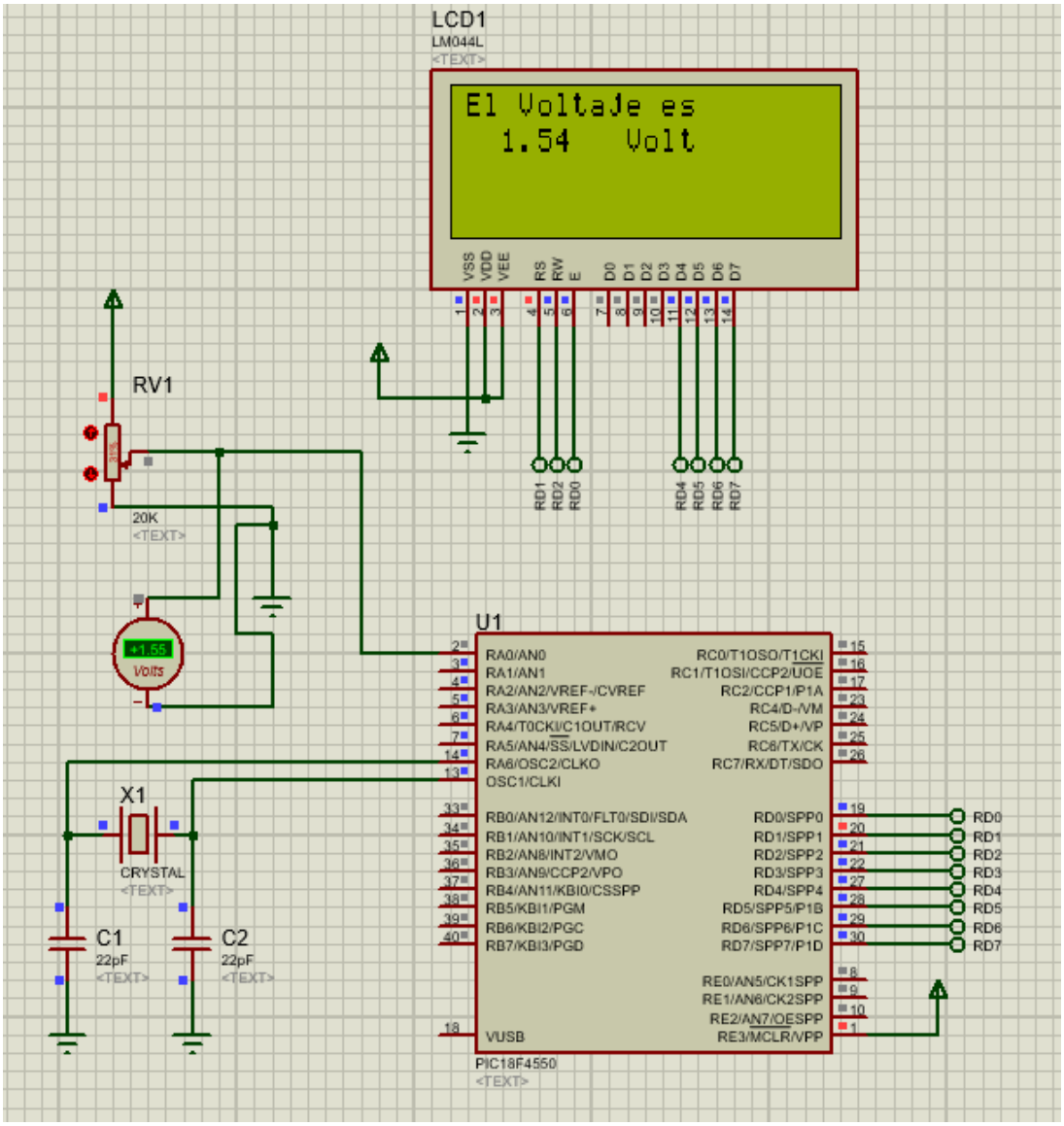


Figura 1. Simulación en Proteus.

Pines del PIC	Pines de la Pantalla/Potenciómetro
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
A0	3-POTENCIÓMETRO
Pines del Potenciómetro	
1-POTENCIÓMETRO	VDD-5V
2-POTENCIÓMETRO	VSS-0V

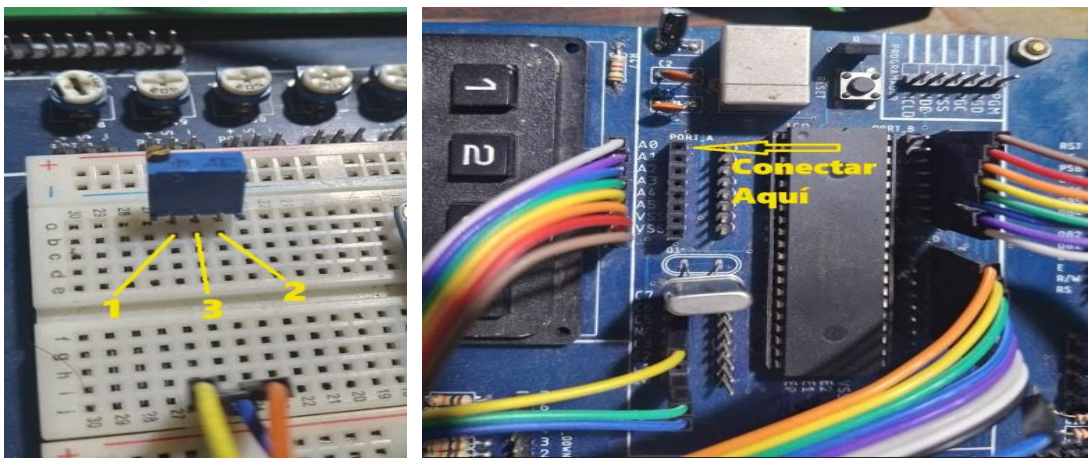


Figura 2. Conexión del potenciómetro al PIC.

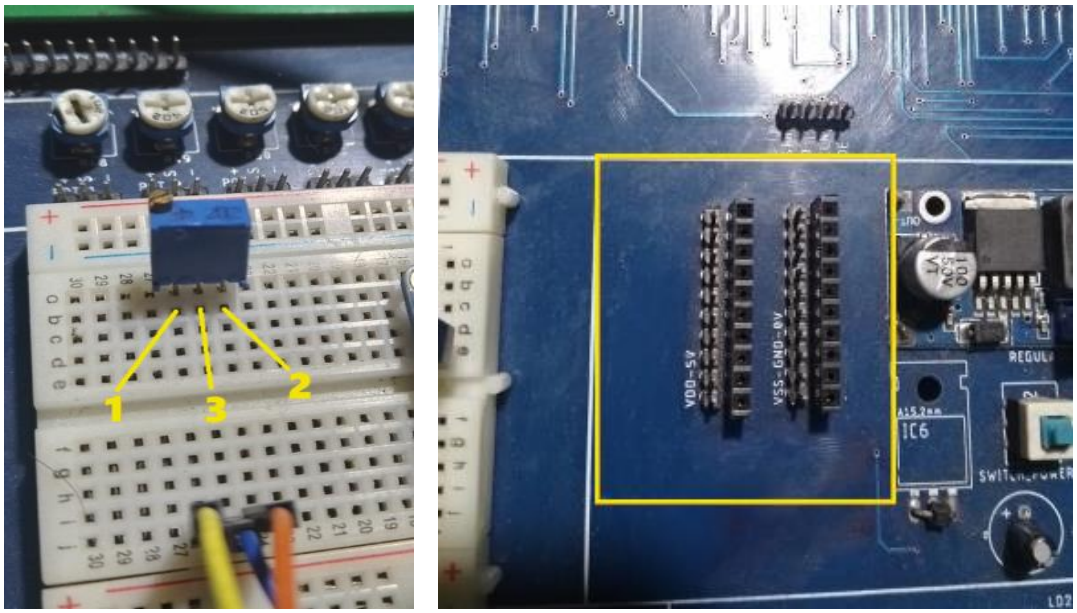


Figura 3. Conexión del potenciómetro a la alimentación.

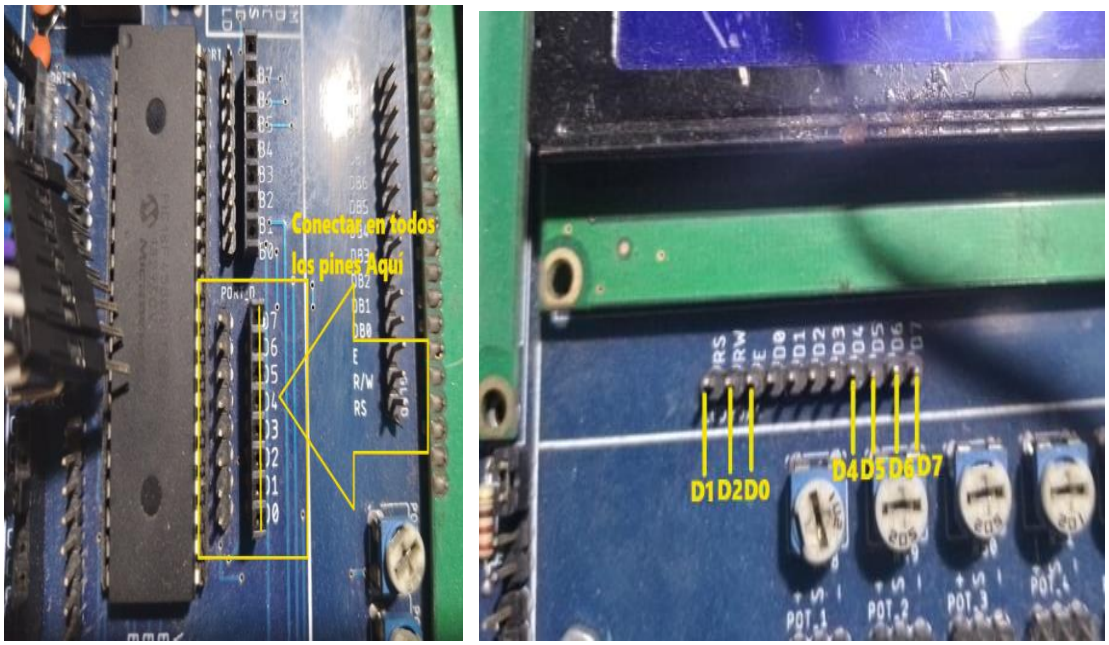


Figura 4. Conexiones del Puerto D a la pantalla LCD.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

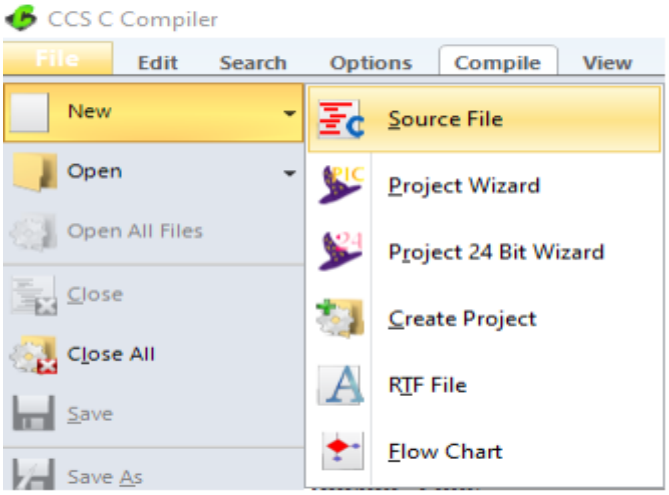


Figura 5. Crear un nuevo proyecto en el ide de CCS C Compiler.

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18F4550.h> //Libreria para usar el PIC18F4550
#device adc=10 //seteamos el adc de 10 bits
#fuses INTRC_IO //Fusible para usar el oscilador interno
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer
#use delay(clock=20Mhz) //reloj externo

#include <lcd.c> // incluimos la libreria del LCD

#BYTE PORT_A = 0xf80 //Declaramos el puerto A
#BYTE PORT_D= 0xF83 //Declaramos el puerto B

# USE FAST_IO(A)//usamos el puerto A en fast i/o
# USE FAST_IO(D)//usamos el puerto D en fast i/o
```

```c
//Declaramos los pines a usar por la pantalla LCD
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////////////////////////////////////////////
#define vref 5.00     /* Definimos el voltaje de referencia a medir = 5V*/
long bits;     //Variable almacena los bits
float volt;     //Almacena el voltaje
```

```c
void main()
{

   set_tris_a(0b00000001);          //Pongo el RA0 como entrada
   set_tris_d(0);                   //Pongo el PuertoD como Salida
   setup_adc_ports(all_analog);     //Pongo todo el puerto A analogo
   setup_adc(adc_clock_internal);   //Selecciono reloj interno para conversion
      lcd_init();                      //Inicializo el LCD
   lcd_putc("\f");                  //Borro el LCD
   lcd_gotoxy(1,1);             //Ubiquese en la posicion 1,1
   lcd_putc("El Voltaje es ");
   while(1)
   {
       set_adc_channel(0);          //Selecciono el canal 0 (RA0)
       delay_ms(1);                 //llamo retardo de 1 ms
       bits=read_adc();             //Guarde el dato del ADC en volts ****ADC
       volt=bits*((float)vref/(float)1023);//Conversion de bits a voltaje ******DAC
       lcd_gotoxy(2,2);             //Ubiquese en la posicion 2,2
       printf(lcd_putc," %f   Volt ",volt);  //Muestra el valor numerico de la conversión
       delay_ms(100);
   }
}
```

Figura 6. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería específica para el microcontrolador PIC18F4550 y estableciendo configuraciones de fusibles. Esto determina aspectos importantes de la configuración del microcontrolador, como el oscilador y la protección del código.

2. Configuración del Convertidor Analógico-Digital (ADC):

Se establece el ADC para trabajar con una resolución de 10 bits y se configura para utilizar el reloj interno (`adc_clock_internal`). Esto prepara el ADC para convertir señales analógicas en digitales.

3. Inclusión de la Librería LCD y Declaración de Puertos:

Se incluye la librería para controlar un LCD y se declaran los puertos A y D como puertos de entrada/salida.

4. Definición de Pines para el LCD:

Se definen los pines del microcontrolador que se conectarán al LCD, incluyendo pines para controlar la habilitación (`ENABLE`), la selección del registro (`RS`), la lectura/escritura (`RW`), y los pines de datos (`DATA4` a `DATA7`).

5. Inicialización del LCD:

Se inicializa el LCD y se borra la pantalla (`lcd_putc("\f")`). Luego, se posiciona el cursor en la parte superior izquierda del LCD (`lcd_gotoxy(1,1)`).

6. Bucle Principal:

El programa entra en un bucle infinito (`while(1)`) que se ejecuta continuamente.

7. Configuración de Puertos:

Se configuran los puertos A y D para las operaciones del programa. El pin RA0 se configura como entrada (`set_tris_a(0b00000001)`), y el puerto D se configura como salida (`set_tris_d(0)`).

8. Medición de Voltaje:

Se selecciona el canal del ADC (RA0) usando `set_adc_channel(0)`, se introduce un pequeño retraso (`delay_ms(1)`), se realiza una conversión ADC (`bits = read_adc()`), y se convierte el valor de bits a voltaje (`volt`). Luego, el voltaje se muestra en el LCD.

9. Visualización en el LCD:

El voltaje medido se muestra en el LCD (`printf(lcd_putc, " %f  Volt ", volt)`).

10. Espera y Repetición:

El programa espera durante 100 ms antes de realizar otra medición. Esto se hace para que el voltaje no se actualice demasiado rápido en la pantalla LCD.

Paso N.º 4: Compilar **Según Practica # 0** :

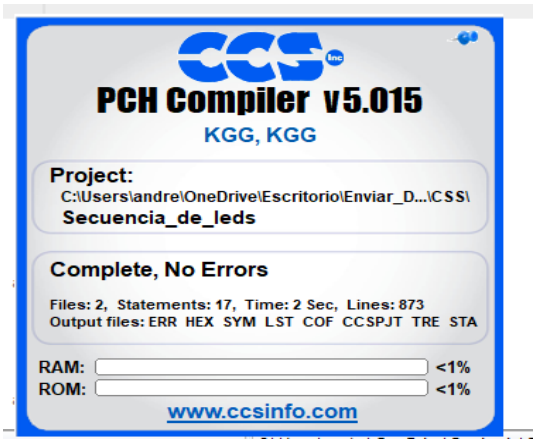


Figura 7. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 8. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

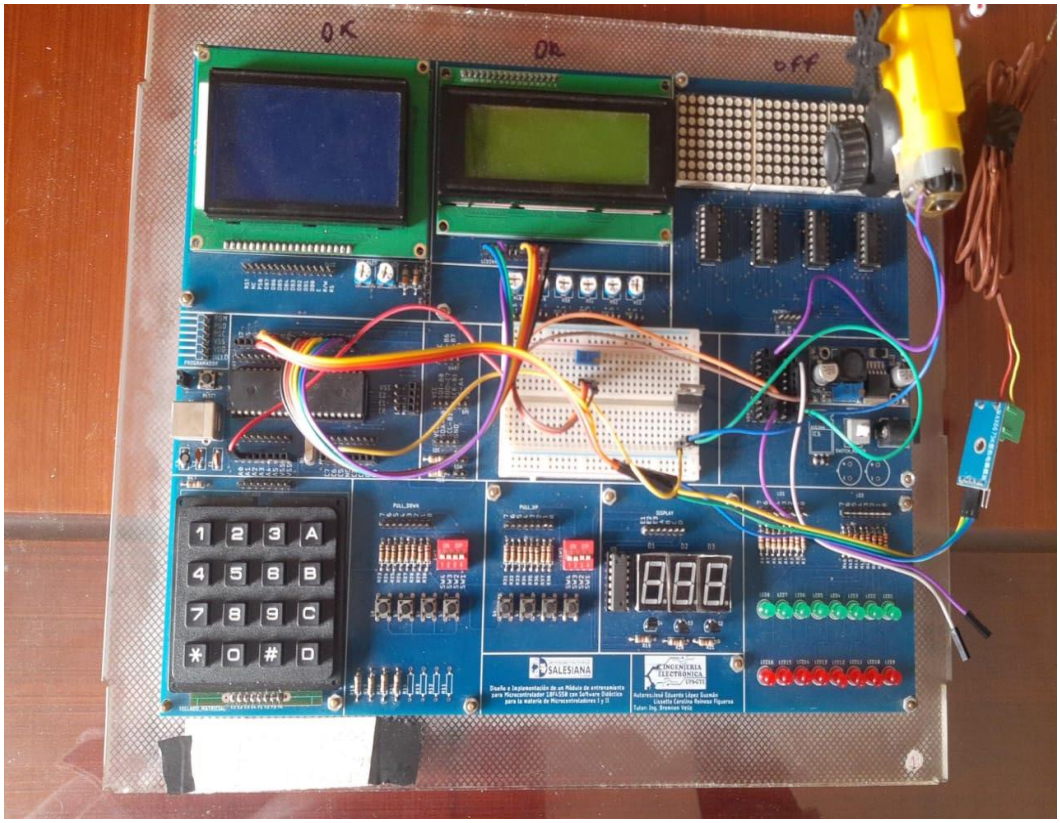


Figura 9. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 3**
NOMBRE PRÁCTICA:	Implementación de interrupciones internas y externas usando pulsadores.	

1. **Objetivo General.**

Implementar un programa para reforzar los conocimientos acerca de las interrupciones.

2. **Objetivos Específicos.**

1. Implementar un programa con una interrupción externa por pulsador.
2. Implementar un programa con interrupción interna por comunicación serial.

3. **Marco Teórico.**

Interrupciones Externas:

Las interrupciones externas son eventos generados por componentes externos al microcontrolador, como pulsadores, sensores u otros dispositivos. Estas interrupciones permiten al microcontrolador responder de manera inmediata a eventos específicos sin tener que esperar en un bucle principal. En este código, se utiliza una interrupción externa para detectar un cambio en el estado de un pin específico del puerto B.

Función de Interrupción:

Cuando se produce un cambio en el estado del pin específico del puerto B, se ejecuta la función de interrupción. Esta función enciende y apaga los LEDs en una secuencia definida, creando un efecto visual.

Interrupciones Internas:

Las interrupciones internas son eventos generados dentro del microcontrolador que pueden interrumpir la ejecución normal del programa principal. Se utilizan para manejar eventos críticos o tareas de alta prioridad sin tener que esperar a que se complete una tarea en curso. En este código, se utiliza una interrupción serial (`#int_rda`) para recibir comandos a través de la comunicación serial RS232.

Comunicación Serial RS232:

La comunicación serial RS232 es un protocolo utilizado para transmitir datos serie a través de un cable o conexión. En este código, se configura la comunicación serial para recibir comandos del usuario a través de un puerto serie. Los comandos pueden ser enviados desde un terminal o un programa de comunicación serie en una computadora para controlar acciones en el microcontrolador.

Medición de Temperatura:

El código incluye una función de medición de temperatura utilizando un potenciómetro simulando un sensor conectado al pin AN0 (RA0). Se utiliza el Convertidor Analógico-Digital (ADC) para convertir la señal analógica del "sensor" en un valor digital que representa la temperatura. Esta función demuestra cómo se pueden utilizar los periféricos internos del microcontrolador para realizar mediciones y procesar datos.

LCD con Comunicación I2C:

Se utiliza un módulo LCD controlado por I2C para mostrar información, como la temperatura medida. La comunicación I2C permite la transferencia de datos entre dispositivos de manera serial, simplificando la conexión y el control de periféricos como el LCD. Esto muestra cómo se pueden utilizar protocolos de comunicación para interactuar con dispositivos externos de manera eficiente. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

Interrupción Externa:

El puerto B estaría configurado para detectar cambios en el estado de un pin específico, conectado a un botón.

Cuando se detecta un cambio en el estado de ese pin, se ejecuta la función de interrupción "Interrupcion". En este caso, la interrupción provoca una secuencia de encendido y apagado de LEDs en el puerto D.

Los LEDs se encienden secuencialmente hacia la izquierda y luego hacia la derecha, con retardos de 200 ms entre cada cambio de estado.

El programa principal simplemente espera durante 2 segundos antes de continuar, lo que significa que se producirán interrupciones solo cada 2 segundos.

Este tipo de código es útil en aplicaciones donde se requiere una respuesta a eventos específicos, como la presión de un botón, y se desea un efecto visual

como resultado, como una secuencia de luces que indica que se ha realizado una acción.

Interrupción Interna:

El programa puede recibir comandos a través de la comunicación serial RS232, lo que permite controlar acciones como encender o apagar un LED.

Además, el programa mide y muestra la temperatura utilizando un potenciómetro simulando un sensor conectado al pin AN0 (RA0) del microcontrolador. La temperatura se muestra en el LCD y se actualiza cada segundo.

Este código se puede utilizar en aplicaciones que requieran monitoreo de temperatura y comunicación con un usuario a través de la interfaz serial.

5. Materiales.

- Módulo didáctico.
- Jumpers Conectores.
- Display LCD I2C.
- Potenciómetro de 10 kΩ.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

Interrupción Externa:

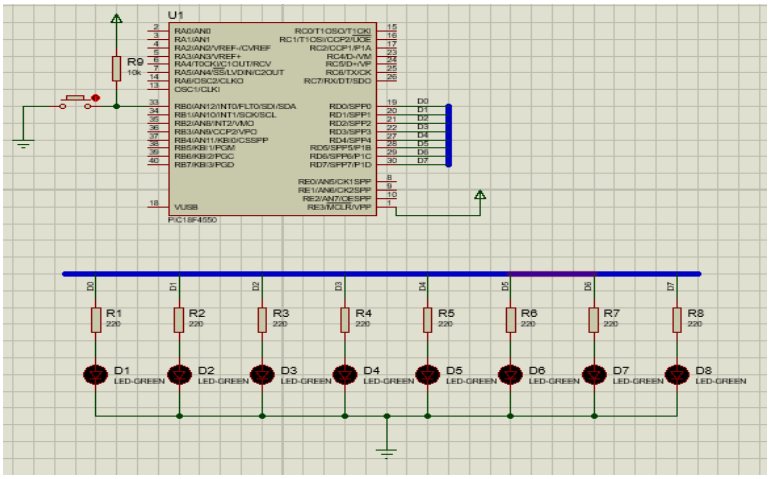


Figura 1. Simulation en Proteus.

Pines del PIC	Pines del Pulsador/LEDs
D0	R1-LED1
D1	R2-LED2
D2	R3-LED3
D3	R4-LED4
D4	R5-LED5
D5	R6-LED6
D6	R7-LED7
D7	R8-LED8
B0	R31-PULLUP7

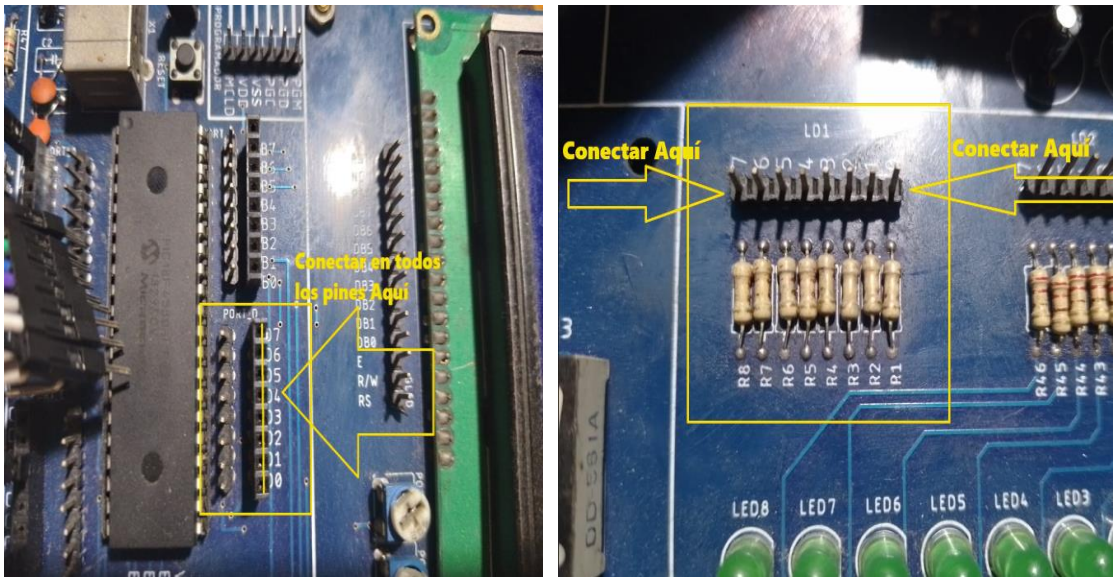


Figura 2. Conexiones del Puerto D a los LEDs.

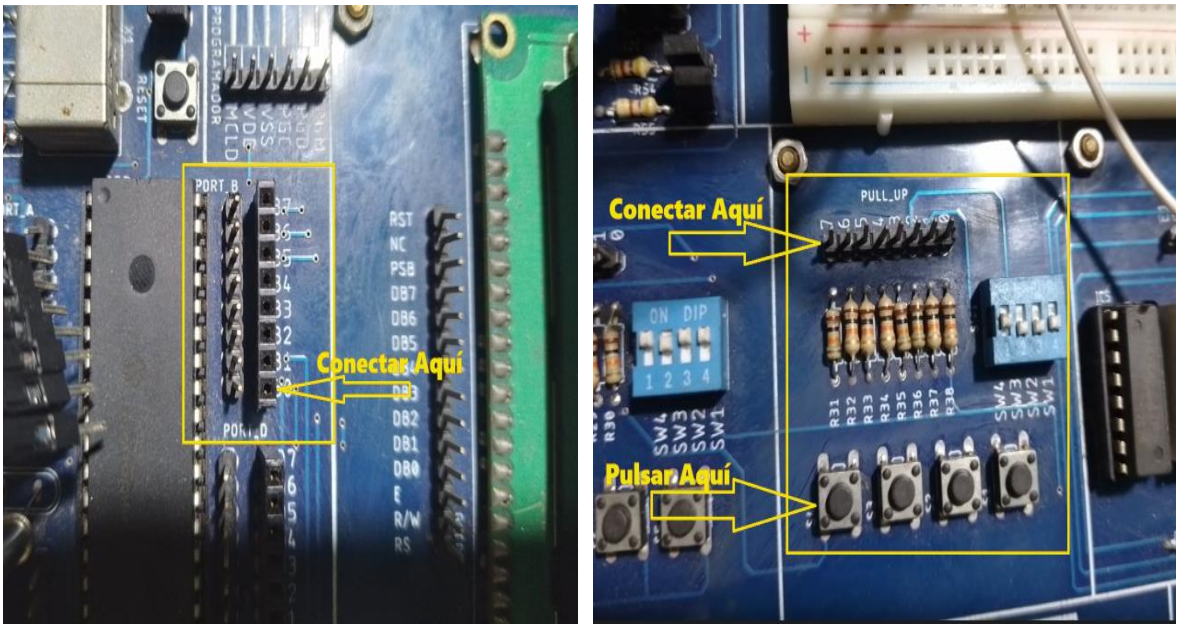


Figura 3. Conexión del Pin B0 a Botón a Pull_UP.

Interrupción Interna:

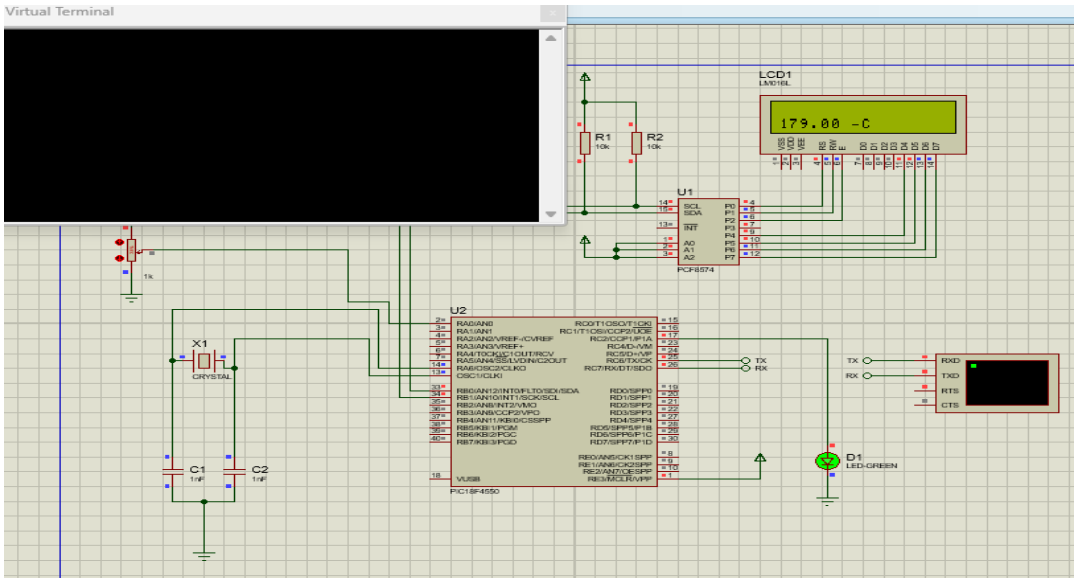


Figura 4. Simulación en Proteus.

Pines del PIC	Pines del Módulo USB/LCD/POT
A0	3-POTENCIÓMETRO
B0	SDA-LCD I2C
B1	SCL-LCD I2C
C2	R1-LED1
C6	TX-MÓDULO USB
C7	RX-MÓDULO USB
Pines del Potenciómetro	

1-POTENCIÓMETRO	VDD-5V
2-POTENCIÓMETRO	VSS-0V
Pines del LCD I2C	
VCC-LCD	VDD-5V
GND-LCD	VSS-0V

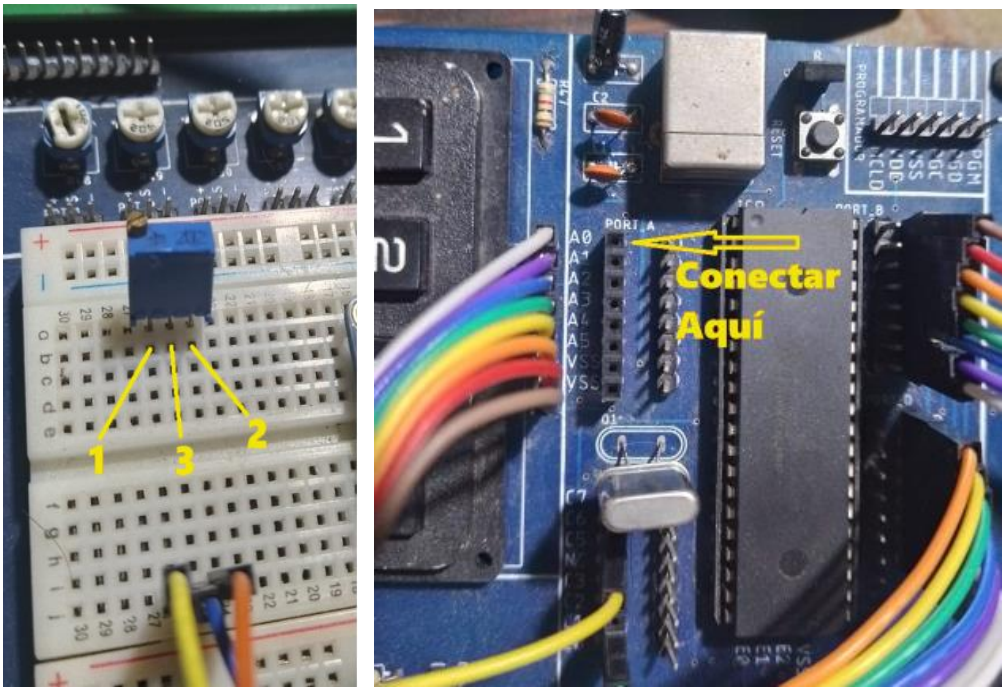


Figura 5. Conexión del potenciómetro al PIC.

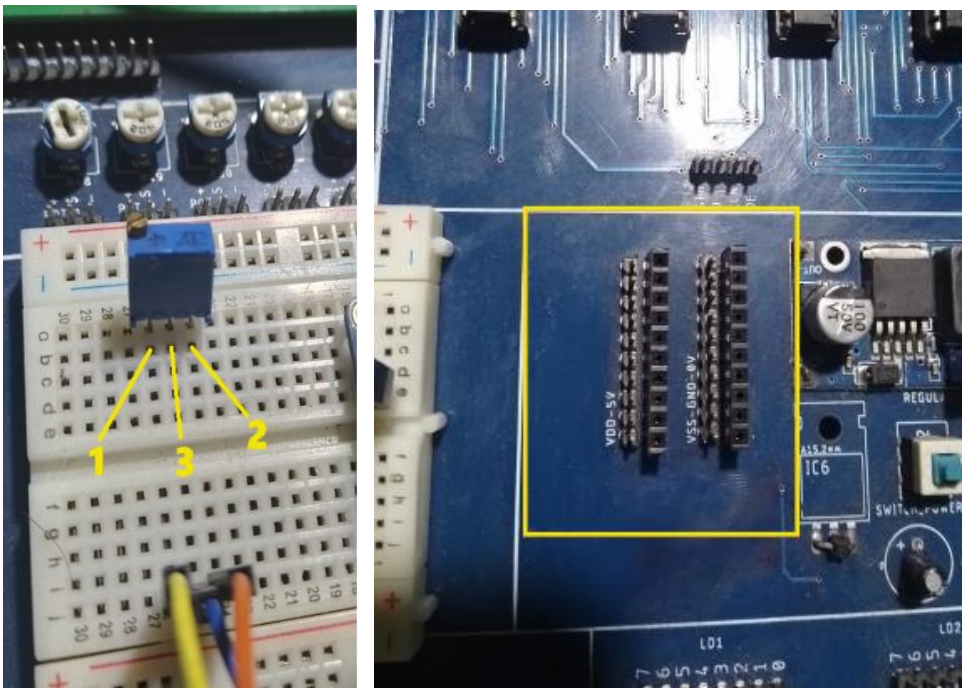


Figura 6. Conexión del potenciómetro a la alimentación.

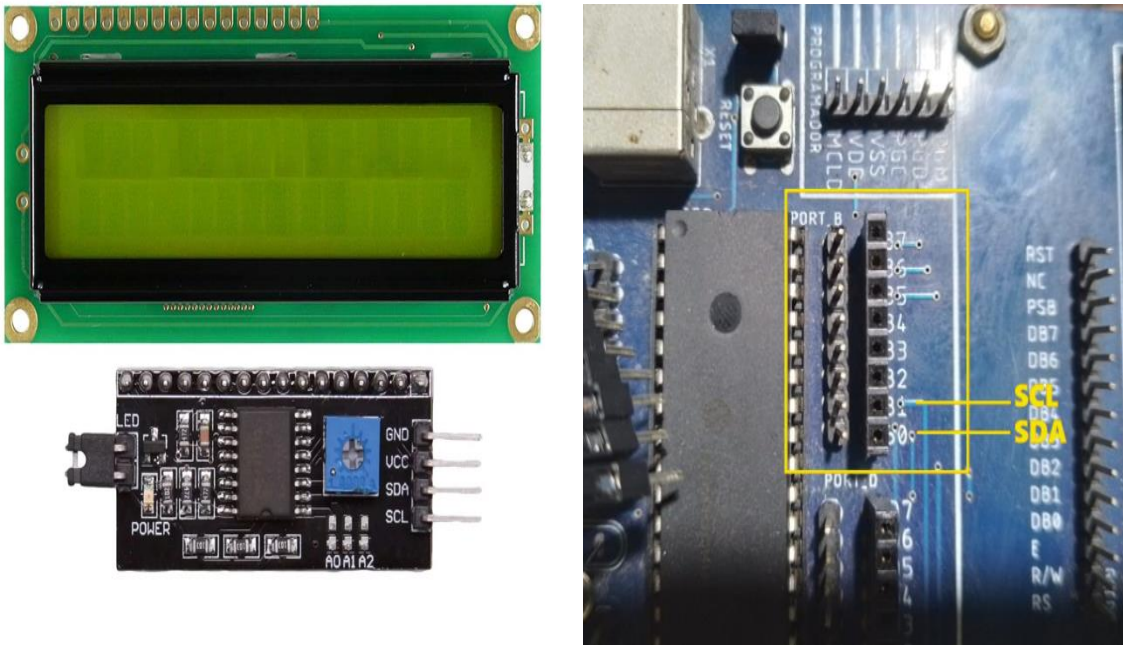


Figura 7. Conexión de la pantalla LCD I2C al PIC.

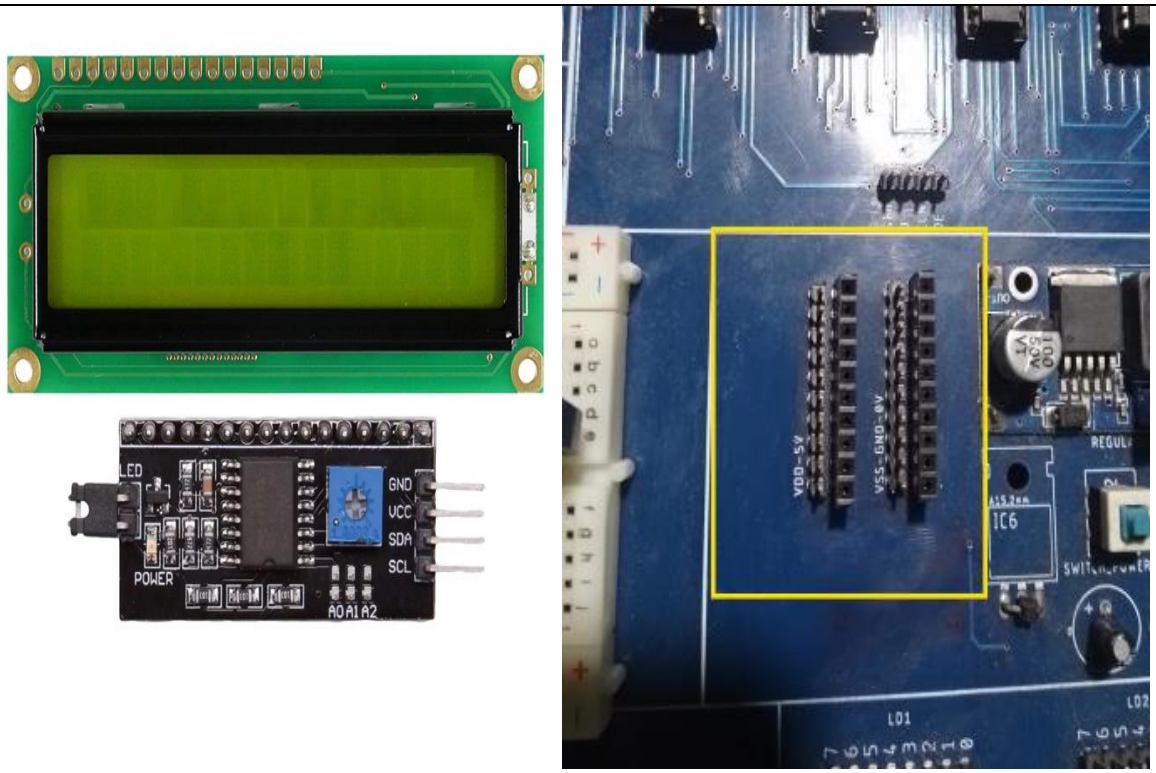


Figura 8. Conexión de la pantalla LCD I2C a la alimentación.

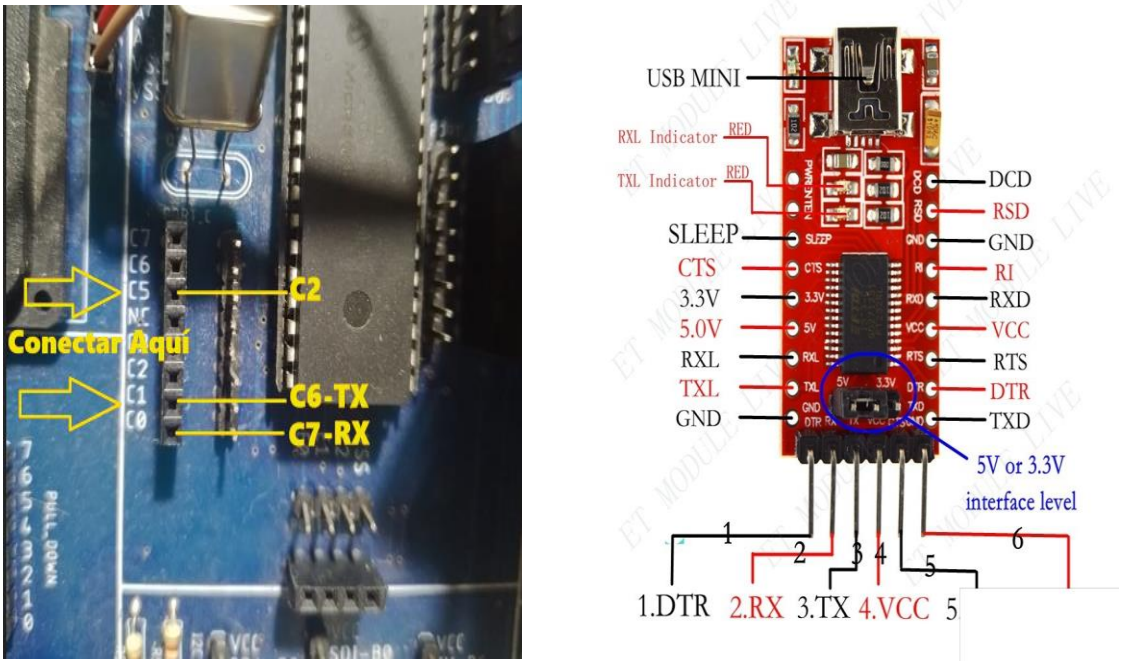


Figura 9. Conexión del PIC a la Interfaz USB.

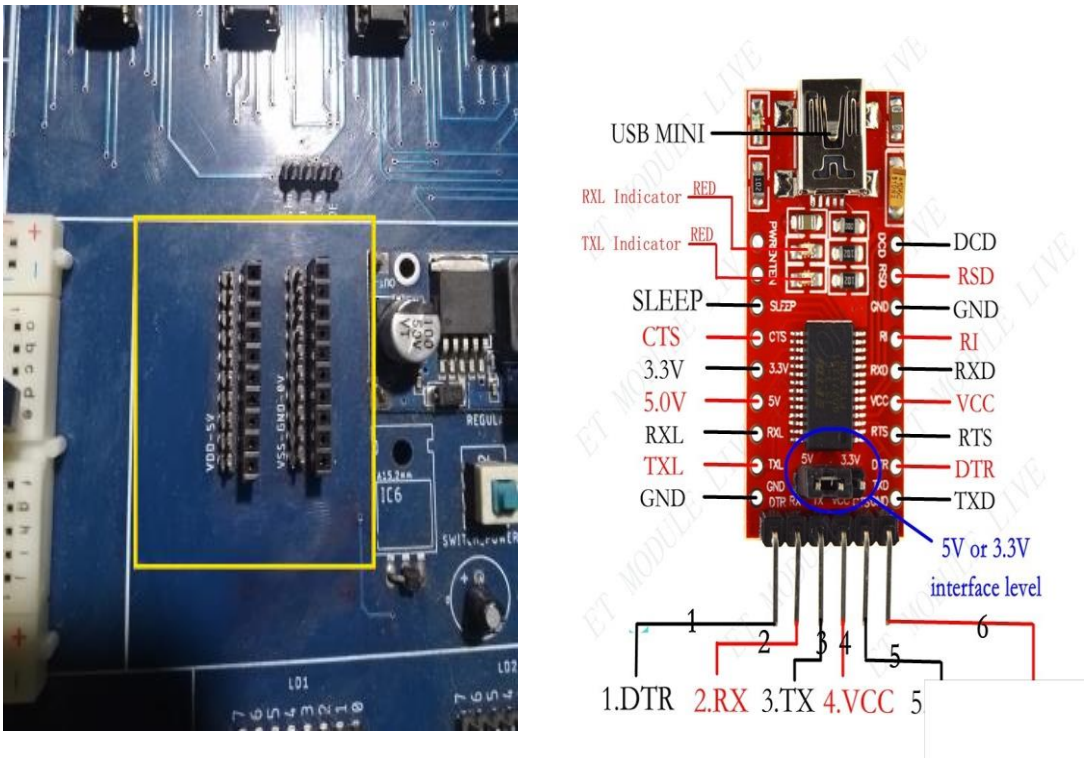


Figura 10. Conexión de la alimentación a la Interfaz USB.

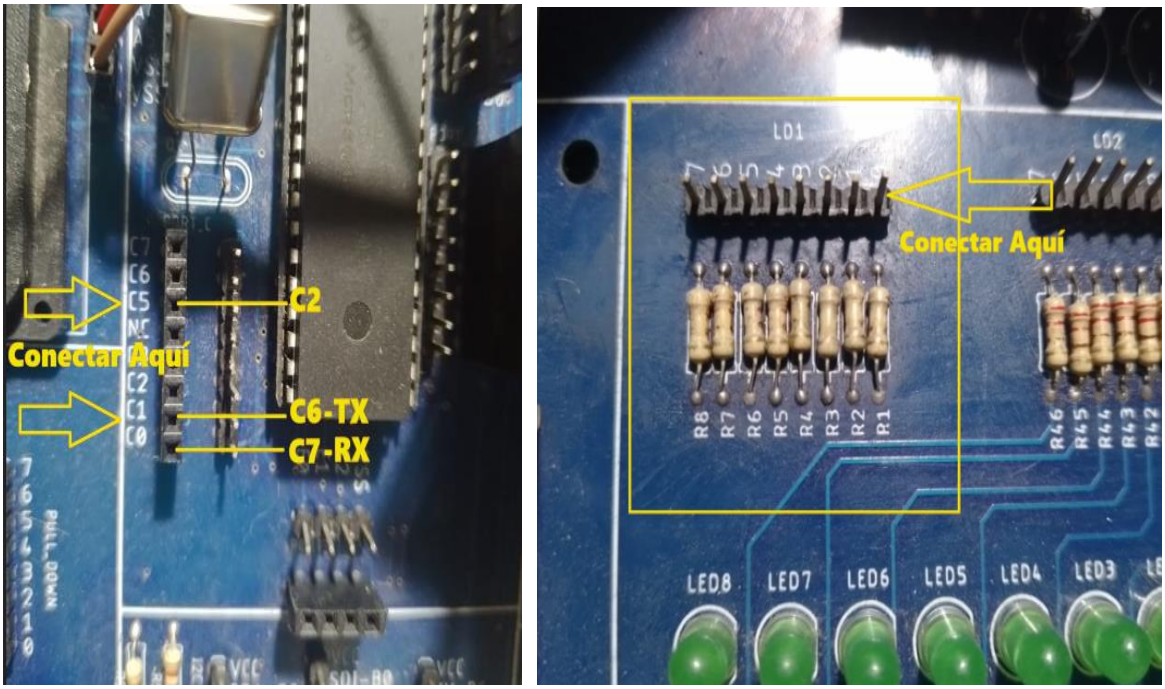


Figura 11. Conexión del PIC al LED.

Paso N.º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

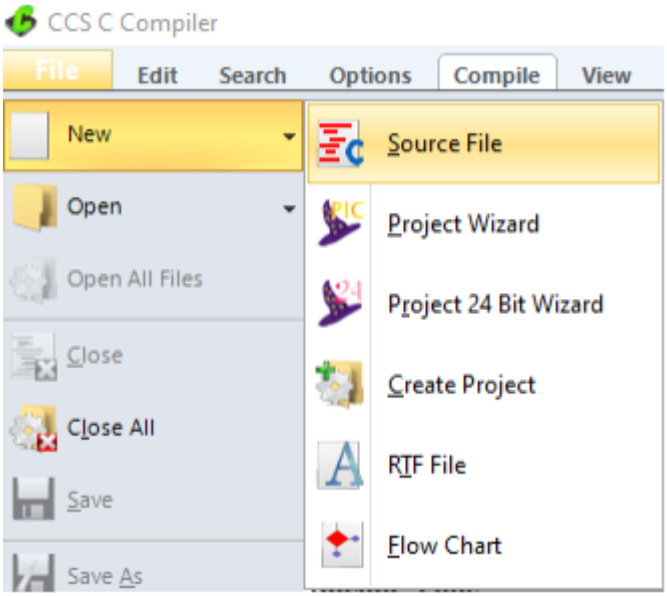


Figura 12. Crear un nuevo proyecto en el ide de CCS C Compiler.

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

Interrupción Externa:

```
#include <18f4550.h> //Libreria para usar el PIC18F4550
#fuses INTRC_IO //Fusible para usar el oscilador interno
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(internal=8M) // Reloj Interno

#byte portb=0x0f81// Identificador para el puerto B.
#byte portd=0x0f83// Identificador para el puerto D.

#int_ext
Interrupcion(){
       //DELAY_MS(200);    //Retardo

      PORTD=0B00000001;             //Prendo unicamente el led de RD0
      DELAY_MS(200);                //Retardo de 500 milisegundos
      WHILE (!BIT_TEST(PORTD,7))    //Haga mientras el LED RD7 se encuentre apagado
      {
         PORTD=PORTD<<1;            //Rote hacia la izquierda una unidad
         DELAY_MS(200);             //Retardo de 200 milisegundos
      }
      DELAY_MS(200);                //Retardo de 200 milisegundos
      WHILE (!BIT_TEST(PORTD,0))    //Haga mientras el LED RD0 se encuentre apagado
      {
         PORTD=PORTD>>1;            //Rote hacia la derecha una unidad
         DELAY_MS(200);             //Retardo de 200 milisegundos
      }
}
void main ()
{

   SET_TRIS_B(0B11111111);       //Configurar el puerto B
   SET_TRIS_D(0B00000000);          //Configura el Puerto D Como salidas
   PORTB=0;                         //Comienza con todos los Leds Apagados
   enable_interrupts(GLOBAL);
   enable_interrupts(INT_EXT);
   WHILE(TRUE)                      //Haga por siempre ....
   {
   delay_ms(2000);
   }
}
```

Figura 13. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería `<18f4550.h>`, que es específica para el microcontrolador PIC18F4550. También configura varios fusibles del microcontrolador que determinan aspectos importantes de su funcionamiento, como el oscilador, la protección del código y el Power-Up Timer.

2. Configuración del Reloj Interno:

Se configura el reloj interno del PIC18F4550 a 8 MHz utilizando `#use delay(internal=8M)`. Esto establece la velocidad de funcionamiento del microcontrolador.

3. Declaración de Puertos:

Se declaran los puertos B y D utilizando las directivas `#byte`. Esto permite acceder y manipular los registros de estos puertos de manera más conveniente.

4. Interrupción Externa:

Se declara una función de interrupción externa (`#int_ext`) llamada "Interrupcion". Esta función se ejecutará cuando se produzca una interrupción externa, que en este caso está relacionada con cambios en el estado de un pin específico.

5. Manejo de Interrupción:

En la función de interrupción, se realiza una secuencia de encendido y apagado de LEDs en el puerto D cuando se produce la interrupción externa. Los LEDs se encienden secuencialmente hacia la izquierda y luego hacia la derecha, con retardos de 200 ms entre cada cambio de estado.

6. Configuración de Puertos:

Se configura el puerto B como entradas (`SET_TRIS_B(0B11111111)`) y el puerto D como salidas (`SET_TRIS_D(0B00000000)`).

7. Espera Principal:

El programa entra en un bucle principal (`WHILE(TRUE)`) que se ejecuta continuamente con un retraso de 2 segundos (`delay_ms(2000)`).

Interrupción Interna:

```
#include <18F4550.h>  //Declaramos el PIC18F4550
#device ADC=10  //Seteamos el adc en 10 bits
#fuses INTRC_IO //Fusible para usar el oscilador interno
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses PLL1 //Fusible para multiplicar el oscilador interno por un factor de 4
#fuses CPUDIV1 // Fusible para activar el reloj divisor 1
#use delay(clock=20000000)//Declaramos el reloj externo de 20 Mhz

#byte porta = 0xf80 // Identificador para el puerto A.
#byte portb = 0xf81 // Identificador para el puerto B.
#byte portc = 0xf82 // Identificador para el puerto C.
#byte portd = 0xf83 // Identificador para el puerto D.
#byte porte = 0xf84 // Identificador para el puerto E.
#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c7, bits=8)//Declaramos la comunicación serial
#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) // Declaramos la comunicación I2C para el LCD
#include <i2c_Flex_LCD.c> //incluimos la libreria del LCD I2C
#include <stdio.h>//libreria para usar printf
#include <math.h> // libreria para claculos matemáticos

void temperatura();
char dato;
float temp;
```

```
#int_rda // entra a interrupción, ejecuta lo que mande el monitor serial
void rda_isr()
{
   dato=getc();

   switch(dato)
   {
      case '1' : output_high(pin_c2);
                 break;
      case '2' : output_low(pin_c2);
                 break;
      case '9' : temperatura();
                 break;
   }
}
```

```
void main()
{
enable_interrupts(int_rda);

lcd_init(0x4E,16,2);
lcd_backlight_led(ON); //Enciende la luz de Fondo
lcd_clear();  //Limpia el LCD

   while(1)
   {
   }
}

void temperatura(){

       setup_adc_ports(AN0);
       setup_adc(adc_clock_div_64);
       set_adc_channel(0);          //Selecciono el canal 0 (RA0)
       delay_us(50);                 //llamo retardo de 50 ms
       temp=(read_adc())/2;         //Convertimos la lectura del adc a temperatura
       lcd_gotoxy(1, 2);
       printf(lcd_putc,"%0.2f °C",temp);
       delay_ms(1000);

}
```

Figura 14. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería `<18F4550.h>`, que es específica para el microcontrolador PIC18F4550. Luego, configura varios fusibles del microcontrolador que determinan aspectos importantes de su funcionamiento, como el oscilador, la protección del código, el divisor de reloj y la comunicación serial.

2. Configuración del Reloj y Osciladores:

Se configura el oscilador interno (`INTRC_IO`) y el oscilador externo de alta velocidad (`HS`) según los fusibles. También se configura el divisor de reloj a 1 (`CPUDIV1`) y se establece el reloj externo a 20 MHz (`#use delay(clock=20000000)`).

3. Declaración de Puertos:

Se declaran los puertos A, B, C, D y E utilizando las directivas `#byte`. Esto permite acceder y manipular los registros de estos puertos de manera más conveniente.

4. Configuración de Comunicación Serial RS232:

Se configura la comunicación serial RS232 con una velocidad de transmisión de 9600 baudios, sin paridad (`parity=N`) y 8 bits de datos. Se especifican los pines de transmisión (`xmit=pin_c6`) y recepción (`rcv=pin_c7`) para la comunicación.

5. Configuración de Comunicación I2C:

Se configura la comunicación I2C en modo maestro con una velocidad de 100,000 Hz. Los pines B0 y B1 se utilizan como SDA y SCL, respectivamente.

6. Inclusión de Librería para el LCD I2C:

Se incluye la librería `i2c_Flex_LCD.c`, que se utiliza para controlar un LCD mediante la comunicación I2C.

7. Declaración de Variables y Funciones:

Se declaran variables como `dato` y `temp`, y se define una función llamada `temperatura` para medir y mostrar la temperatura.

8. Interrupción Serial:

Se habilita una interrupción serial (`#int_rda`) para recibir comandos a través de la comunicación serial. Cuando se recibe un carácter, se ejecuta la función `rda_isr`, que actúa según el comando recibido (por ejemplo, encender o apagar un LED).

9. Función Principal:

En la función `main`, se inicializa un LCD de 16x2 caracteres y se enciende la luz de fondo. Luego, el programa entra en un bucle infinito.

10. Función de Medición de Temperatura:

La función `temperatura` configura el ADC para medir la temperatura en el pin AN0 (RA0). Luego, convierte la lectura del ADC a temperatura y muestra el resultado en el LCD. La temperatura se actualiza cada segundo.

Paso N.º 4: Compilar: Según Practica # 0

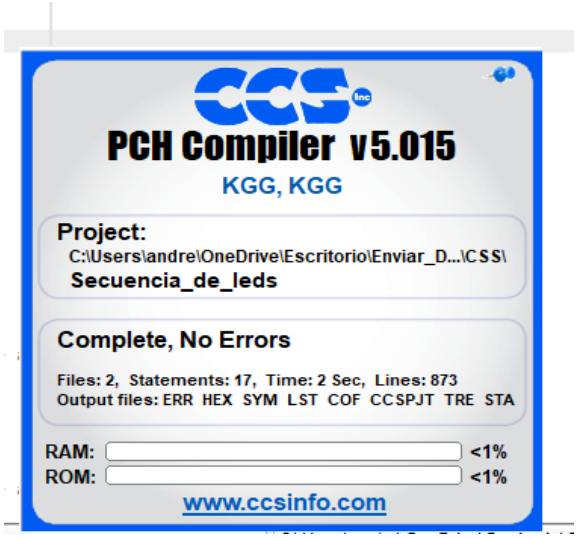


Figura 15. *Compilación Realizada.*

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 16. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

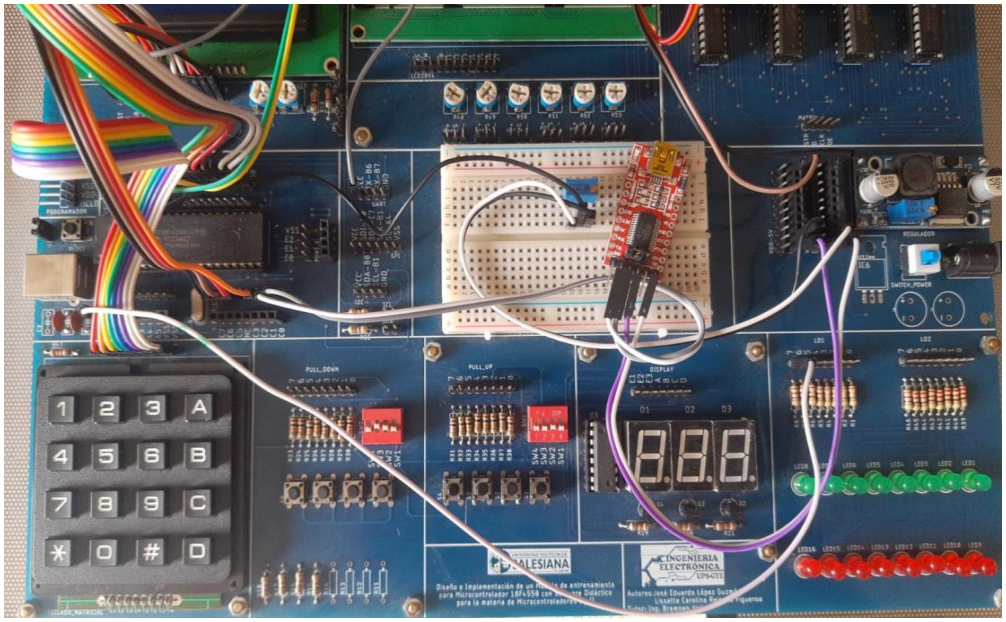


Figura 17. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 4**
NOMBRE PRÁCTICA:	Uso de Timmers para generar retardos de encendido de leds.	

1. **Objetivo General.**

   Implementar un programa para reforzar los conocimientos acerca de los Timmers.

2. **Objetivos Específicos.**

   Implementar un programa con el Timmer 1 del PIC que genera un retardo de encendido de 8 segundos.

3. **Marco Teórico.**

   Retardos de Encendido:

   Los retardos de encendido son intervalos de tiempo durante los cuales se espera antes de que ocurra un evento, como encender un dispositivo o realizar una acción específica. En este código, se utiliza un retardador de encendido para controlar el parpadeo de un LED. El LED se enciende y apaga en intervalos regulares determinados por el período de muestreo (`Ts_cont`).

   Interrupción del Timer 1:

   Se utiliza una interrupción del Timer 1 para controlar el parpadeo del LED. El Timer 1 es un contador de tiempo que genera interrupciones a intervalos regulares. Cuando se produce una interrupción, se ejecuta la función `sampling_time`, que invierte el estado del LED conectado al pin C2. Esto crea un efecto de parpadeo visible.

   Configuración del Timer 1:

   El Timer 1 se configura para utilizar una fuente de reloj interna y una división por 8. Esto determina la velocidad a la que se generan las interrupciones. El período de muestreo se ajusta mediante la carga inicial del Timer 1 y la cantidad de decrementos en `Ts_cont`. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

El programa hace parpadear un LED conectado al pin C2 del microcontrolador a intervalos regulares.

El intervalo de parpadeo está determinado por el valor de `Ts_cont` y la configuración del Timer 1. En este caso, el LED cambia de estado aproximadamente cada 8 segundos.

5. Materiales.

- Módulo didáctico.
- Jumpers conectores.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

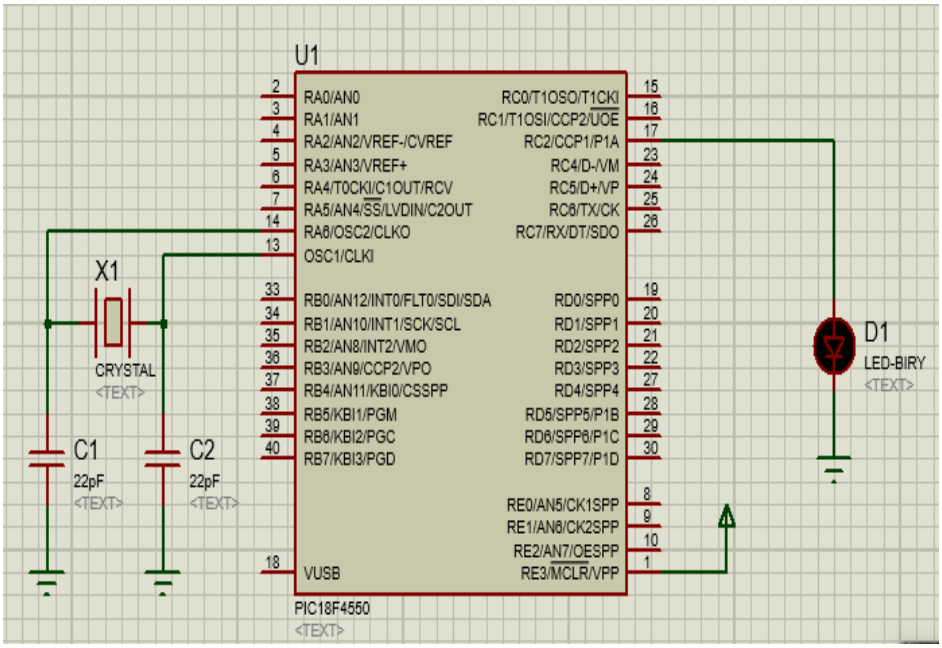


Figura 1. Simulación en Proteus.

Pines del PIC	Pines del LED
C2	R1-LED1

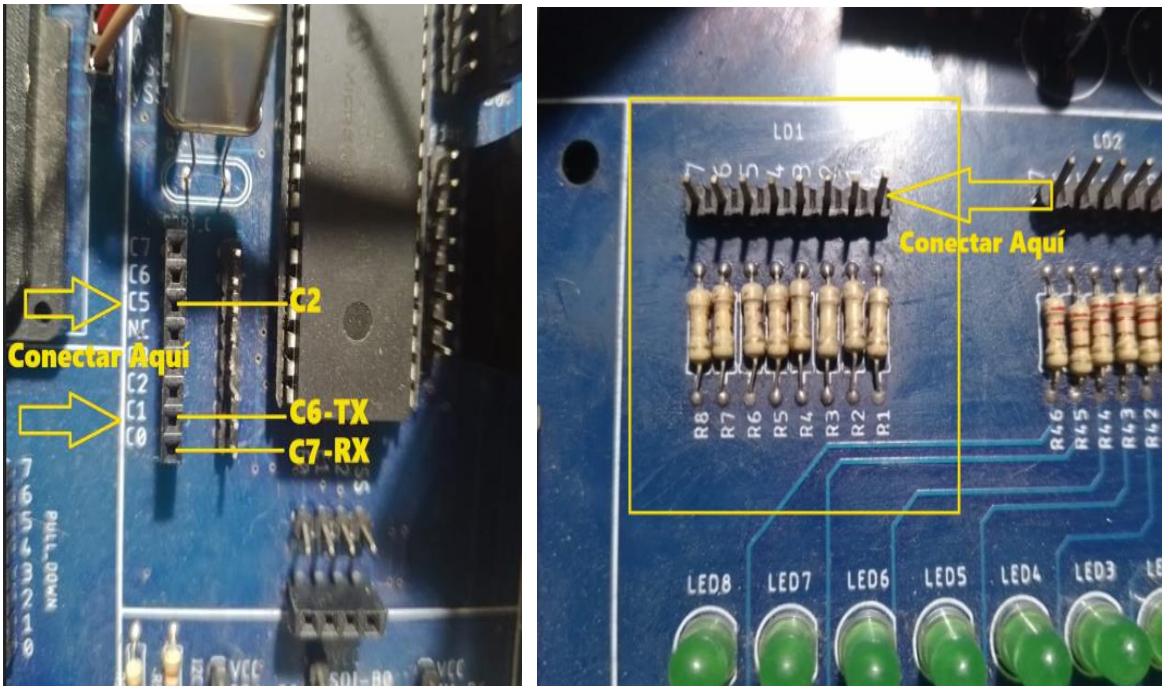


Figura 2. Conexión del PIC al LED.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

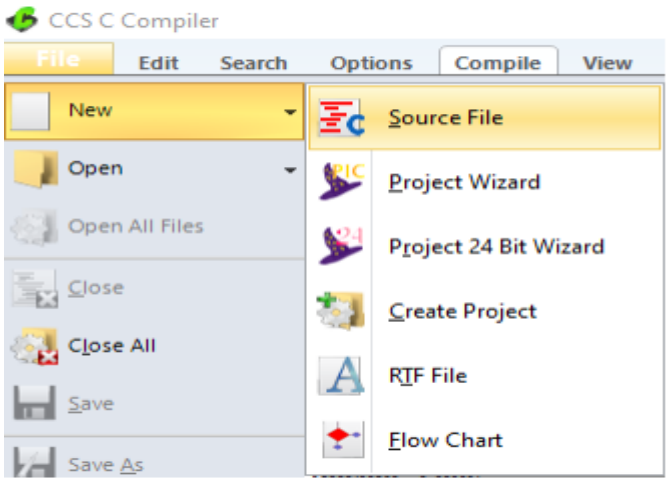


Figura 3. Crear un nuevo archivo.

Paso N.º 3: Realizar la programación de acuerdo con la siguiente figura

```
#include <18f4550.h>// Declaramos al Pic 18F4550
#DEVICE ADC=10//seteamos el adc de 10 bits
#USE DELAY(crystal=20000000)//reloj externo 20 Mhz
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer

#byte porta = 0xf80 // Identificador para el puerto A.
#byte portb = 0xf81 // Identificador para el puerto B.
#byte portc = 0xf82 // Identificador para el puerto C.
#byte portd = 0xf83 // Identificador para el puerto D.
#byte porte = 0xf84 // Identificador para el puerto E.
int16 Ts_cont=0;
// ******************************** //
// ******* Interrupción Timer 1 **** //
// ******************************** //
```

```
#int_timer1  //TIMER1
void sampling_time(void){
   Ts_cont--;           //Se decrementa hasta llegar a cero
   set_timer1(15536);//Carga de nuevo el timer1
   if (Ts_cont<=0) // Si llega a cero, se cumplió el periodo de muestreo
   {
      output_toggle(pin_c6);
      Ts_cont=100;  // Inicializa el contador para el próximo periodo
   }
}
// ******************************** //
// ****** Programa Principal ******* //
// ******************************** //
void main() {
   set_tris_d(0b0); //Configura el pin D0 como salida (LED)
   // Configura el Timer 1
   setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8 );
   Ts_cont=100;         //Carga Contrador para Calcular el Periodo de Muestreo
   set_timer1(15536); // Inicializa el Timer 1 para calcular 8 Segundos de Ts

   // Habilitar las interrupciones del PIC
   enable_interrupts(int_timer1);
   enable_interrupts(GLOBAL);
   while(1){
   }
}
```

Figura 4. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería `<18f4550.h>`, que declara al microcontrolador PIC 18F4550. Luego, se configuran los fusibles del microcontrolador que determinan aspectos importantes de su funcionamiento, como el oscilador, la protección del código, el Power-Up Timer y otros.

2. Configuración del Reloj y Oscilador:

Se configura el reloj externo a 20 MHz (`#USE DELAY(crystal=20000000)`), lo que establece la velocidad de funcionamiento del microcontrolador. También se utiliza el fusible `HS` para indicar que se utiliza un oscilador externo de alta velocidad.

3. Declaración de Puertos:

Se declaran los puertos A, B, C, D y E utilizando las directivas `#byte`. Esto permite acceder y manipular los registros de estos puertos de manera más conveniente.

4. Interrupción del Timer 1:

Se define una interrupción (`#int_timer1`) para el Timer 1. Esta interrupción se ejecuta periódicamente y se utiliza para realizar una acción específica, en este caso, el parpadeo de un LED.

5. Función de Interrupción `sampling_time`:

La función de interrupción `sampling_time` se encarga de controlar el parpadeo de un LED en el pin C2. Este parpadeo ocurre a intervalos regulares y se configura mediante el Timer 1. El LED se invierte cada vez que se produce esta interrupción.

6. Configuración del Timer 1:

Se configura el Timer 1 para que funcione con una fuente de reloj interna y una división por 8 (`T1_INTERNAL | T1_DIV_BY_8`). Esto establece el período y la velocidad de la interrupción.

7. Programa Principal:

En la función `main`, se configura el pin C2 como salida para conectar un LED. Luego, se inicializa el Timer 1 y se inicia el ciclo principal del programa.

8. Configuración de Período de Muestreo:

Se establece un período de muestreo (`Ts_cont`) que determina cada cuánto se ejecutará la interrupción del Timer 1. En este caso, se inicializa en 100 y se decrementa en cada ciclo de interrupción hasta llegar a cero, momento en el cual se invierte el estado del LED.

9. Habilitación de Interrupciones:

Se habilitan las interrupciones del Timer 1 (`int_timer1`) y las interrupciones globales (`GLOBAL`). Esto permite que el microcontrolador responda a la interrupción del Timer 1 y ejecute la función `sampling_time`.

Paso N.º 4: Compilar: **Según Practica # 0**

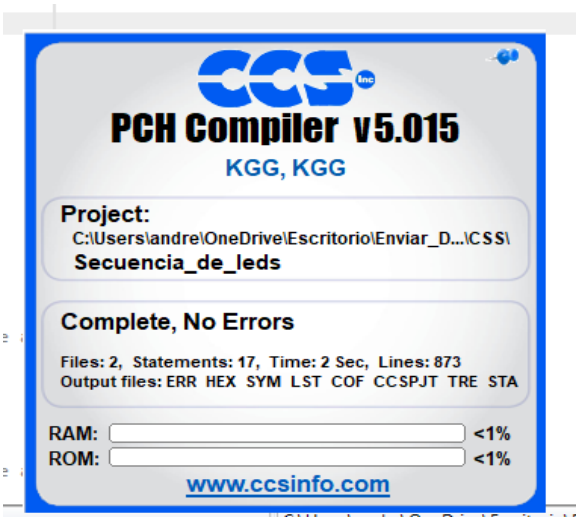


Figura 5. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 6. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

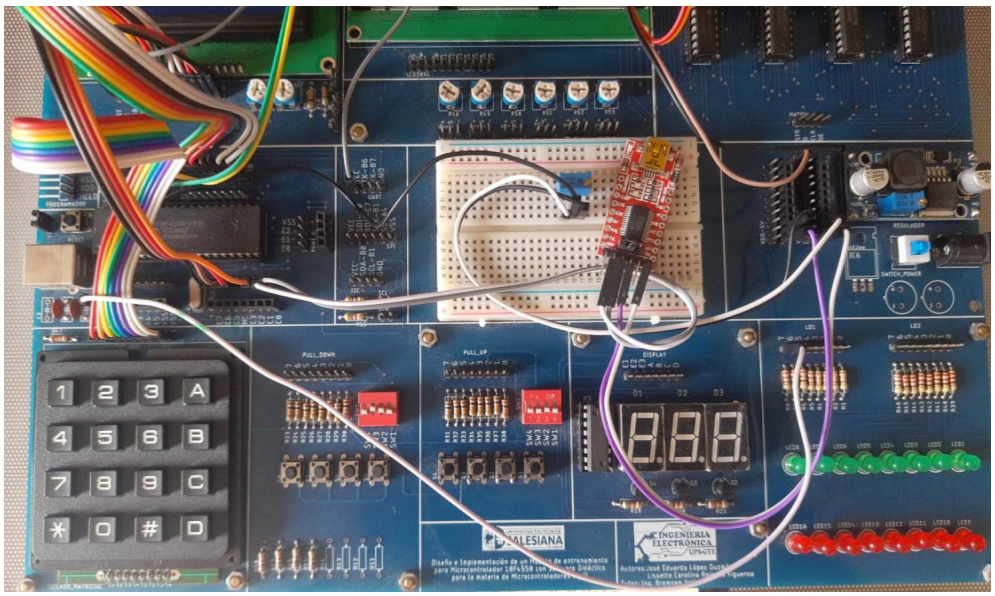


Figura 7. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 5**
NOMBRE PRÁCTICA:	Lectura de temperatura con una termocupla usando comunicación SPI para la optimización de pines.	

1. Objetivo General.

Implementar un programa que realice la lectura de temperatura.

2. Objetivos Específicos.

1. Implementar un programa que utilice un módulo SPI para termocuplas.

3. Marco Teórico.

Comunicación SPI (Serial Peripheral Interface):

La comunicación SPI es un protocolo de comunicación serie ampliamente utilizado en electrónica. Se utiliza para intercambiar datos entre un microcontrolador y dispositivos periféricos, como sensores o módulos de comunicación.

En este código, la comunicación SPI se utiliza para comunicarse con el sensor de temperatura MAX6675. El MAX6675 es un convertidor de termocupla a digital que convierte la temperatura medida por la termocupla en una señal digital que el microcontrolador puede leer.

Termocuplas:

Las termocuplas son sensores de temperatura que se basan en el principio de que la temperatura afecta la tensión eléctrica generada por dos metales diferentes unidos en un extremo. La diferencia de temperatura entre el extremo caliente y el extremo frío de la termocupla genera una pequeña señal de voltaje que se puede medir y convertir en una lectura de temperatura.

En este código, se utiliza una termocupla tipo K conectada al MAX6675 para medir la temperatura. El MAX6675 convierte la señal de voltaje de la termocupla en una lectura digital que se utiliza para mostrar la temperatura en el LCD`. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

El programa realiza mediciones de temperatura utilizando una termocupla tipo K y muestra la temperatura medida en un LCD.

La temperatura se muestra en grados Celsius (°C) con dígitos separados en centenas, decenas y unidades.

El bucle principal se ejecuta continuamente, lo que significa que la temperatura se mide y muestra constantemente en el LCD.

Esto proporciona un monitoreo en tiempo real de la temperatura ambiente o de un proceso específico en el entorno físico donde se implementa el sistema.

5. Materiales.

- Módulo didáctico.
- Jumpers conectores.
- Termocupla tipo K.
- Módulo Max6675.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

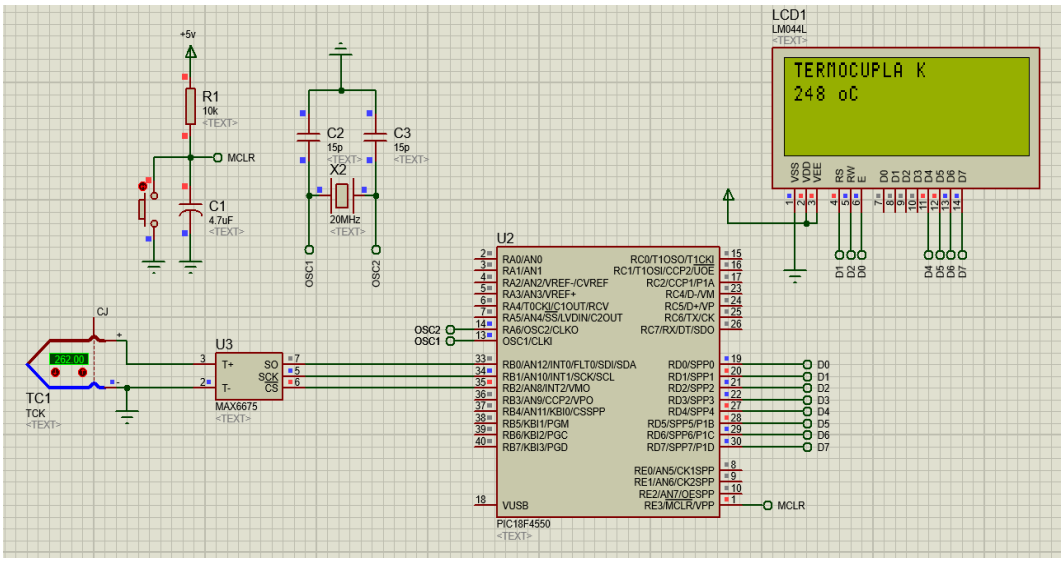


Figura 1. Simulación en Proteus.

Pines del PIC	Pines de la Pantalla/Max6675
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
B0	SO-MAX6675
B1	SCK-MAX6675
B2	CS-MAX6675
Pines del Max6675	
VCC-MAX6675	VDD-5V
GND-MAX6675	VSS-0V

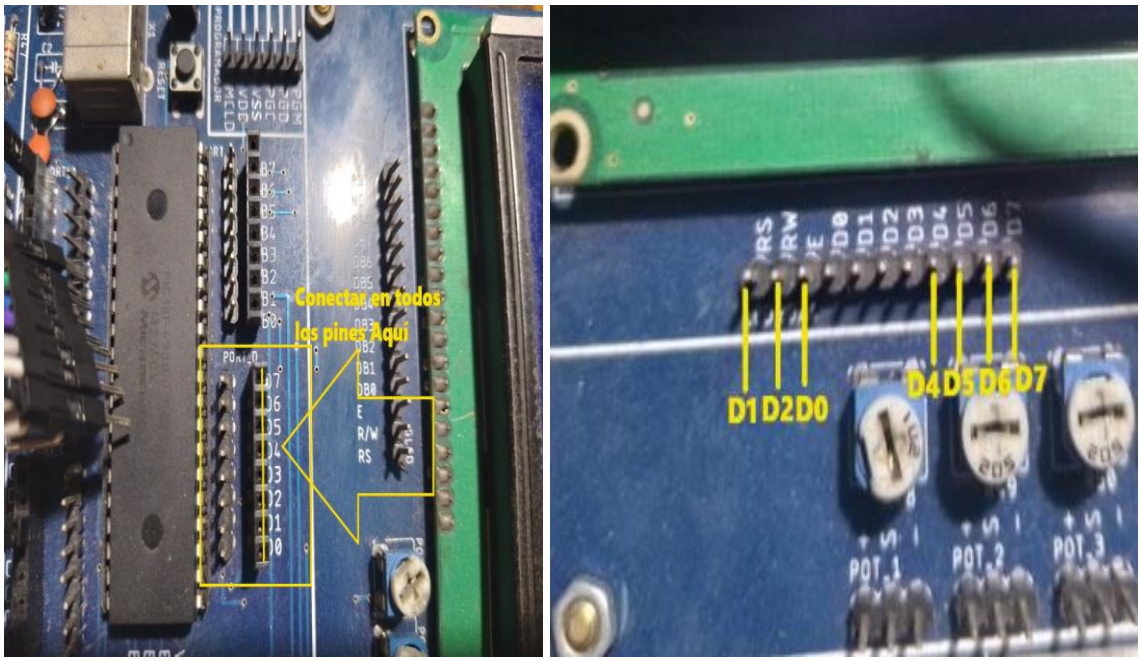


Figura 2. Conexiones del Puerto D a la pantalla LCD.

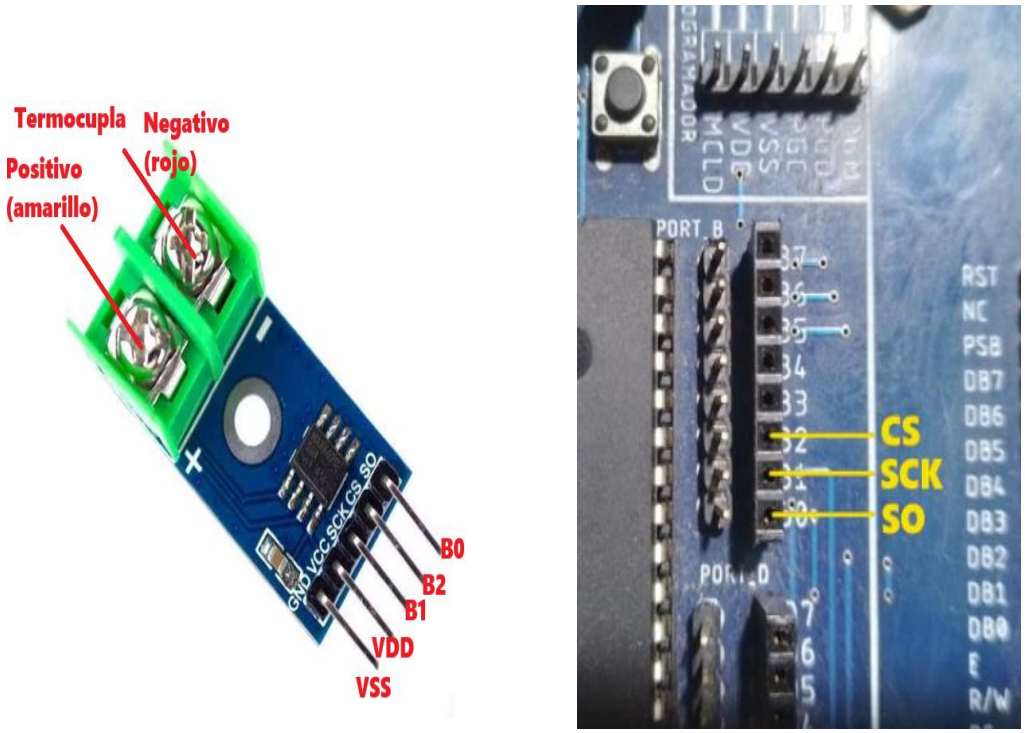


Figura 3. Conexiones del Puerto B al Módulo Max6675.

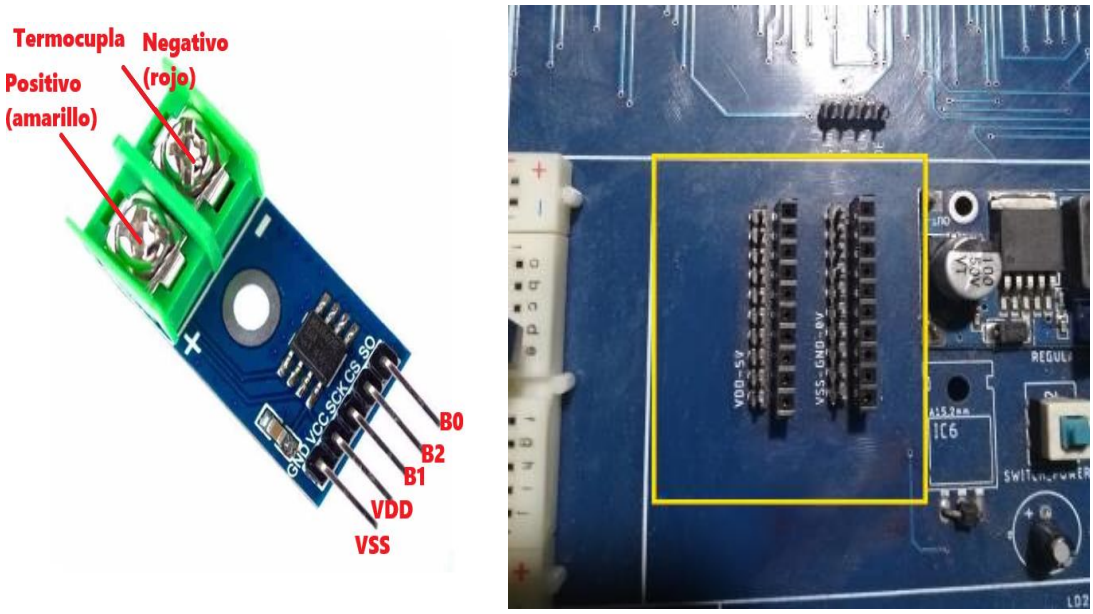


Figura 4. Conexiones del Max6675 a la alimentación.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

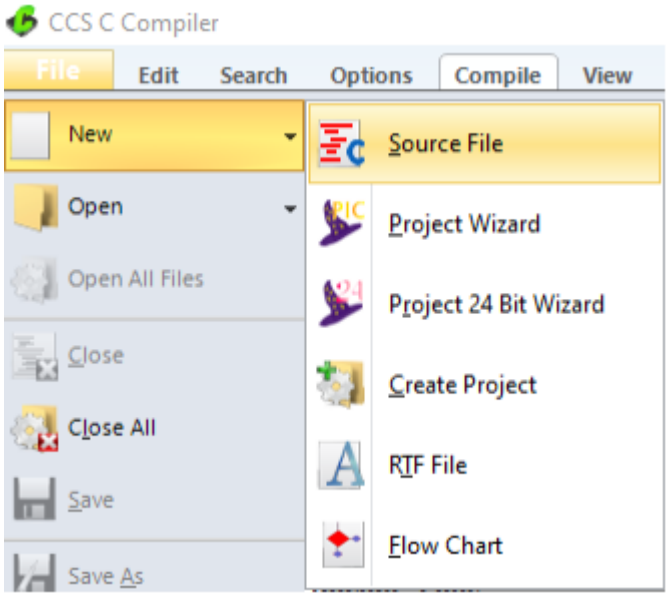


Figura 5. Crear un nuevo proyecto en el ide de CCS C Compiler.

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18f4550.h>          // Incluimos el fichero del Microcontrolador PIC18F4550
#fuses HS,NOWDT,NOPROTECT     // Definimos las palabras de configuración del PIC
                              // HS(cristal 20MHz), No Watch Dog Timer, Sin proteccion de memoria de programa
#use delay(clock=20000000)    // Definimos la Frecuencia del oscilador de PIC(20MHz)

#include "MAX6675_Lib.c"      // Incluimos la librería MAX6675
#include <lcd.c> // incluimos la libreria del LCD

//Declaramos los pines a usar por la pantalla LCD
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////////////////////////////////////////////

int16 Tm;                             // Definimos las variable de temperatura del sensor RTD PT100 como flotante

int16 Unidad, Decena, Centena, Centena1; // Definimos las variables como enteros de 16-Bit

void main() {                         // Inicia la función principal

   set_tris_d(0x00);                  // Seteamos el puerto D como salida

   lcd_init();

   do{                                // Inicia al bucle do ... While infinito

      Tm = read_termocupla_k();              // Lectura de la termocupla tipo K
    Tm = Tm - 35;

    Centena = Tm/100;                       // Captura el dígito de las centenas

    Centena1 = Tm - Centena*100;            // Captura el resto de la división anterior

    Decena = Centena1/10;                   // Captura el dígito de las decenas

    Unidad = Centena1 - Decena*10;          // Captura el dígito de las unidades

   lcd_gotoxy(1,1);

   LCD_PUTC("TERMOCUPLA K");
    lcd_gotoxy(1,2);
    printf(lcd_putc,"%ld",Centena); // Visualiza el dígito de las centenas en la pantalla, posición (1,2)
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%ld",Decena);          // Visualiza el dígito de las decenas en la pantalla, posición (2,2)
    lcd_gotoxy(3,2);
    printf(lcd_putc,"%ld",Unidad);          // Visualiza el dígito de las unidades en la pantalla, posición (3,2)

   lcd_gotoxy(5,2);

   LCD_PUTC("o");             // Imprime texto en la pantalla

   lcd_gotoxy(6,2);

   LCD_PUTC("C");             // Imprime texto en la pantalla /

   delay_ms(100);                          // Tiempo de espera de 100ms

  }while(1);                               // Termina el bucle do ... While infinito
```

Figura 6. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería `<18f4550.h>`, que declara al microcontrolador PIC18F4550. Luego, se configuran los fusibles del microcontrolador para especificar la velocidad del oscilador, la protección del código y otros aspectos de la configuración.

2. Configuración del Oscilador y Reloj:

Se configura el oscilador para utilizar un cristal de alta velocidad de 20 MHz (`HS`) y se desactiva el Watch Dog Timer (`NOWDT`). Además, se utiliza la función `#use delay` para especificar la frecuencia del reloj del microcontrolador, que es 20 MHz en este caso.

3. Inclusión de Librerías Externas:

Se incluye la librería `MAX6675_Lib.c`, que contiene funciones y configuraciones relacionadas con el sensor de temperatura MAX6675. También se incluye la librería `lcd.c` para el control del LCD.

4. Configuración de Pines para el LCD:

Se definen los pines a utilizar para la comunicación con el LCD, como el pin de habilitación (`LCD_ENABLE_PIN`), el pin RS (`LCD_RS_PIN`), el pin RW (`LCD_RW_PIN`) y los pines de datos (`LCD_DATA4` a `LCD_DATA7`).

5. Declaración de Variables:

Se declaran varias variables, incluyendo `Tm` para la temperatura medida, así como variables para descomponer y mostrar los dígitos de la temperatura en la pantalla LCD.

6. Configuración del Puerto D:

Se configura el puerto D como salida (`set_tris_d(0x00)`) para controlar un componente externo, posiblemente un LED u otro dispositivo.

7. Bucle Principal:

Entra en un bucle infinito (`do ... while(1)`) que realizará continuamente las siguientes acciones:

8. Lectura de la Termocupla:

Se realiza una lectura de la termocupla tipo K utilizando la función `read_termocupla_k()`. Esta función está definida en la librería `MAX6675_Lib.c`. La temperatura medida se almacena en la variable `Tm`.

9. Descomposición de la Temperatura:

La temperatura medida (`Tm`) se descompone en sus dígitos de centenas, decenas y unidades.

10. Visualización en el LCD:

Se muestra la temperatura medida en el LCD. Los dígitos de centenas, decenas y unidades se muestran en posiciones específicas en la pantalla LCD. Además, se muestra "°C" como texto.

11. Espera y Repetición:

Después de mostrar la temperatura, el programa espera durante 100 milisegundos y luego repite el proceso para medir y mostrar la temperatura nuevamente.

Paso N.º 4: Compilar: **Según Practica # 0**

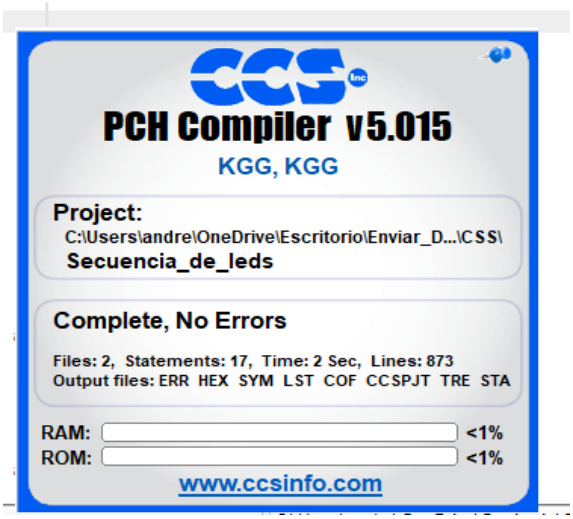


Figura 7. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

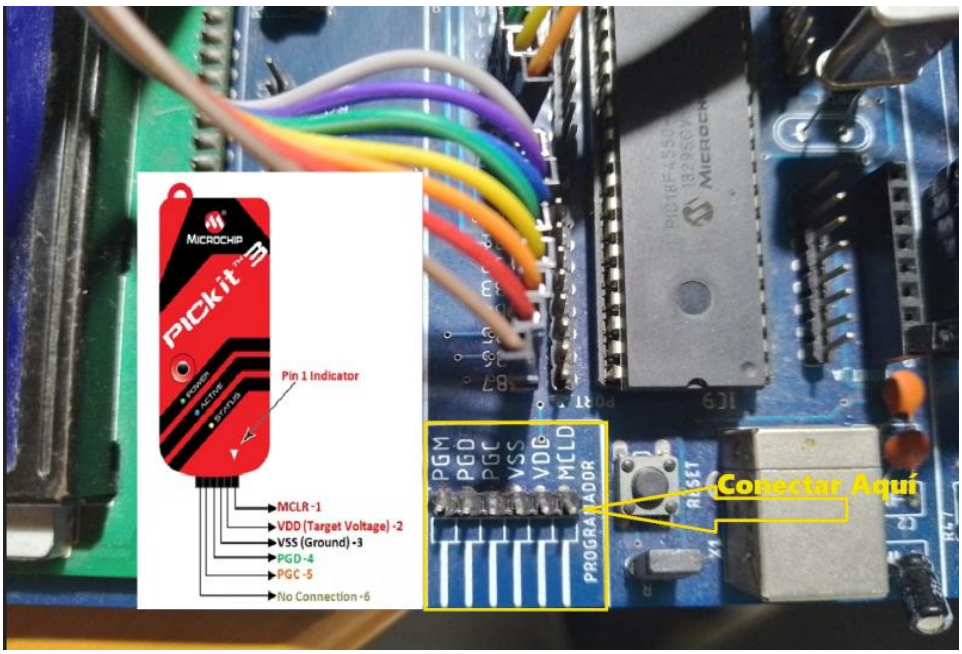


Figura 8. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

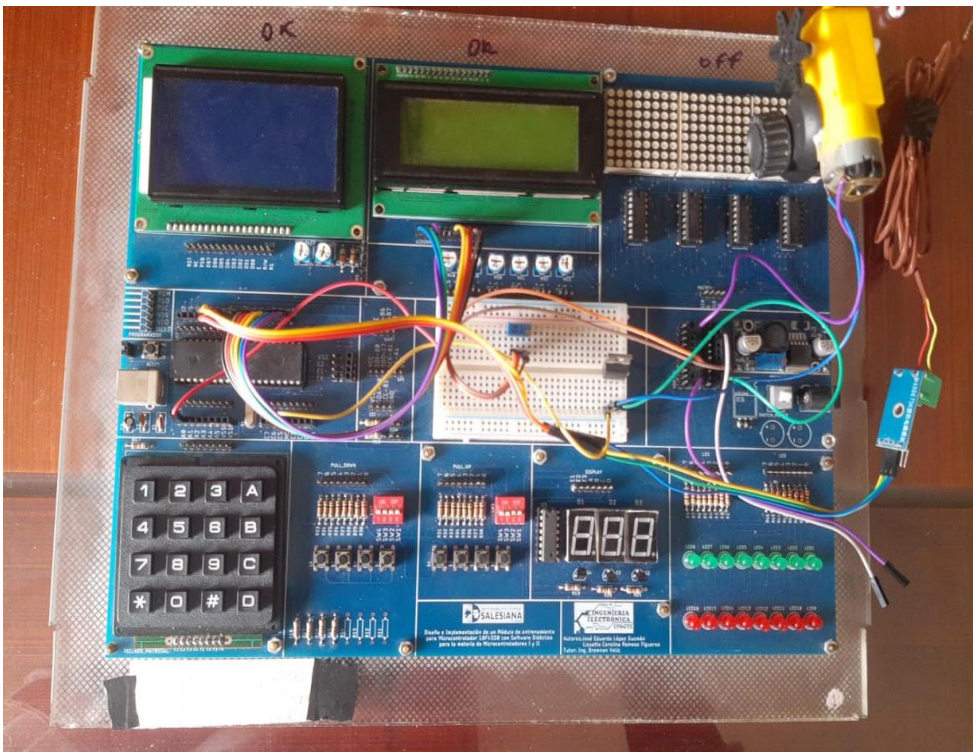


Figura 9. *Prueba de funcionamiento.*

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	**MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS**	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 6**
NOMBRE PRÁCTICA:	Conexión del LCD usando comunicación I2C para la optimización de pines.	

1. Objetivo General.

Implementar un programa para usar comunicación I2C (inter integrated circuits).

2. Objetivos Específicos.

1. Implementar un programa para conectar una pantalla LCD por medio de comunicación I2C.

3. Marco Teórico.

Desplazamiento de Mensajes en LCD:

El desplazamiento de mensajes en un LCD se utiliza para mostrar mensajes largos en una pantalla LCD que tiene un número limitado de caracteres visibles a la vez. En lugar de cortar el mensaje o hacerlo ilegible, se puede implementar un desplazamiento horizontal para mostrar el mensaje completo en partes.

En este código, se utiliza un bucle `for` para mostrar segmentos del mensaje en el LCD, avanzando de derecha a izquierda a intervalos regulares. Esto permite que los mensajes `mensaje` y `mensaje2` se desplacen horizontalmente en el LCD, lo que hace que la información sea legible y completa.

Uso de LCD I2C:

Se utiliza un LCD I2C que está controlado mediante el protocolo de comunicación I2C. Esto simplifica la conexión y el control del LCD, ya que se pueden utilizar solo dos pines (SDA y SCL) para la comunicación en lugar de una conexión de varios cables. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

El programa muestra un mensaje de bienvenida (`"Hola, Dennys León!"`) Y otro mensaje (`"Universidad Salesiana"`) en el LCD.

Los mensajes se desplazan horizontalmente en el LCD, lo que da la impresión de que el texto se mueve de derecha a izquierda y viceversa.

Este tipo de efecto de desplazamiento de texto puede ser utilizado en aplicaciones como carteles electrónicos, tableros de anuncios o cualquier otra aplicación donde se requiera mostrar mensajes en movimiento.

5. Materiales.

- Módulo didáctico.
- Jumpers conectores.
- Display LCD I2C.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

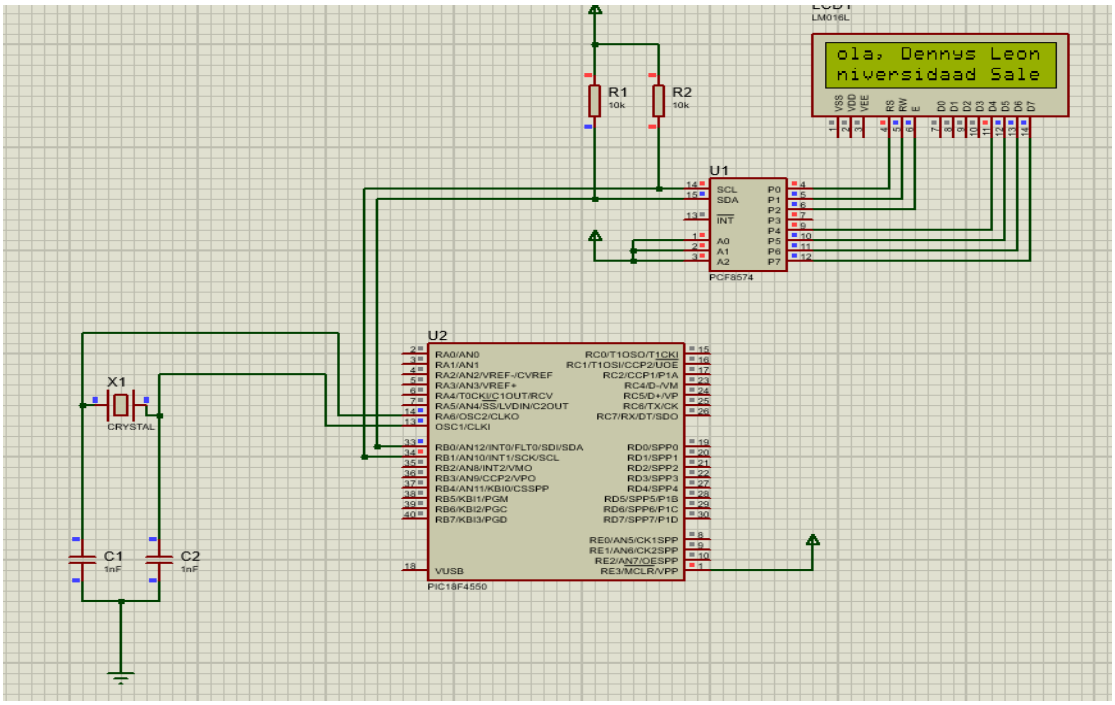


Figura 1. Simulación en Proteus.

Pines del PIC	Pines del Módulo USB/LCD/POT
B0	SDA-LCD I2C
B1	SCL-LCD I2C
Pines del LCD I2C	
VCC-LCD	VDD-5V
GND-LCD	VSS-0V

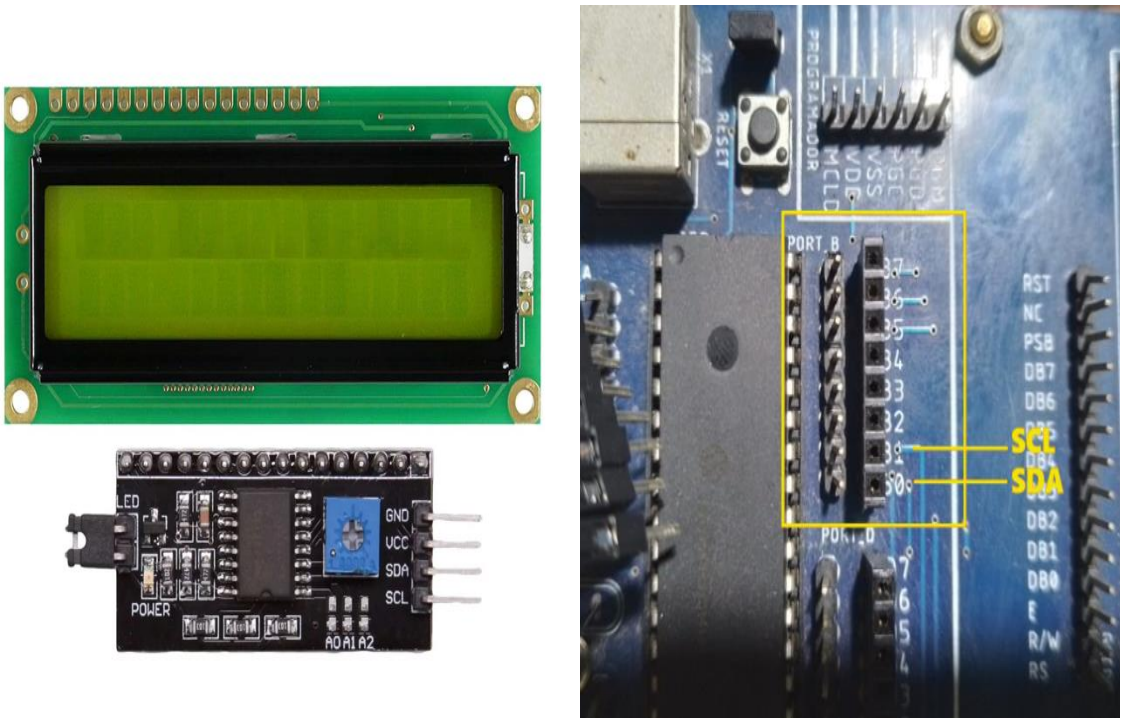


Figura 2. Conexión de la pantalla LCD I2C al PIC.

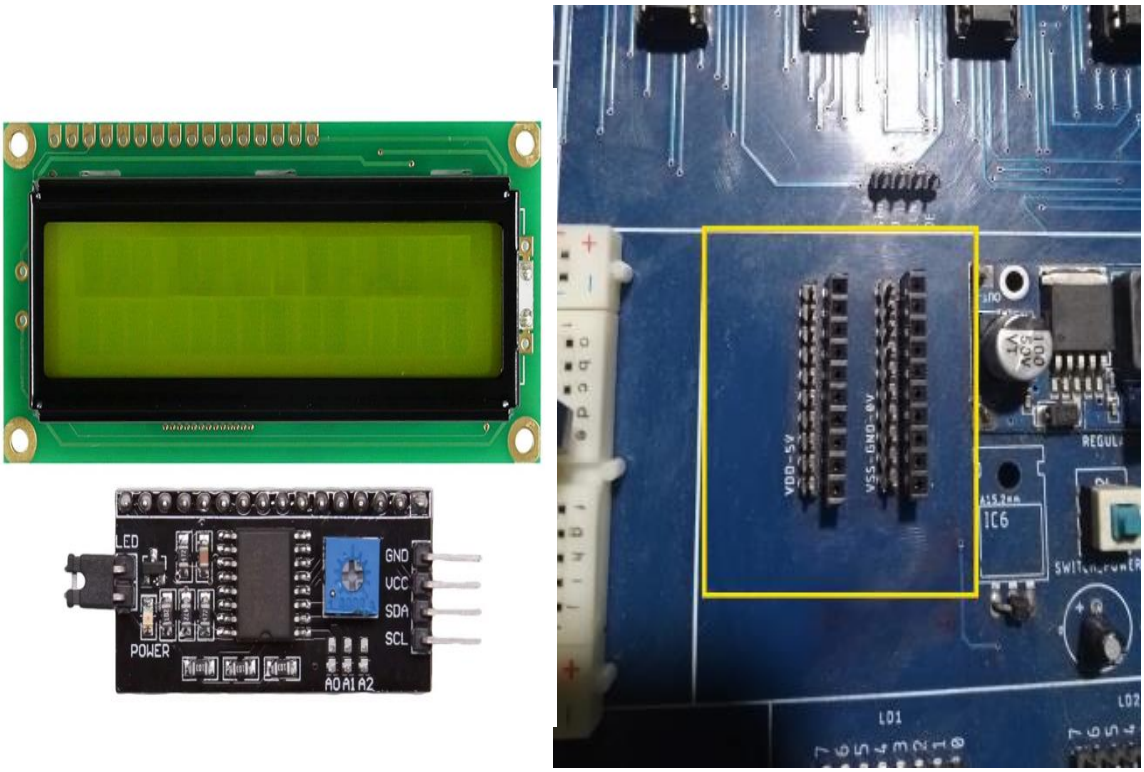


Figura 3. Conexión de la pantalla LCD I2C a la alimentación.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

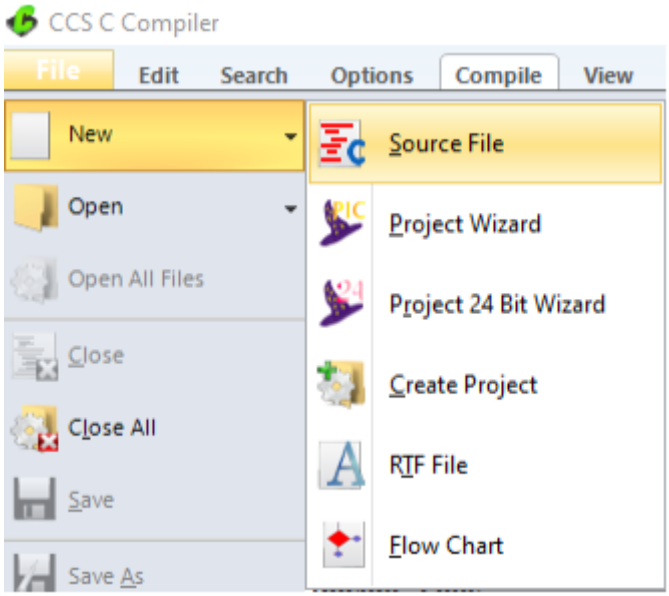


Figura 4. Crear un nuevo proyecto en el ide de CCS C Compiler

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18f4550.h>// Declaramos al Pic 18F4550
#USE DELAY(crystal=200000000)//Seteamos el reloj externo a 20MHz
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer

#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD
#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C
#define LCD_COLS 16

void main() {
lcd_init(0x4E,16,2);
lcd_backlight_led(ON); //Enciende la luz de Fondo
lcd_clear();  //Limpia el LCD

    char mensaje[] = "Hola, Dennys Leon !";
    int mensaje_len = 21; // tamaño del mensaje
    char mensaje2[] = "Universidad Salesiana";
    int end_index = mensaje_len - LCD_COLS;
  while(true) {
   for (int i = 0; i <= end_index; i++)
          {
              lcd_gotoxy(1, 1);
              for (int j = i; j < i + LCD_COLS; j++)
              {
                  lcd_putc(mensaje[j]);
              }
              delay_ms(100);

               lcd_gotoxy(1, 2);
              for (int k = i; k < i + LCD_COLS; k++)
              {
                  lcd_putc(mensaje2[k]);
              }
              delay_ms(100);
          }
  }
  }
```

Figura 5. Programación en CCS C Compiler.

Descripción Paso a Paso del Código:

1. Inclusión de Librerías y Configuración de Fusibles:

El código comienza incluyendo la librería `<18f4550.h>`, que declara al microcontrolador PIC18F4550. Luego, se configuran los fusibles del microcontrolador para especificar la velocidad del oscilador, la protección del código y otros aspectos de la configuración.

2. Configuración del Oscilador y Reloj:

Se configura el oscilador para utilizar un cristal de alta velocidad de 20 MHz (`HS`) y se desactiva el Watch Dog Timer (`NOWDT`). Además, se utiliza la función `#USE DELAY` para especificar la frecuencia del reloj del microcontrolador, que es 20 MHz en este caso.

3. Inclusión de Librerías Externas:

Se incluye la librería `i2c_Flex_LCD.c`, que contiene funciones y configuraciones relacionadas con la comunicación I2C para el LCD. También se configura la comunicación I2C para ser el maestro (`Master`) con una velocidad de 100,000 bps (`Fast`).

4. Inicialización del LCD:

Se inicializa el LCD con `lcd_init()`, se enciende la luz de fondo del LCD con `lcd_backlight_led(ON)` y se limpia el LCD con `lcd_clear()`.

5. Definición de Mensajes:

Se definen dos mensajes de texto (`mensaje` y `mensaje2`) que se mostrarán en el LCD. Cada mensaje tiene una longitud específica (`mensaje_len`).

6. Bucle Principal:

El bucle principal (`while(true)`) se ejecuta continuamente y realiza lo siguiente:

7. Desplazamiento de Texto en el LCD:

Se realiza un desplazamiento de texto en el LCD para mostrar los mensajes completos. El bucle `for` itera a través de los caracteres de los mensajes y muestra un segmento de texto en el LCD durante un breve período de tiempo (`delay_ms(100)`).

Esto permite que los mensajes se desplacen horizontalmente en el LCD, dando una apariencia de texto en movimiento.

8. Fin del Mensaje:

Cuando se ha mostrado todo el mensaje, el bucle se repite y continúa mostrando el mensaje desde el principio.

Paso N.º 4: Compilar: **Según Practica # 0**

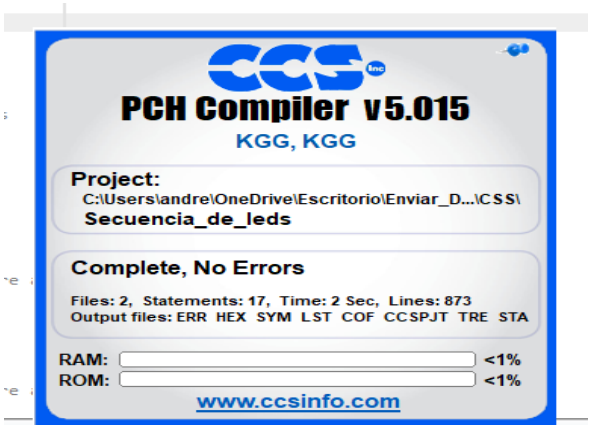


Figura 6. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 7. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

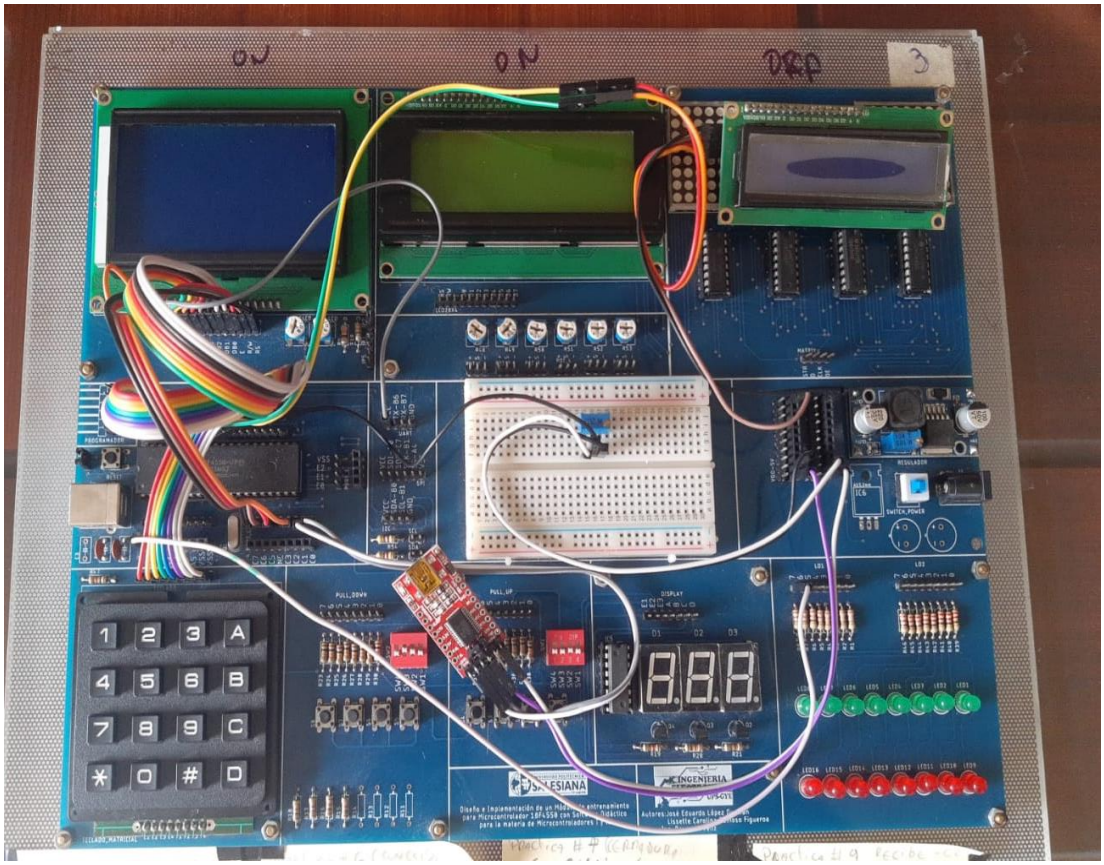


Figura 8. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 7**
NOMBRE PRÁCTICA:	Control de acceso (cerradura electrónica) usando LCD y teclado matricial.	

1. **Objetivo General.**

   Implementar un programa de control de acceso.

2. **Objetivos Específicos.**

1. Implementar un programa que utilice la pantalla LCD y el teclado matricial para realizar una cerradura electrónica.

3. **Marco Teórico.**

   Teclado Matricial:

   Un teclado matricial es un dispositivo de entrada que permite al usuario ingresar datos numéricos o alfanuméricos.

   Consiste en una matriz de botones o teclas organizados en filas y columnas.

   En este código, se utiliza un teclado matricial 4x4, lo que significa que tiene 4 filas y 4 columnas.

   Cada tecla en el teclado matricial está conectada a una fila y una columna específica, y se presiona combinando una fila y una columna.

   Contraseña de Acceso:

   (DESCRIPCION DE CONTROL DE ACCESO)

   La contraseña de acceso es una secuencia de dígitos numéricos que el usuario debe ingresar correctamente para obtener acceso a un sistema o dispositivo.

   En el código, se almacena una contraseña de cuatro dígitos en la memoria EEPROM del microcontrolador PIC18F4550.

   EEPROM (Electrically Erasable Programmable Read-Only Memory):

   La EEPROM es una memoria no volátil que permite almacenar datos que no se pierden cuando se apaga la alimentación del microcontrolador.

   En este código, se utiliza la EEPROM para almacenar la contraseña de acceso de forma persistente.

Cerradura Electrónica:

Una cerradura electrónica es un mecanismo de cierre que se controla electrónicamente y puede bloquear o desbloquear una puerta o acceso de manera segura.

En este contexto, se menciona una "puerta abierta" como una representación del acceso concedido, pero en una implementación práctica, podría conectarse a una cerradura electrónica que permite el acceso cuando se ingresa la contraseña correcta.

Seguridad de Acceso:

El sistema de seguridad de acceso implementado en este código utiliza la combinación de un teclado matricial y una contraseña para controlar el acceso a un área o dispositivo.

Proporciona una capa adicional de seguridad al requerir que los usuarios ingresen una contraseña válida para obtener acceso. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

Al ejecutar este programa en el microcontrolador PIC18F4550 y conectar el teclado matricial 4x4, se mostrará un mensaje en el LCD pidiendo al usuario que ingrese una contraseña de cuatro dígitos.

El usuario ingresa los dígitos utilizando el teclado matricial, y los dígitos se muestran en la pantalla LCD a medida que se ingresan.

Una vez ingresada la contraseña completa, el programa la compara con la contraseña almacenada en la EEPROM.

Si la contraseña ingresada coincide con la almacenada, se muestra "puerta abierta" en el LCD y se activa una cerradura electrónica durante 1 segundo (representada con un led).

Si la contraseña ingresada no coincide, se muestra "acceso denegado" en el LCD durante 1 segundo.

El programa vuelve a estar listo para ingresar una nueva contraseña.

Este código se utiliza para implementar una forma básica de control de acceso utilizando un teclado matricial y una pantalla LCD, lo que proporciona una capa adicional de seguridad en aplicaciones como sistemas de alarma o acceso a áreas restringidas.

5. Materiales.

- Módulo didáctico.
- Display LCD I2C.
- Jumpers conectores.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

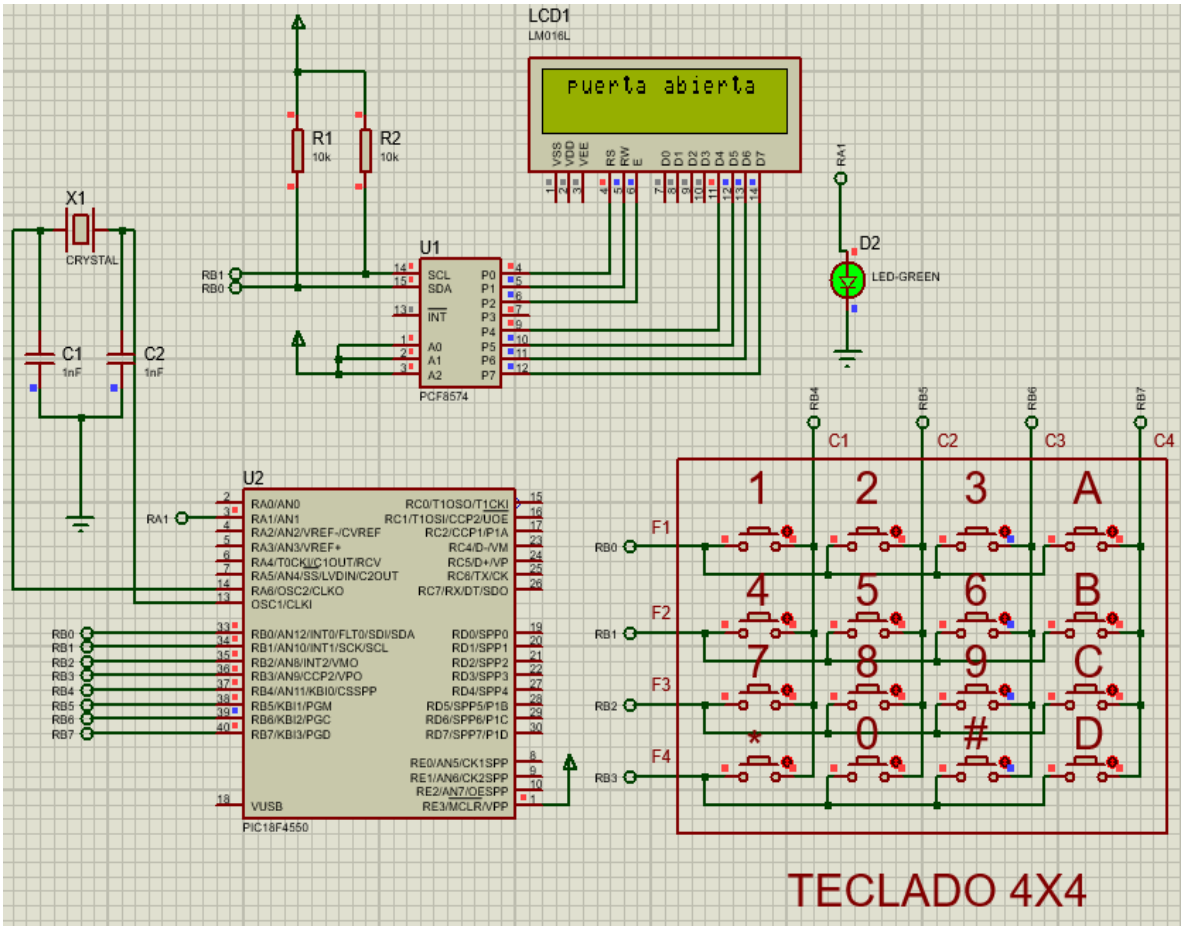


Figura 1. Simulación en Proteus

Pines del PIC	Pines del Teclado/LCD I2C/Led
B0	SDA-LCD I2C
B1	SCL-LCD I2C
B0	r1-TECLADO
B1	r2-TECLADO
B2	r3-TECLADO
B3	r4-TECLADO
B4	c1-TECLADO
B5	c2-TECLADO
B6	c3-TECLADO
B7	c4-TECLADO
A1	R1-LED1
Pines del LCD I2C	
VCC-LCD	VDD-5V
GND-LCD	VSS-0V

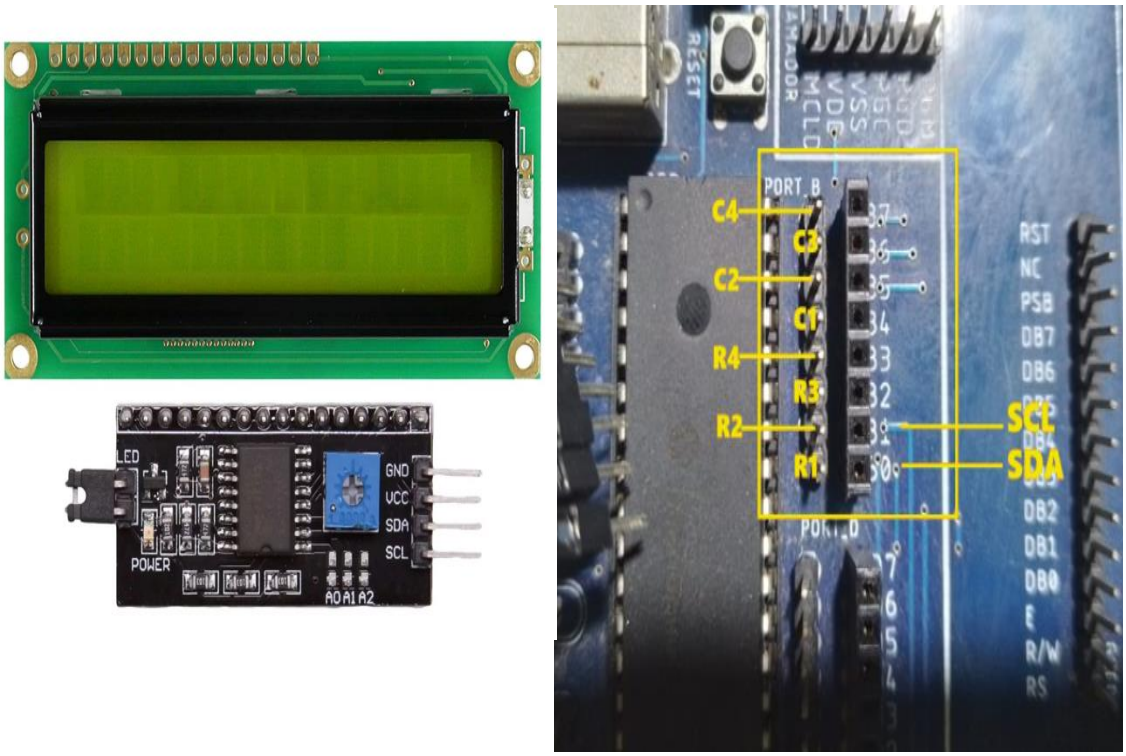


Figura 2. Conexión de la pantalla LCD I2C al PIC.

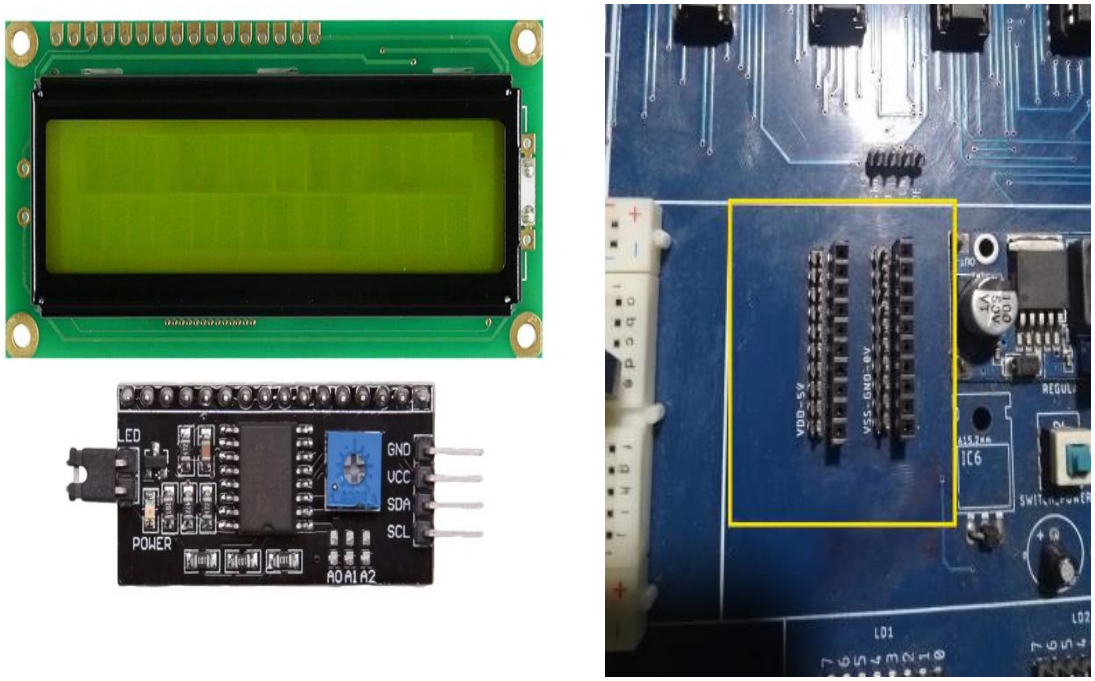


Figura 3. Conexión de la pantalla LCD I2C a la alimentación.

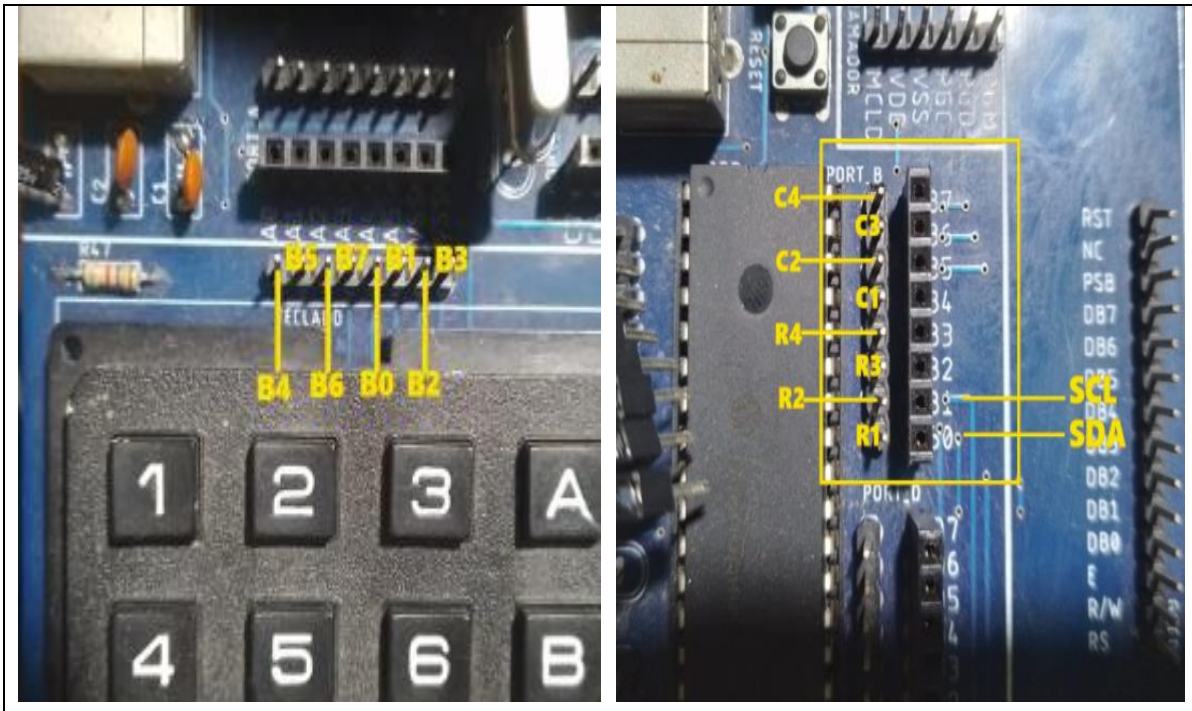


Figura 4. Conexión del Teclado al PIC.

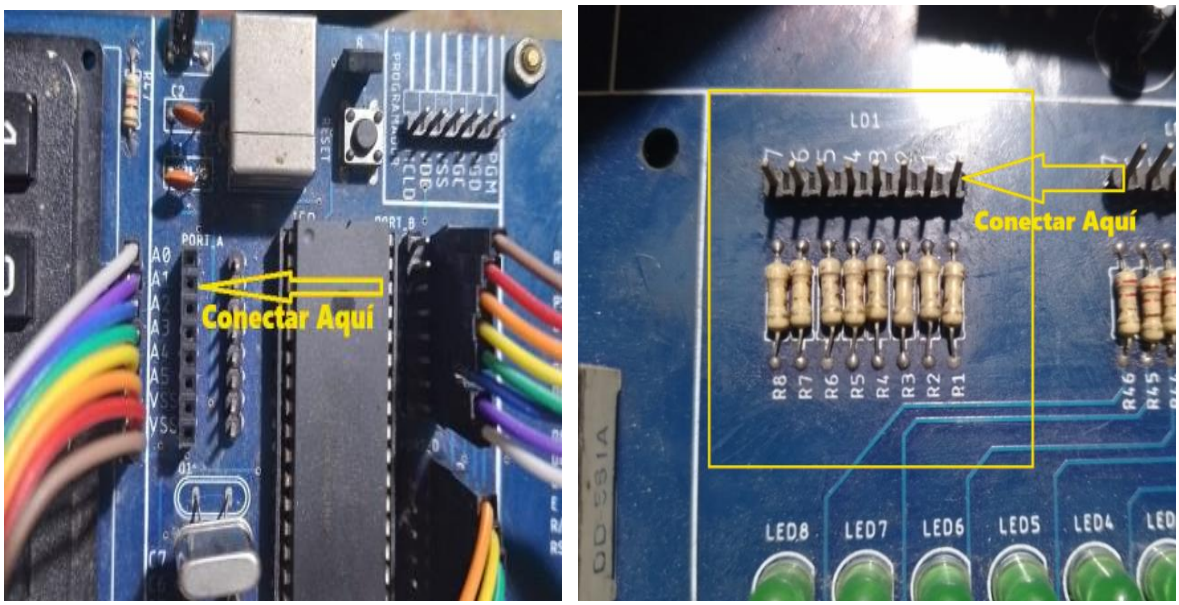


Figura 5. Conexión del LED Indicador de Activación de la Cerradura.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

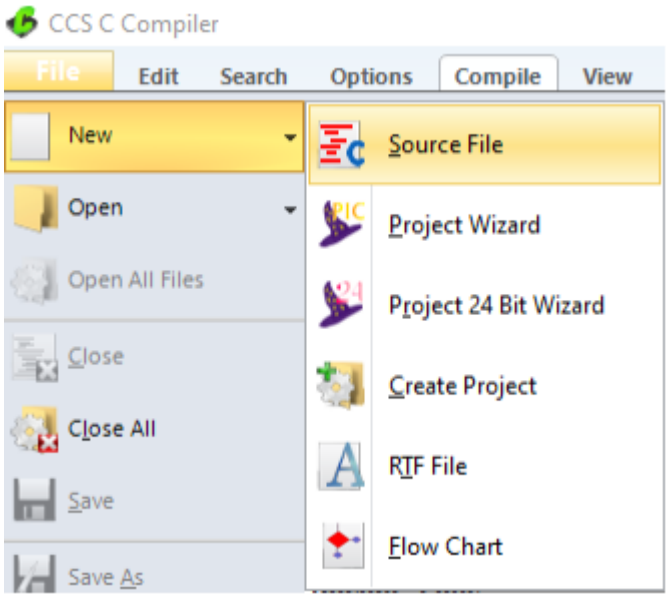


Figura 6. Crear un nuevo proyecto en el ide de CCS C Compiler

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18F4550.h>// Declaramos al Pic 18F4550
#fuses INTRC_IO //Fusible para usar el oscilador interno
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#fuses PUT // Fusible para activar el Power-Up Timer
#use delay(clock=20Mhz)//Seteamos el reloj externo a 20MHz
#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD
#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C
#include <stdlib.h>//libreria para usar printf
#BYTE PORT_A = 0xf80//Declaramos el puerto A
#BYTE PORT_B = 0xf81//Declaramos el puerto B

#define row0 PIN_B0                          //Filas del teclado colocar resistencia pullup
#define row1 PIN_B1                          //Filas del teclado colocar resistencia pullup
#define row2 PIN_B2                          //Filas del teclado colocar resistencia pullup
#define row3 PIN_B3                          //Filas del teclado colocar resistencia pullup
#define col0 PIN_B4                          //Columnas del teclado
#define col1 PIN_B5                          //Columnas del teclado
#define col2 PIN_B6                          //Columnas del teclado
#define col3 PIN_B7                          //Columnas del teclado
#include "Teclado4x4.h"                      //Librería de teclado
```

```
//Variables Globales
CHAR K;

void main(){

 WRITE_EEPROM(0,7); //Guarda en la eemprom posicion cero la nueva contraseña
 WRITE_EEPROM(1,2);
 WRITE_EEPROM(2,3);
 WRITE_EEPROM(3,3);

char k;   //declaracion de variable
int i;   //declaracion de variable
char data[4], clave[4] ;   //declaracion de arreglo de variable
lcd_init(0x4E,16,2);
lcd_backlight_led(ON); //Enciende la luz de Fondo
lcd_clear();  //Limpia el LCD
kbd_init();   //inicializa teclado
port_b_pullups(true); //inicializa pull ups puerto b

while (true){      //mientras sea verdad
i=0;       //i=0
lcd_gotoxy(1,1);
printf(lcd_putc,"\f pulsar tecla 1\n"); //muestra mensaje cursor posicion 1,2
//lazo de captura de datos-----------

while(i<=3){      //mientras i<=2 hacer

k=kbd_getc();       //captura dato del teclado
if(k!=0){          //pregunta si k es distinto (k!=0) a cero
 switch (k)
   {
      case 48:
         k=0;
         break;

      case 49:
         k=1;
         break;

      case 50:
         k=2;
         break;

      case 51:
         k=3;
         break;

      case 52:
         k=4;
         break;
```

```
    case 53:
       k=5;
       break;

    case 54:
       k=6;
       break;

    case 55:
       k=7;
       break;

    case 56:
       k=8;
       break;

    case 57:
       k=9;
       break;

    default:
       k=0;
       break;
 }
```

```
data[i]=k;          //guardo en espacio de data segun i el valor de k
i++;             //incremento i
if(i<4){
lcd_clear();  //Limpia el LCD
 lcd_gotoxy(1, 1);
printf(lcd_putc," pulsar tecla %u",i+1);
}      //imprime valor numerico}

lcd_gotoxy(1, 2);
printf(lcd_putc," # pulsado %u",k);

}
}
//------------------------------

for (i=0 ; i<=3 ; i++){   //incremento i de 0 hasta 2
clave[i]=read_eeprom(i);
delay_ms(50);
}   //guarda la clave de la eeprom en la variable clave
                         //en cada posicion de i, por eso hace lazo hasta i=2
```

```
if ((data[0]==clave[0])&&(data[1]==clave[1])&&(data[2]==clave[2])&&(data[3]==clave[3])){
printf(lcd_putc,"\f puerta abierta");      //muestra mensaje puerta abierta
output_high(pin_a1);
delay_ms(1000);
output_low(pin_a1);
}

else      //si los datos no son iguales
printf(lcd_putc,"\f acceso denegado");      //muestra mensaje puerta cerrada
delay_ms(1000);
}
}
```

Figura 7. Programación en CCS C Compiler.

Descripción paso a paso del código:

1. Configuración inicial:

Se configuran las palabras de configuración del microcontrolador PIC18F4550 a través de los fusibles (oscilador, watchdog, protección de código, etc.).

Se establece la velocidad del reloj externo en 20 MHz.

2. Configuración de puertos y librerías:

Se declaran los identificadores para los puertos A y B.

Se definen los pines para la comunicación I2C, que se utiliza para controlar una pantalla LCD.

Se incluye la librería `i2c_Flex_LCD.c` para facilitar la comunicación con el LCD.

Se configuran pines para conectar un teclado matricial 4x4 (filas y columnas).

3. Inicialización del LCD:

Se inicializa el LCD, se enciende la luz de fondo y se borra la pantalla.

4. Configuración de variables:

Se declaran las variables necesarias, incluyendo `K` para la lectura de teclas, `data` para almacenar los datos ingresados por el usuario y `clave` para almacenar la contraseña.

5. Bucle principal:

Se inicia un bucle principal infinito (`while (true)`) que se ejecutará constantemente.

6. Captura de datos del teclado:

Se muestra un mensaje en el LCD solicitando al usuario que ingrese una tecla (`pulsar tecla 1`).

El programa entra en un bucle para capturar cuatro dígitos ingresados por el usuario utilizando el teclado matricial.

Los dígitos ingresados se almacenan en el arreglo `data`.

7. Comparación con la contraseña almacenada:

Se lee la contraseña almacenada en la EEPROM y se almacena en el arreglo `clave`.

Se compara la contraseña ingresada por el usuario (`data`) con la contraseña almacenada (`clave`).

Si son iguales, se muestra un mensaje de "puerta abierta" en el LCD y se activa una señal de salida (`pin_a1`) durante 1 segundo.

Si no son iguales, se muestra un mensaje de "acceso denegado" en el LCD durante 1 segundo.

8. Reinicio del ciclo:

El programa vuelve al inicio del ciclo para esperar una nueva entrada del usuario.

Paso N.º 4: Compilar: **Según Practica # 0**

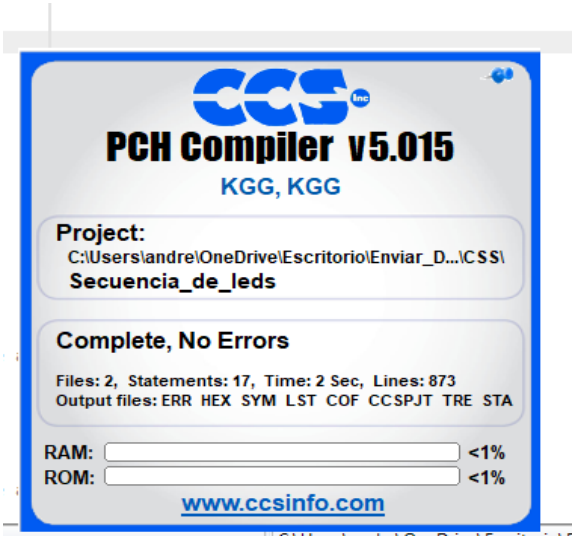


Figura 8. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practica # 0**

Figura 9. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

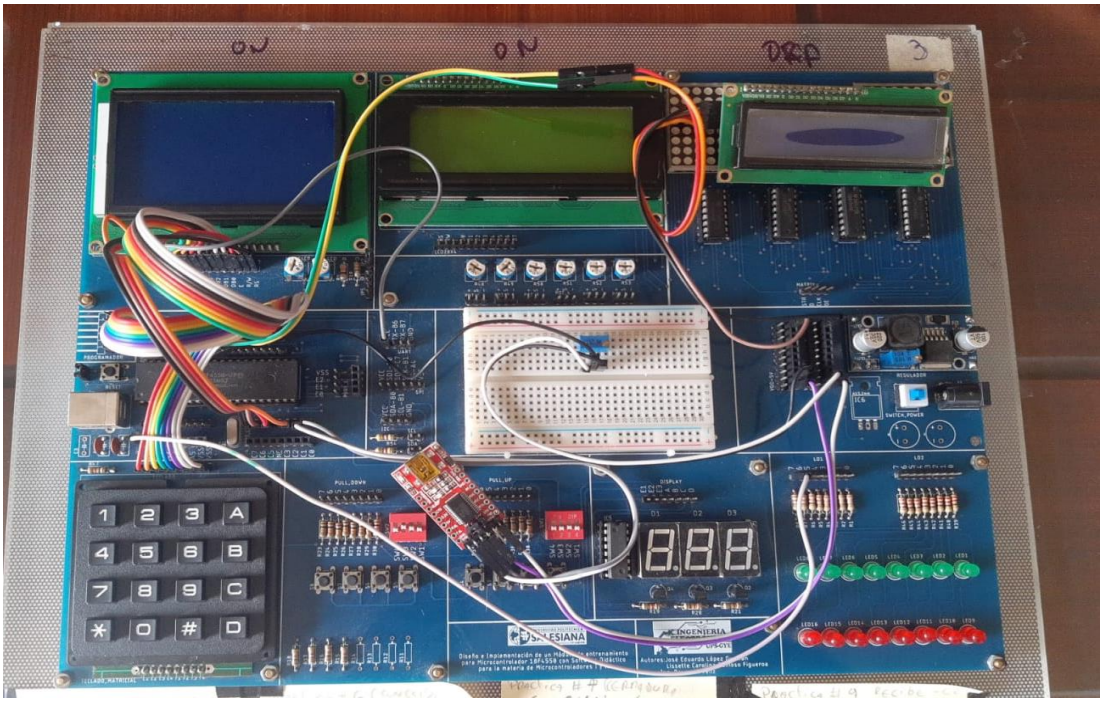


Figura 10. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 8**
NOMBRE PRÁCTICA:	Comunicación serial USP entre dos PICs.	

1. **Objetivo General.**

   Implementar un programa para poder realizar una comunicación serial entre dos PICs.

2. **Objetivos Específicos.**

1. Implementar un programa en el PIC que envía los datos.
2. Implementar un programa en el PIC que recibe los datos.

3. **Marco Teórico.**

Configuración de la Comunicación Serial:

Ambos microcontroladores deben configurar sus puertos serie RS-232 para que coincidan en términos de velocidad de baudios, bits de datos, paridad y configuración de bits de parada. Esto garantiza que la comunicación sea confiable y que los datos se interpreten correctamente.

Transmisión de Datos:

Uno de los microcontroladores actúa como transmisor y el otro como receptor. El microcontrolador transmisor envía datos a través del puerto serie RS-232, que luego se transmiten a través de un cable a la entrada del puerto serie del microcontrolador receptor.

Recepción de Datos:

El microcontrolador receptor está constantemente monitoreando su puerto serie RS-232 para detectar la llegada de datos entrantes. Cuando se reciben datos, el microcontrolador los interpreta y realiza las acciones correspondientes según los comandos o datos recibidos.

Comandos y Acciones:

Los datos transmitidos entre los microcontroladores pueden representar comandos o información. En función de estos datos, el microcontrolador receptor puede realizar diversas acciones, como controlar periféricos, mostrar información en una pantalla LCD, etc.

Feedback y Eco:

En algunos casos, el microcontrolador receptor puede enviar un eco de los datos recibidos de vuelta al microcontrolador transmisor como confirmación de que los datos se recibieron correctamente. (Muhammad Ali Mazidi, 2018 (Tercera edición).)

4. Descripción.

Para hacer funcionar esta práctica, debes conectar un LED al pin `PIN_C2` del módulo que recibe. Además, debes conectar otro módulo (el que envía) para poder enviar comandos 'A' y 'B' desde el por medio de comunicación serial (para enviar estos comandos se presionan dos botones conectados a resistencias Pull Up y conectar a los pines `PIN_D0` y `PIN_D1`).

Cuando presiones el botón 'A', el microcontrolador detectará la pulsación y enviará "A\r\n" a través de la comunicación serial. Del mismo modo, cuando presiones el botón 'B', enviará "B\r\n".

Cuando envíes 'A' a través de la comunicación serial, el LED se encenderá y se mostrará "LED Encendido" en el LCD. Cuando envíes 'B', el LED se apagará y se mostrará "LED Apagado" en el LCD.

Esta práctica permite controlar un LED a través de comandos enviados por comunicación serial RS-232 y muestra el estado del LED en un LCD I2C conectado al microcontrolador PIC18F4550.

5. Materiales.

- Dos Módulos didácticos.
- Jumpers Conectores.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

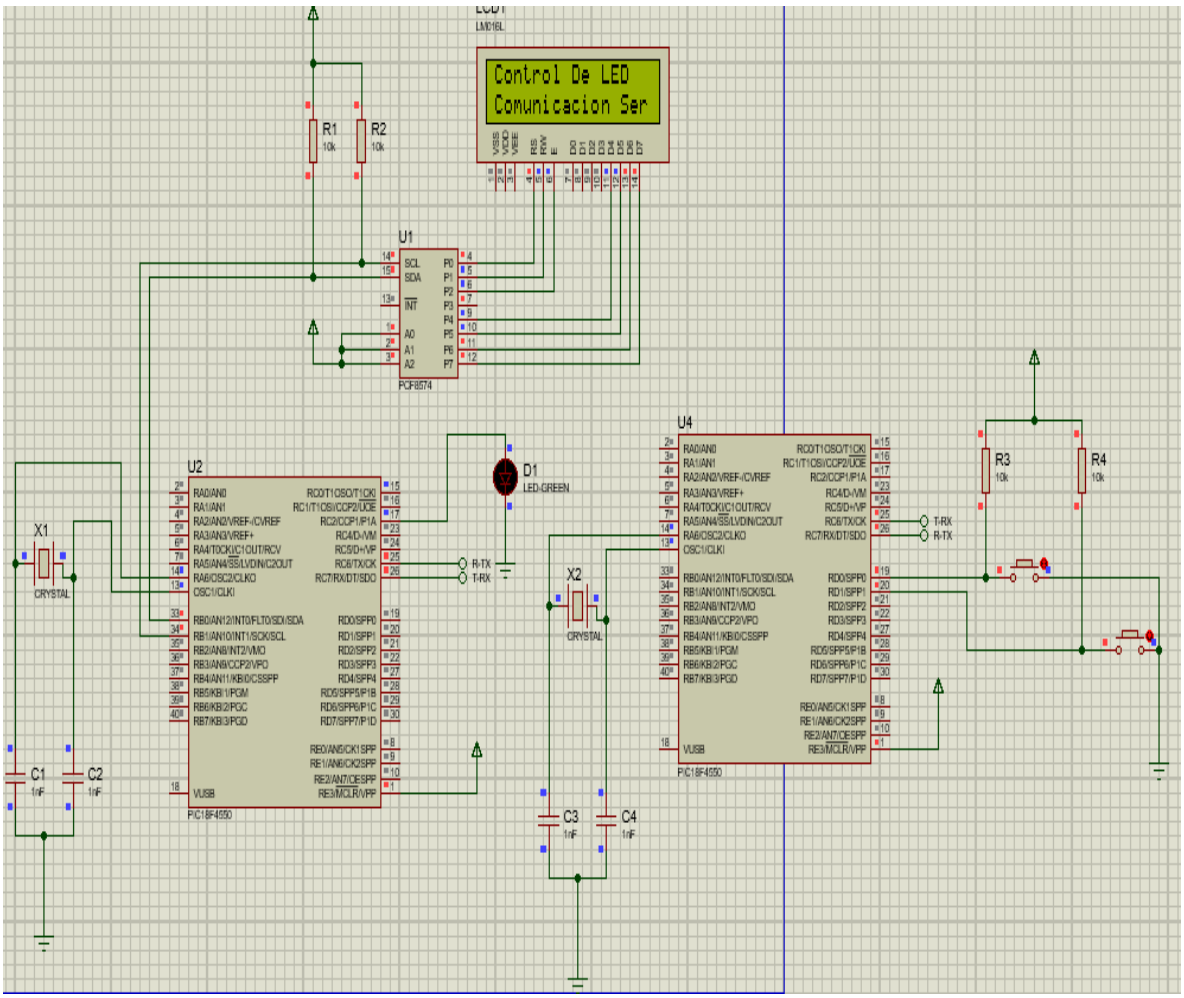


Figura 1. Simulación en Proteus.

Pines del PIC 1 (El que envía)	Pines del Módulo-Pulsadores
C6	RX-PIC2
C7	TX-PIC2
D0	R31-PULLUP7
D1	R32-PULLUP6
Pines del PIC 2 (El que recibe)	**Pines del Módulo-LCD I2C/Led**
B0	SDA-LCD I2C
B1	SCL-LCD I2C
C2	R1-LED1
C6	TX-PIC1
C7	RX-PIC1
Pines del LCD I2C	
VCC-LCD	VDD-5V
GND-LCD	VSS-0V

PIC 1 (EL QUE ENVÍA)

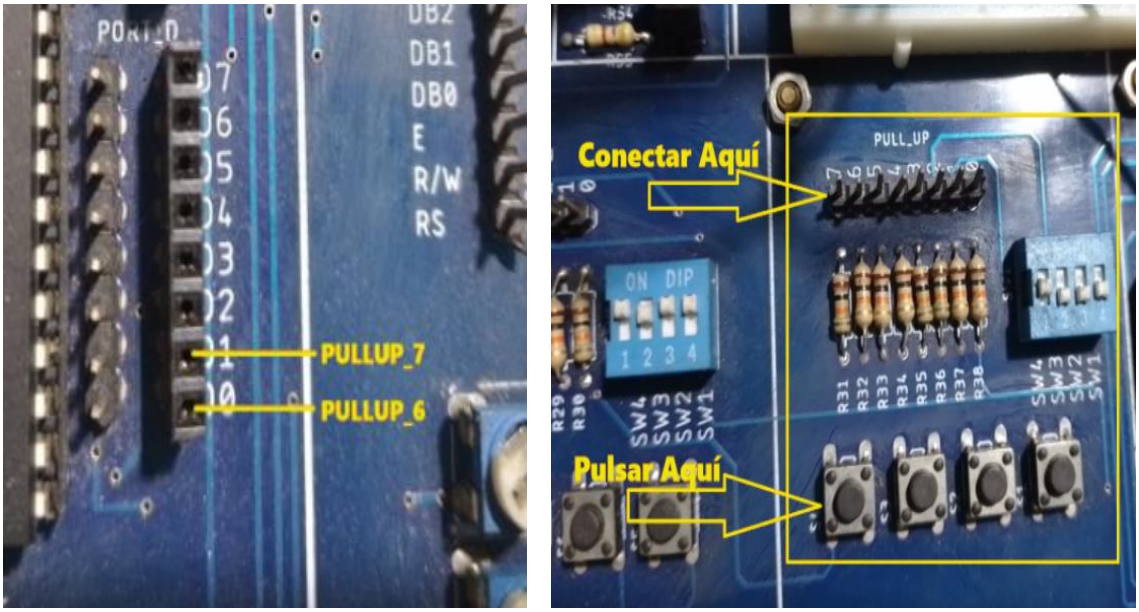


Figura 2. Conexión de los pulsadores de encendido y apagado.

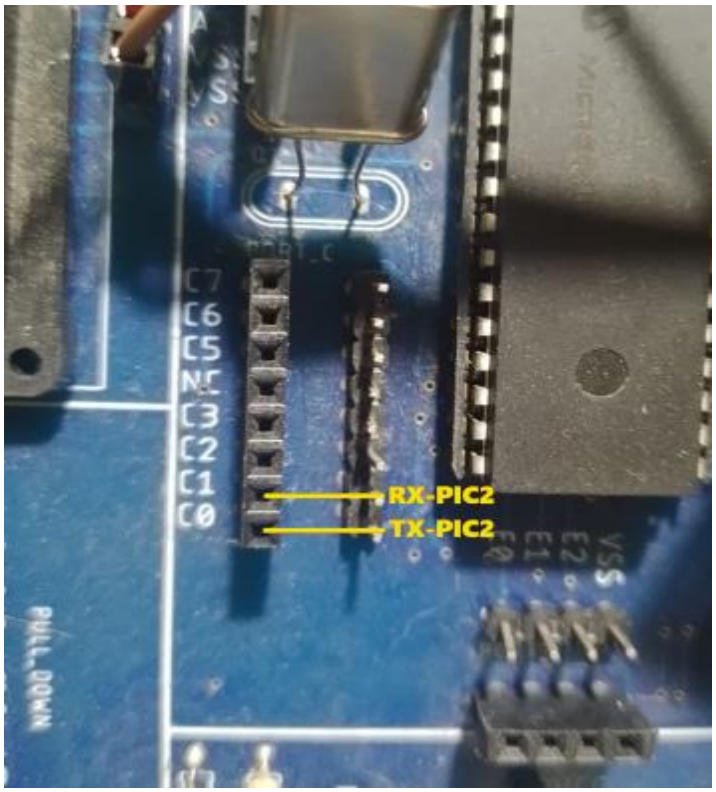


Figura 3. Conexión de los pines seriales del PIC 1.

PIC 2 (EL QUE RECIBE)

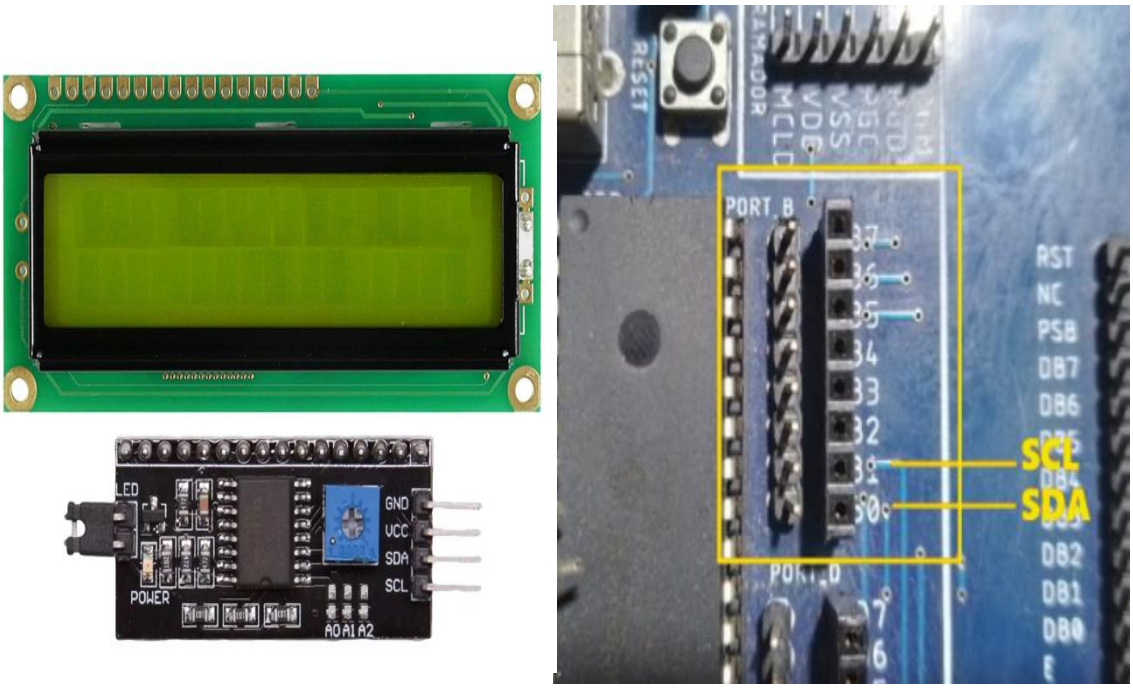


Figura 4. Conexión del Teclado al PIC 2.

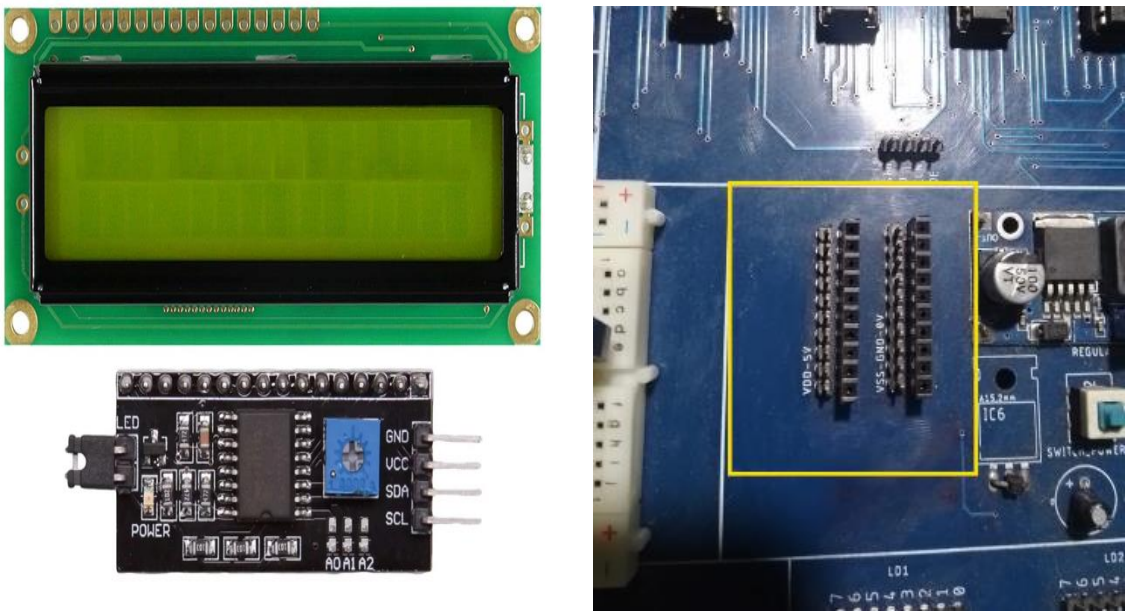


Figura 5. Conexión del LCD a la alimentación.

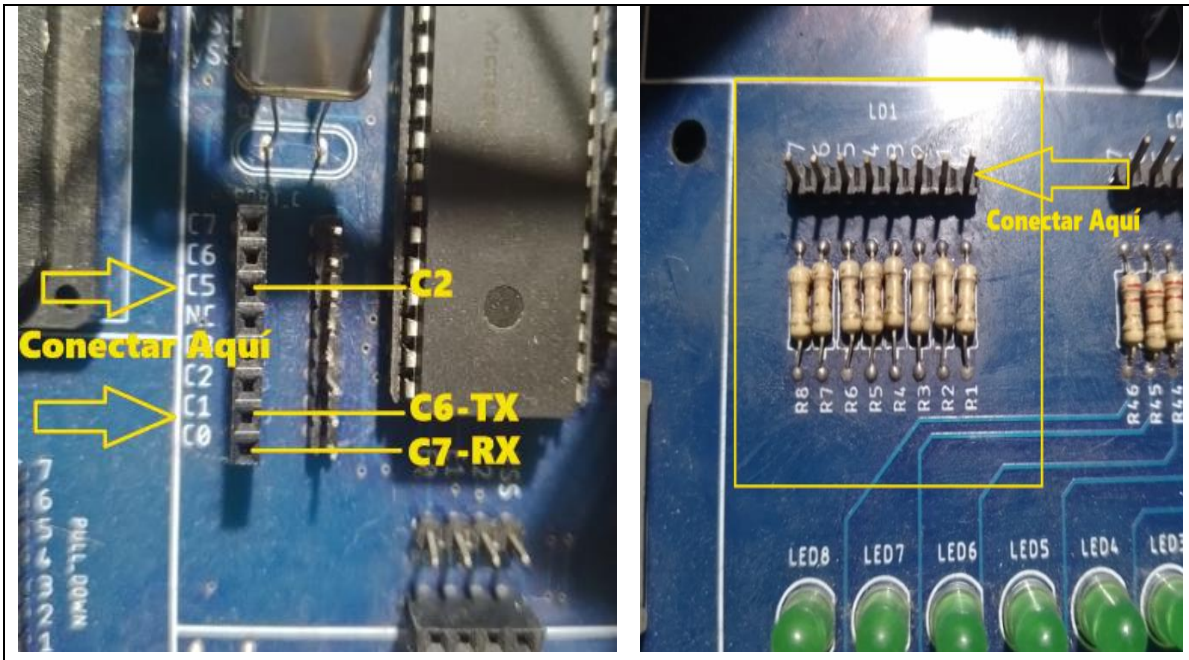


Figura 6. Conexión del LED Indicador de Envío/Recepción de datos.

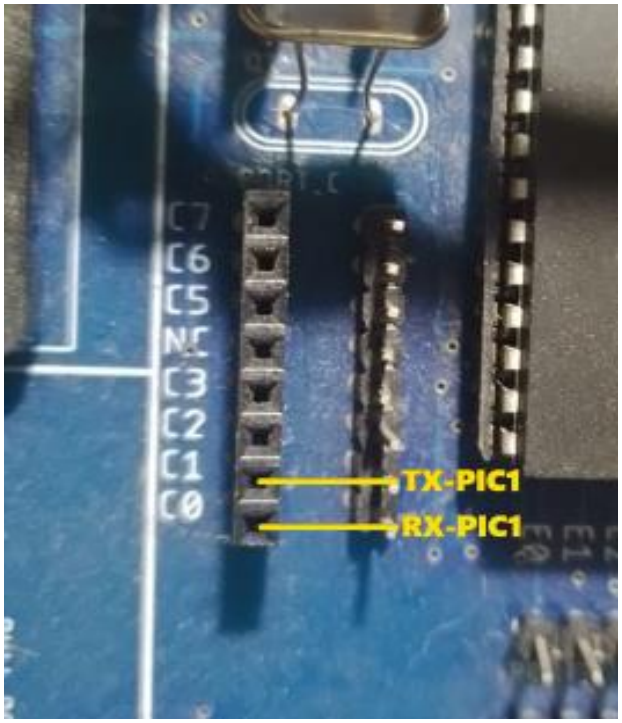


Figura 7. Conexión de los pines seriales del PIC 2.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

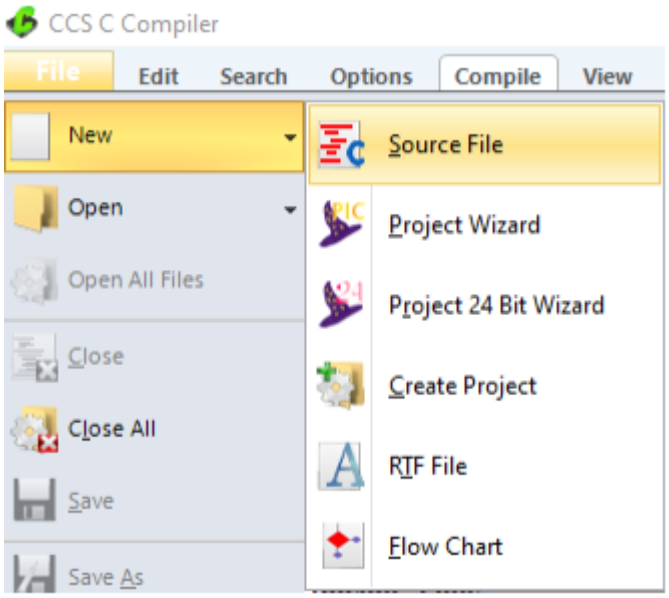


Figura 8. Crear un nuevo proyecto en el ide de CCS C Compiler

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

PIC 1 (EL QUE ENVÍA)

```
#include <18F4550.h> //Libreria para usar el PIC18F4550
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#use delay(clock=20000000)//reloj externo
#use rs232(baud=9600, xmit=PIN_c6,rcv=PIN_c7,parity=N, bits=8)//Declaramos
                                                        //la comunicación serial
#define BUTTON_A  PIN_D0 // Definimos el botón para el comando de 'A' alto
#define BUTTON_B  PIN_D1 // Definimos el botón para el comando de 'B' bajo

void main()
{

    set_tris_d(0xFF); // Seteamos el puerto D como entradas

    while (1)
    {
        if (!input(BUTTON_A)) // Si el botón de alto es presionado
        {
            delay_ms(100); // un delay
            printf("A\r\n"); // envia 'A' por serial
        }

        if (!input(BUTTON_B)) // Si el botón de bajo es presionado
        {
            delay_ms(100); // un delay
            printf("B\r\n"); // envia 'B' por serial
        }

        delay_ms(20); // retardo para estabilidad
    }
}
```

Figura 9. Programación en CCS C Compiler.

Descripción paso a paso del código:

1. Incluye la librería `18F4550.h` para utilizar el microcontrolador PIC18F4550 y configura los fusibles para el funcionamiento del oscilador, el watchdog timer, la protección del código y la programación por bajo voltaje.

2. Establece la frecuencia del reloj externo en 20 MHz con `#use delay(clock=20000000)`.

3. Configura la comunicación serial RS-232 con una velocidad de baudios de 9600, utilizando los pines `PIN_C6` para transmitir (`xmit`) y `PIN_C7` para recibir (`rcv`). También se especifica que no se utilizará paridad y se enviarán 8 bits de datos.

4. Se definen las constantes `BUTTON_A` y `BUTTON_B` que representan los pines `PIN_D0` y `PIN_D1`, respectivamente. Estos pines se utilizan para conectar los botones que enviarán los comandos 'A' y 'B' al microcontrolador.

5. En el bucle principal `main()`, se configura el puerto D (puerto de entrada) con `set_tris_d(0xFF)`, lo que significa que los pines D0 y D1 se configurarán como entradas para leer el estado de los botones.

6. Dentro del bucle infinito `while (1)`, el programa realiza las siguientes acciones repetidamente:

Verifica si el botón etiquetado como 'A' se presiona (`!input(BUTTON_A)`). Si se detecta una pulsación, espera un breve retardo de 100 ms y luego envía el carácter 'A' seguido de retorno de carro y salto de línea ("\r\n") a través de la comunicación serial usando `printf("A\r\n")`.

Verifica si el botón etiquetado como 'B' se presiona (`!input(BUTTON_B)`). Si se detecta una pulsación, espera un breve retardo de 100 ms y luego envía el carácter 'B' seguido de retorno de carro y salto de línea ("\r\n") a través de la comunicación serial usando `printf("B\r\n")`.

Después de procesar las pulsaciones de botones, se agrega un pequeño retardo de 20 ms para asegurar la estabilidad del programa.

PIC 2 (EL QUE RECIBE)

```
#include <18F4550.h>//Libreria para usar el PIC18F4550
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#use delay(clock=20000000) //reloj externo
#use rs232(baud=9600, xmit=PIN_c6,rcv=PIN_c7,parity=N, bits=8)//Declaramos la comunicación serial
#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD
#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C
#include <stdio.h>//para usar printf
#include <math.h> // libreria para claculos matemáticos

void main()
{
lcd_init(0x4E,16,2);
lcd_backlight_led(ON); //Enciende la luz de Fondo
lcd_clear();  //Limpia el LCD
    set_tris_c(0b11111110); // Setea RC2 como salida, el resto de pines como entradas
    lcd_gotoxy(1,1);
    lcd_putc("Control De LED");
    lcd_gotoxy(1,2);
    lcd_putc("Comunicacion Serial");
    while (1)
    {
        if (kbhit()) // Chequea si hay un dato disponible en el buffer UART
        {
            char datoRecibido = getc(); // Reibe un dato desde el UART
            putc(datoRecibido);        // Eco del dato recibido

            if (datoRecibido == '\r') // Chequea por un final de linea
            {
                putc('\n'); // agrega un caracter de nueva linea
            }

             if (datoRecibido == 'A') // Si dato recibido es 'A' alto
             {
              LCD_PUTC("\f");   //Borrar el contenido del LCD
              lcd_gotoxy(1,1);
              lcd_putc("LED");
              lcd_gotoxy(1,2);
              printf(lcd_putc,"Encendido");
                 output_high(PIN_C2); // Setea RC2 alto
             }
             else if (datoRecibido == 'B') // Si dato recibido es 'B' bajo
             {
                 LCD_PUTC("\f");   //Borrar el contenido del LCD
                   lcd_gotoxy(1,1);
                   lcd_putc("LED");
                  lcd_gotoxy(1,2);
                  printf(lcd_putc,"Apagado");
                 output_low(PIN_C2); // Setea RC2 bajo
             }
        }
    }
}
```

Figura 10. Programación en CCS C Compiler

Descripción paso a paso del código:

1. Incluye la librería `18F4550.h` para utilizar el microcontrolador PIC18F4550 y configura los fusibles para el funcionamiento del oscilador, el watchdog timer, la protección del código y la programación por bajo voltaje.

2. Establece la frecuencia del reloj externo en 20 MHz con `#use delay(clock=20000000)`.

3. Configura la comunicación serial RS-232 con una velocidad de baudios de 9600, utilizando los pines `PIN_C6` para transmitir (`xmit`) y `PIN_C7` para recibir (`rcv`). También se especifica que no se utilizará paridad y se enviarán 8 bits de datos.

4. Se declara la comunicación I2C para un LCD utilizando los pines `PIN_B0` como SDA y `PIN_B1` como SCL con una velocidad de transmisión rápida de 100,000 bits por segundo.

5. Incluye las librerías `i2c_Flex_LCD.c`, `stdio.h` y `math.h` para utilizar funciones relacionadas con el LCD y cálculos matemáticos.

6. En el bucle principal `main()`, se realiza la inicialización del LCD, se enciende la luz de fondo y se limpia el LCD para comenzar con una pantalla en blanco con un mensaje de "Control De LED" en la línea 1 y "Comunicacion Serial" en la línea 2.

7. Se configura el pin `PIN_C2` como salida para controlar un LED.

8. En un bucle infinito `while (1)`, el programa verifica si hay datos disponibles en el buffer UART utilizando `kbhit()`. Si hay datos disponibles, se realiza lo siguiente:

Se lee un carácter recibido desde el UART utilizando `getc()` y se envía de vuelta como eco utilizando `putc(datoRecibido)`.

Si el carácter recibido es un retorno de carro `'\r'`, se envía un carácter de nueva línea `'\n'`. Si el carácter recibido es 'A', se borra el contenido del LCD, se coloca "LED" en la línea 1 y "Encendido" en la línea 2 del LCD, y se establece el pin `PIN_C2` en alto para encender el LED.

Si el carácter recibido es 'B', se borra el contenido del LCD, se coloca "LED" en la línea 1 y "Apagado" en la línea 2 del LCD, y se establece el pin `PIN_C2` en bajo para apagar el LED.

Paso N.º 4: Compilar: **Según Practica # 0**

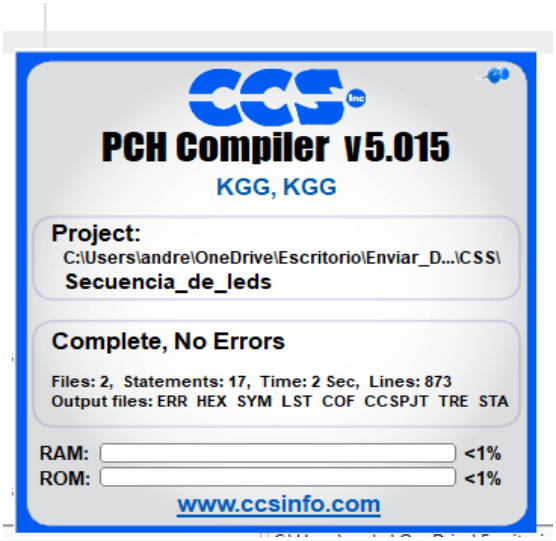


Figura 11. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Práctica # 0**

Figura 12. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

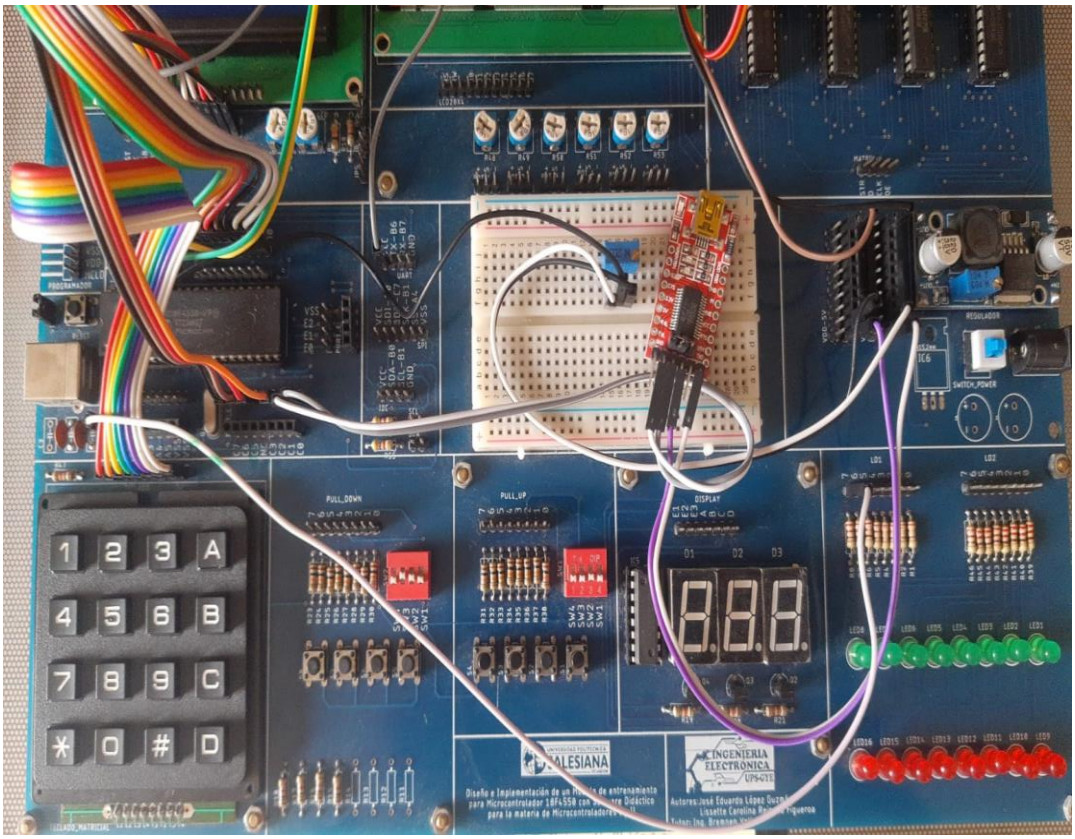


Figura 13. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 9**
NOMBRE PRÁCTICA:	Control de velocidad de un motor mediante un potenciómetro y teclado matricial.	

1. **Objetivo General.**

   Implementar un programa para poder controlar la velocidad de un motor.

2. **Objetivos Específicos.**

1. Implementar un programa en el cual, mediante el teclado matricial, un potenciómetro y PWM se controla la velocidad de un motor.

3. **Marco Teórico.**

   IRF540N - MOSFET de Potencia:

   El IRF540N es un transistor de efecto de campo de metal-óxido-semiconductor (MOSFET) de potencia. Se utiliza comúnmente en aplicaciones de conmutación de alta potencia, como el control de motores y la regulación de tensión. Aquí hay algunas especificaciones clave del IRF540N:

   Tensión Máxima de Drenaje-Source (VDS): El IRF540N tiene una VDS máxima de 100V. Esto significa que puede soportar una diferencia de potencial de hasta 100 voltios entre el drenaje y la fuente.

   Corriente Máxima de Drenaje Continuo (ID): Tiene una ID continua máxima de aproximadamente 33 amperios. Esto indica la cantidad máxima de corriente que puede circular a través del dispositivo cuando está completamente activado.

   Resistencia de Encendido (RDS(on)): La resistencia de encendido típica (RDS(on)) para el IRF540N es de alrededor de 0.044 ohmios. Esta resistencia representa la caída de tensión a través del MOSFET cuando está encendido. Menor RDS(on) indica una menor pérdida de energía.

   Capacidad de Conmutación: El IRF540N puede conmutar a alta velocidad debido a su baja capacitancia de entrada y salida, lo que lo hace adecuado para aplicaciones de PWM.

El control de velocidad de motores DC:

Es esencial en una amplia variedad de aplicaciones, desde robótica hasta sistemas de automatización industrial. En este contexto, se utiliza una técnica llamada Modulación por Ancho de Pulso (PWM) para controlar la velocidad del motor.

Modulación por Ancho de Pulso (PWM):

La PWM es una técnica que se utiliza para variar la cantidad de potencia entregada a un dispositivo, como un motor, mediante el control de la duración de los pulsos de una señal cuadrada. En este caso, se utiliza para controlar la velocidad del motor DC.

El principio detrás de la PWM es que el motor recibe un promedio ponderado de la tensión de alimentación durante un período de tiempo determinado. Si el ciclo de trabajo (el tiempo que el pulso está en estado alto) es alto, el motor funcionará a alta velocidad, y si el ciclo de trabajo es bajo, el motor funcionará a baja velocidad. (Nashelsky, 1996)

4. Descripción.

En el funcionamiento de la práctica el PIC18F4550 está conectado a un potenciómetro y a un teclado numérico. El potenciómetro y el teclado controlan la velocidad del motor, y esta velocidad se muestra en la pantalla LCD. El teclado numérico permite al usuario configurar la velocidad deseada presionando la tecla 'A' y luego ingresando el valor de velocidad en porcentaje. También permite controlar la velocidad usando el potenciómetro presionando la tecla 'B' y detener el modo de control presionando '#'.

El código controla un motor y permite al usuario configurar y ajustar la velocidad tanto a través de un teclado numérico como mediante un potenciómetro, proporcionando una interfaz interactiva para el usuario.

Nota: (Como ya se había mencionado en la teoría el mosfet usado para la práctica puede soportar hasta un motor con un voltaje de 100VDC y 33 Amperios siempre y cuando se posea una fuente de alimentación externa con esa capacidad y se conecte de manera común el pin GND. En esta práctica se usa un motor de 5VDC ya que esta alimentación se posee propia del módulo).

5. Materiales.

- Módulo didáctico.
- Motor de 5VDC.
- Mosfet IRF540N.
- Diodo 1N4001
- Potenciómetro de 10KΩ.
- Jumpers Conectores.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

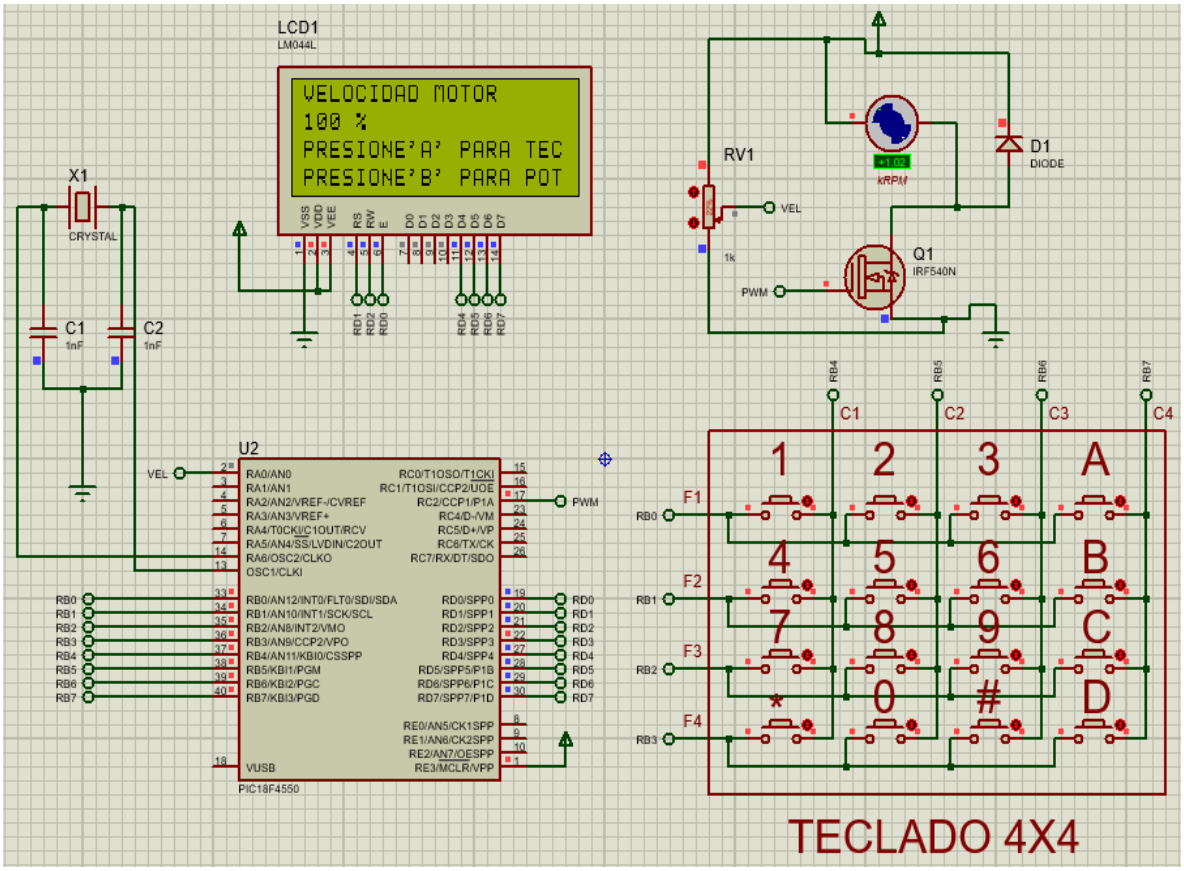


Figura 1. Simulación en Proteus.

Pines del PIC	Pines del Teclado/LCD/POT/Mosfet
B0	r1-TECLADO
B1	r2-TECLADO
B2	r3-TECLADO
B3	r4-TECLADO
B4	c1-TECLADO
B5	c2-TECLADO
B6	c3-TECLADO
B7	c4-TECLADO
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
A0	3-POTENCIÓMETRO
C2	1-GATE MOSFET
Pines del Potenciómetro	

1-POTENCIÓMETRO	VDD-5V
2-POTENCIÓMETRO	VSS-0V
Pines del Mosfet	
2-DRAIN MOSFET	NEGATIVO (-) MOTOR
3-SOURCE MOSFET	VSS-0V
Pines del Motor	
POSITIVO (+) MOTOR	VDD-5V; ÁNODO-DIODO
NEGATIVO (-) MOTOR	CÁTODO-DIODO

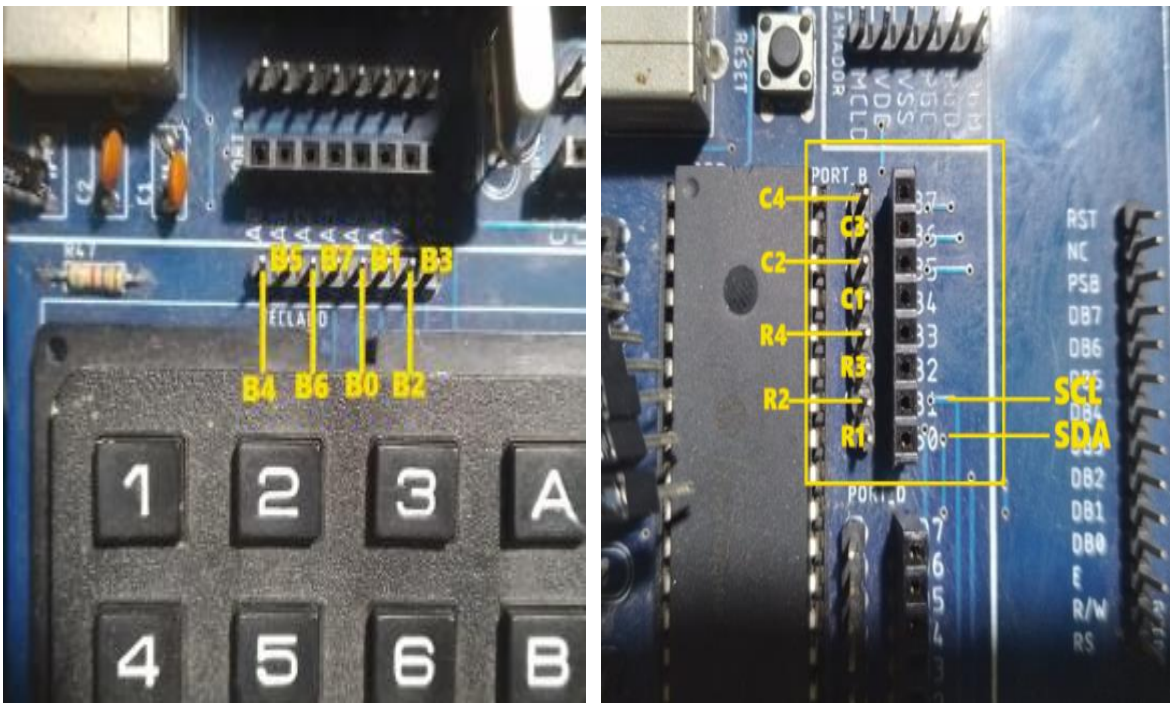


Figura 2. Conexión del Teclado al PIC.

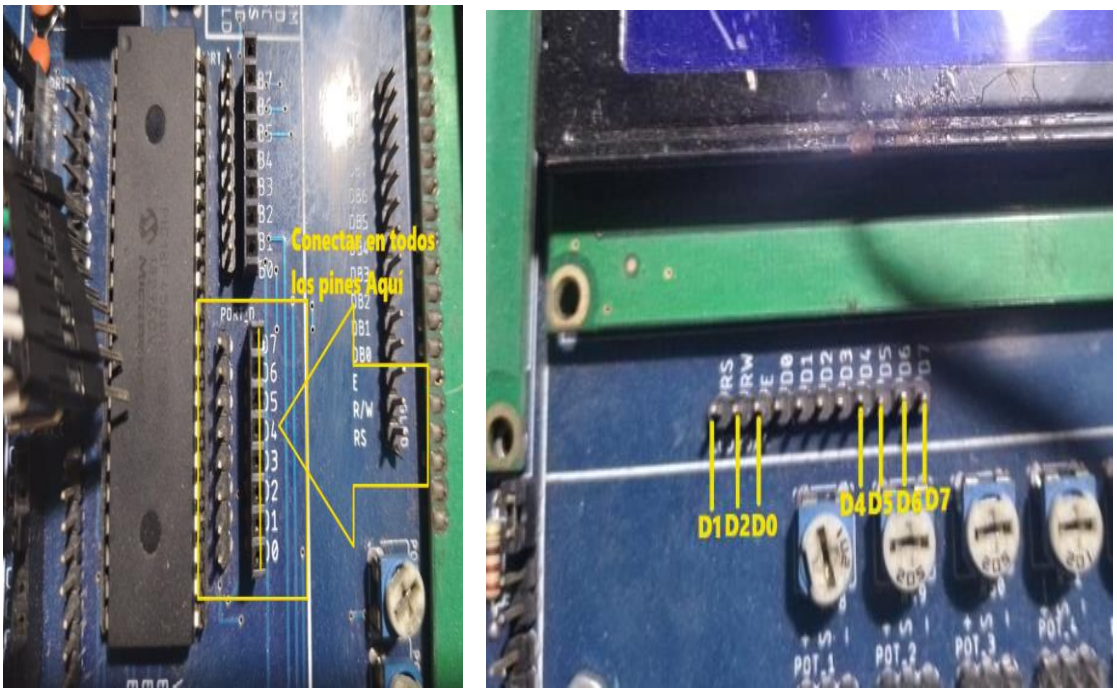


Figura 3. Conexiones del Puerto D a la pantalla LCD.

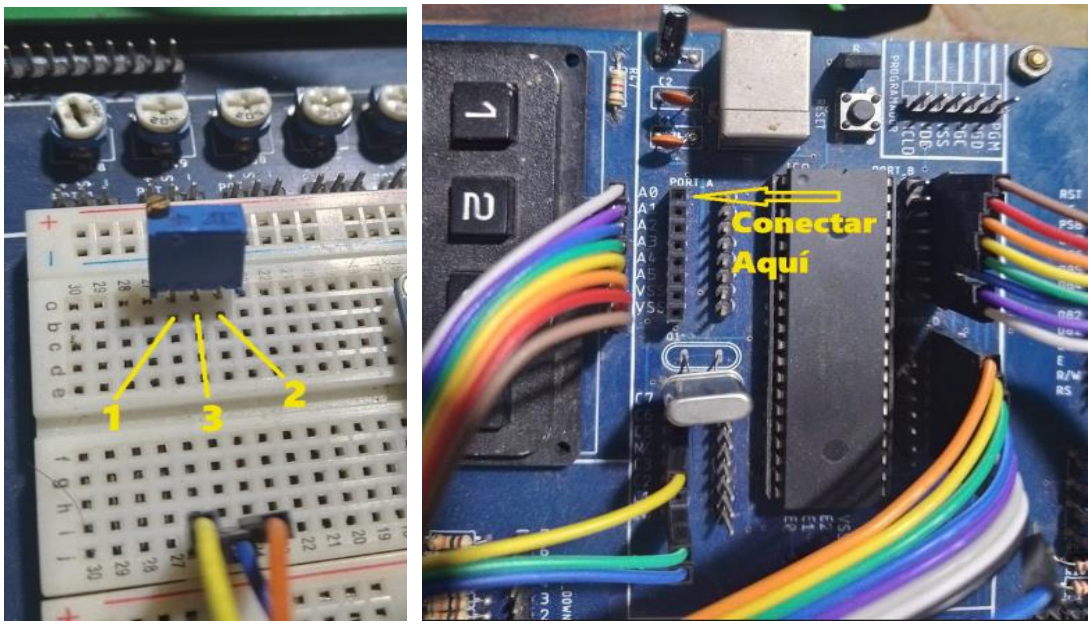


Figura 4. Conexión del potenciómetro al PIC.

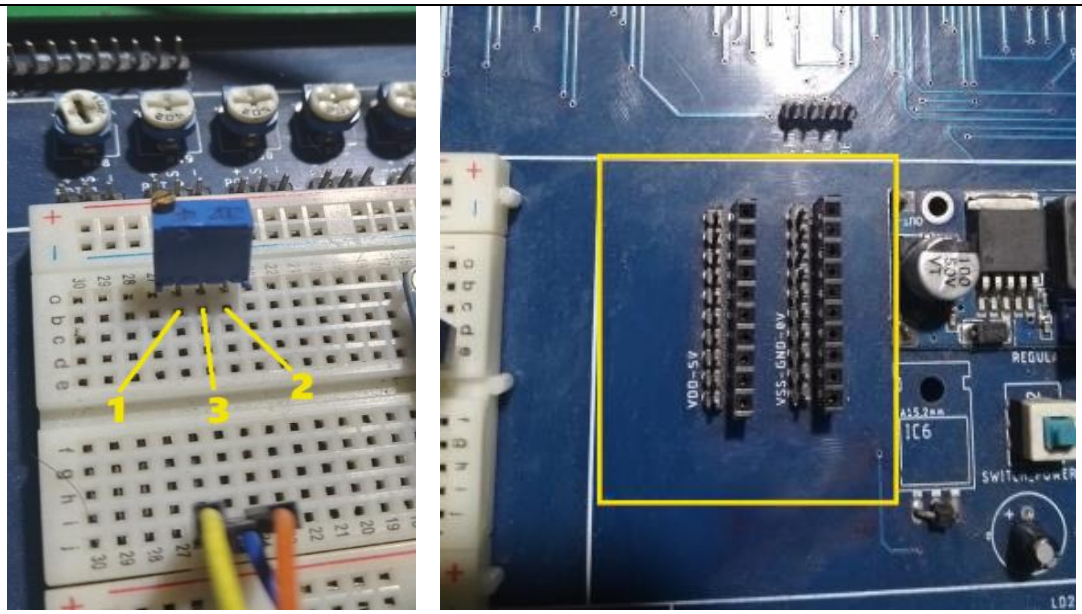


Figura 5. Conexión del potenciómetro a la alimentación.

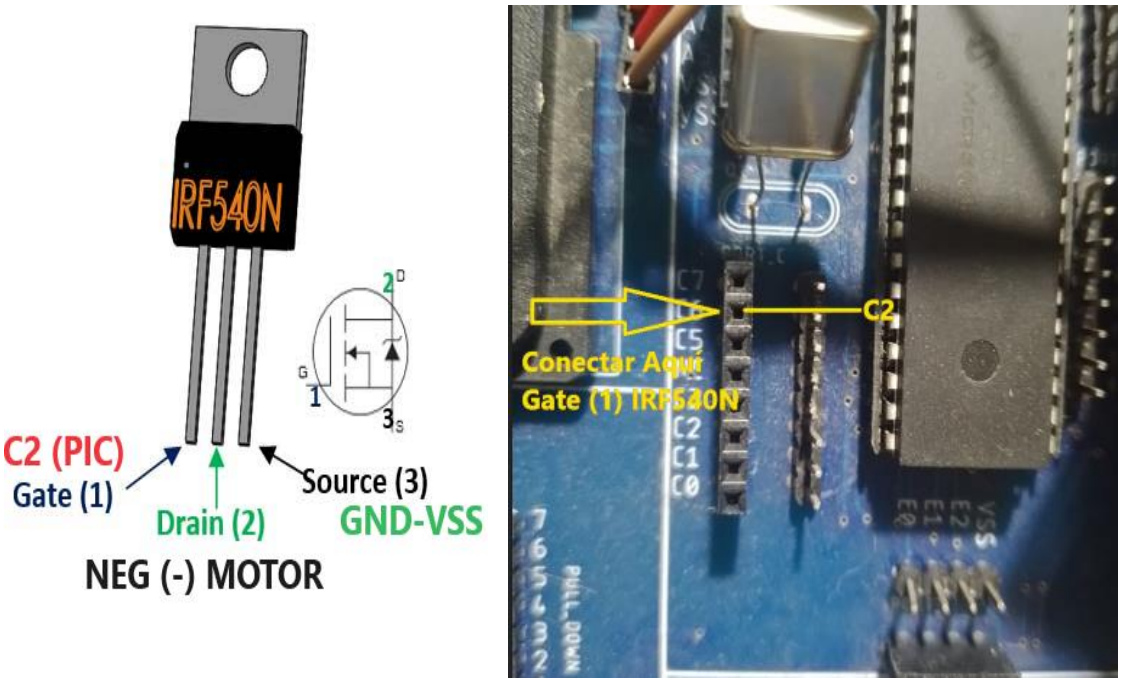


Figura 6. Conexión del Mosfet al PIC.

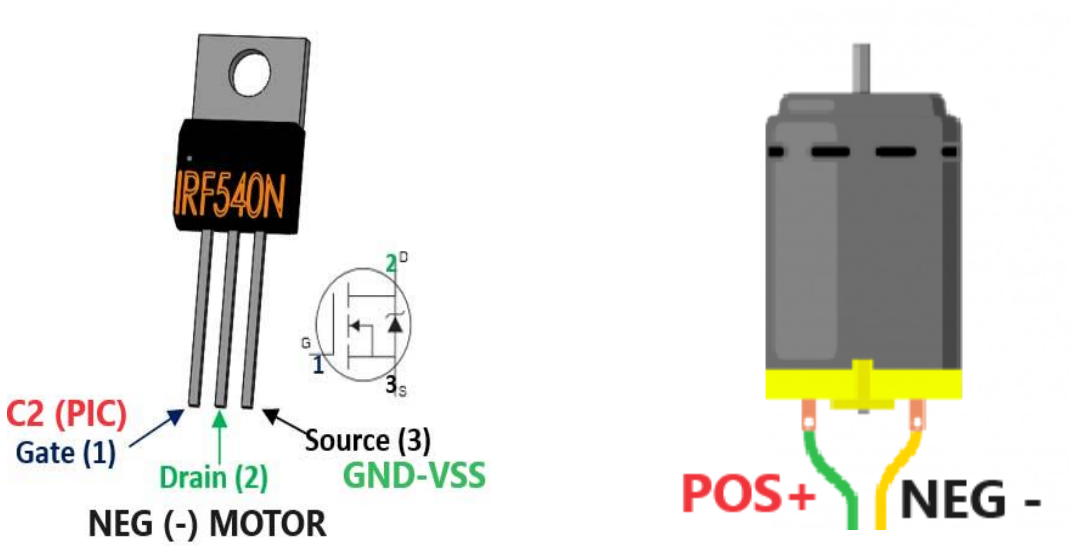


Figura 7. Conexión del Mosfet al Motor NEG -.

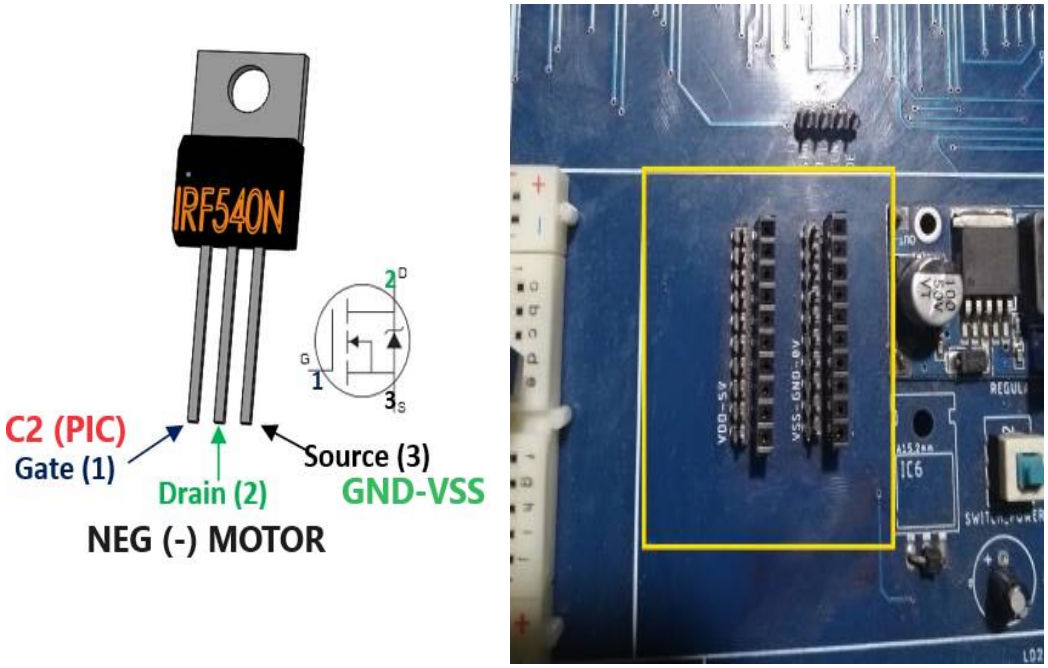


Figura 8. Conexión del Mosfet a VSS.

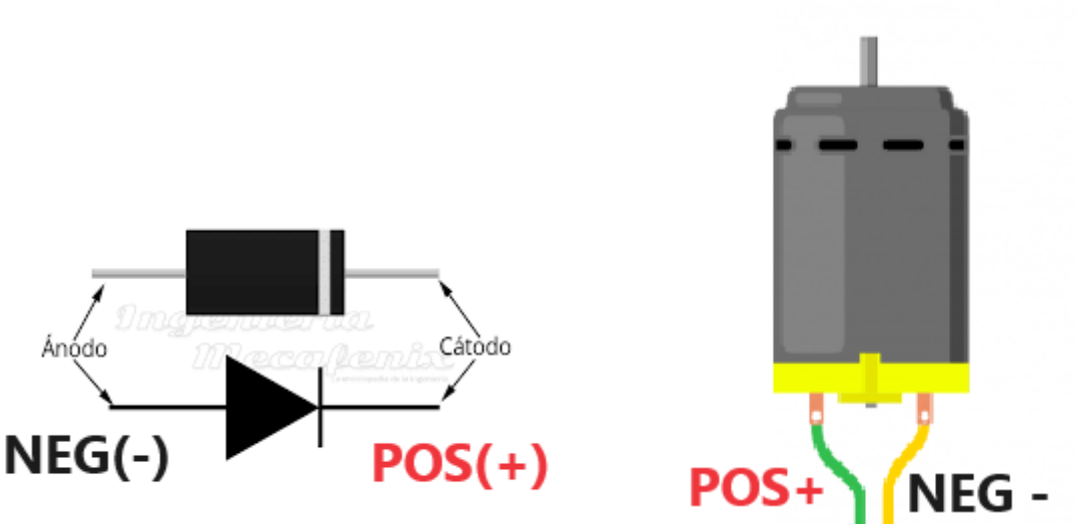


Figura 9. Conexión del Diodo al Motor.

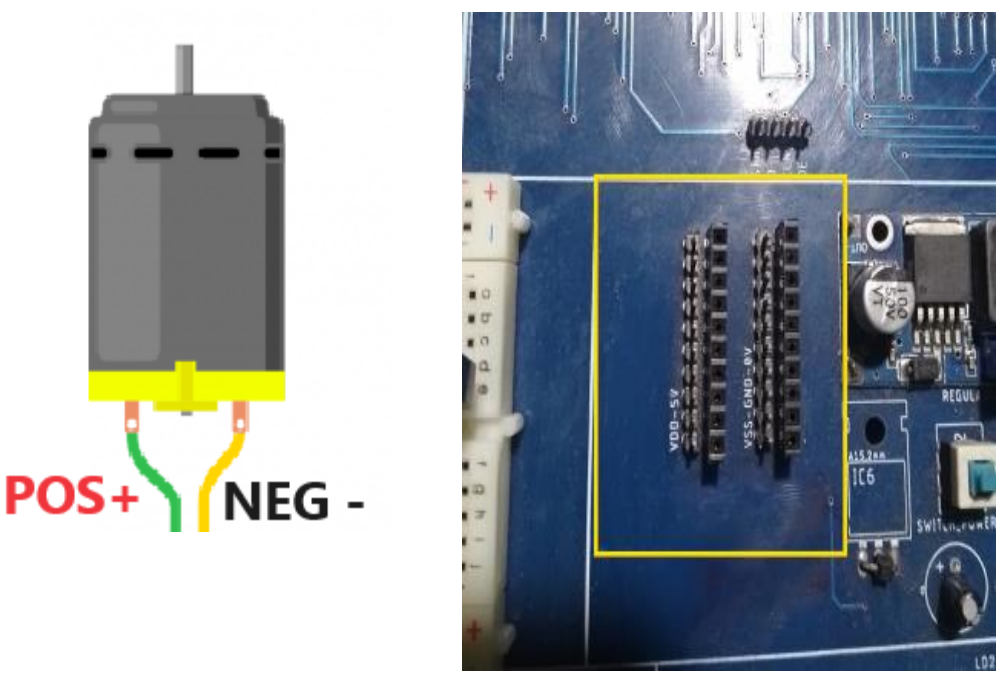


Figura 10. Conexión del Motor POS+ a VDD.

Paso N. º2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

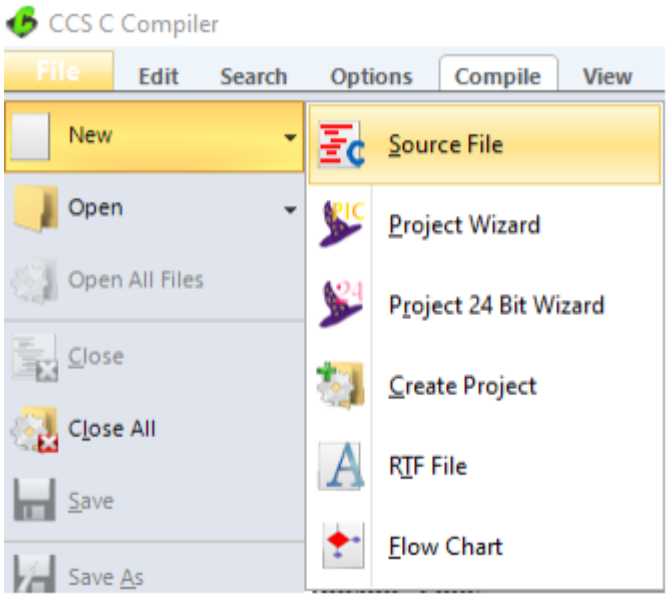


Figura 11. Crear un nuevo proyecto en el ide de CCS C Compiler

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#include <18F4550.h>// Declaramos al Pic 18F4550
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#fuses NODEBUG // Fusible para no usar la función debug
#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out
#DEVICE ADC=10 //Seteamos la conversión analógica con una resolución de 10 bits
#USE DELAY(clock=20000000)//Seteamos el reloj externo a 20MHz

#BYTE PORT_A = 0xf80//Declaramos el puerto A
#BYTE PORT_B = 0xf81//Declaramos el puerto B
#BYTE PORT_C = 0xf82//Declaramos el puerto C
#BYTE port_D = 0xf83//Declaramos el puerto D

# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento
# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento
# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento
# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento
```

```
#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos

#INCLUDE <LCD.C> // Incluimos la libreria de la pantalla LCD

////////Definimos los pines de la pantalla LCD///////
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////Definimos los pines de la pantalla LCD///////

#define row0 PIN_B0                             //Filas del teclado colocar resistencia pullup
#define row1 PIN_B1                             //Filas del teclado colocar resistencia pullup
#define row2 PIN_B2                             //Filas del teclado colocar resistencia pullup
#define row3 PIN_B3                             //Filas del teclado colocar resistencia pullup
#define col0 PIN_B4                             //Columnas del teclado
#define col1 PIN_B5                             //Columnas del teclado
#define col2 PIN_B6                             //Columnas del teclado
#define col3 PIN_B7                             //Columnas del teclado
#include "Teclado4x4.h"                         //Librería de teclado

#include <stdlib.h>//libreria para usar printf
#include <string.h>//Libreria para convertir lo ingresado por el teclado a float

int16 adc,velocidad=0; //declaramos las variables para la lectura del potenciómetro y la velocidad del motor
float R=0; // Variable auxiliar para la velocidad
char c; // variable para guardar el caracter que se ingresa por el teclado


/*=========================================================================*/
/*======================       FUNCION TECLA         ======================*/
/*=========================================================================*/
//Funcion encargada de esperar a que se presione una tecla
char tecla(void)
{
   char c;
   do{ //espera hasta que se presione una tecla
      c=kbd_getc(); //Captura valor del teclado
     }
   while(c=='\0');
   return(c);
}

/*=========================================================================*/
/*=======================    FUNCION TECLA CON TIMER  =====================*/
/*=========================================================================*/
// Pregunta por una Tecla por un tiempo, si no hay actividad, deja de preguntar
// y deja que el PIC continue con su trabajo

char tecla_time(void) {
   char c='\0';
   unsigned int16 timeout;
   timeout=0;
   c=kbd_getc(); //Captura valor del teclado
   while(c=='\0' && (++timeout< (KEYHIT_DELAY*150)))
   {
      delay_us(10);
      c=kbd_getc(); //Captura valor del teclado
   }
   return(c);
}
```

```
/*=============================================================================*/
/*============     FUNCION PARA DIGITAR LA VELOCIDAD  =====================*/
/*=============================================================================*/
long porcentaje(int nd)
{
   //Esta funcion captura el escalon desde el teclado, si el proceso está tomando
   //datos el escalon sirve para exitar el sistema, por otro lado si el sistema
   //está controlando, el escalo sirve para establecer el setpoint del sistema

   long val;
   int i;
   char str[6]; //Variable tipo String

   str[0]='0';
   for(i=0;i<nd;i++)
   {

      c=tecla();  //Lee el valor del teclado y espera hasta que alguna tecla se pulse
      if(c!='#'){
         //Muestra el digito presionado en el LCD
         lcd_gotoxy(25+i,4);
         lcd_putc(c);
         //Almacena el dato presionado en la variable String
         str[i]=c;
      }
      else{i=nd;} //Si se presiona * sale del For
   }
   val = atol(str); //Convierte el String en un valor numerico
}
```

```
/*=============================================================================*/
/*=====================    FUNCION DEL PRINCIPAL    ======================*/
/*=============================================================================*/
void main()
{

   port_b_pullups(true); //inicializa pull ups puerto b
   set_tris_a(0b00000011);          //Pongo el RA0 como entrada
   set_tris_c(0);
   set_tris_d(0);
   setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM a 1kHz
   setup_ccp1(ccp_pwm);                //Configurar modulo CCP1 en modo PWM
   set_pwm1_duty(0);                   //Dejo en cero la salida PWM

   setup_adc_ports(AN0);              //Configurar ADC (Lectura de temperatura)
   setup_adc(adc_clock_internal);      //Reloj interno para la conversion analoga digital)
   set_adc_channel(0);                 //Seleccionar Canal 0 para sensor de Temperatura
   LCD_INIT();                         //Inicializo el LCD
   kbd_init();   //inicializa teclado
   LCD_PUTC("\f");                     //Limpio el LCD
      lcd_gotoxy(1,1);
      LCD_PUTC("VELOCIDAD MOTOR");
      lcd_gotoxy(5,2);
      LCD_PUTC("%");
      lcd_gotoxy(21,1);
      LCD_PUTC("PRESIONE'A' PARA TECL");
      lcd_gotoxy(21,2);
      LCD_PUTC("PRESIONE'B' PARA POT");
```

```
  while(1)
  {


     lcd_gotoxy(1,2);
     printf(lcd_putc,"%Lu\r\n",velocidad/10);//Se visualiza en la pantalla la velocidad en porcentaje de 0 a 100%
     set_pwm1_duty(velocidad);//seteamos la velocidad del motor
       //Espera por 10 mili segundos
       delay_ms(10);

        c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
/*=====================    INGRESAR VELOCIDAD POR EL TECLADO    =====================*/
        if(c=='A')
        {
           LCD_PUTC("\f");
           lcd_gotoxy(1,1);
           LCD_PUTC("DIGITE % VELOCIDAD");
           lcd_gotoxy(1,2);
           LCD_PUTC("Velocidad(0-100%)");
           lcd_gotoxy(21,1);
           LCD_PUTC("y presione '#'");
           lcd_gotoxy(21,2);
           LCD_PUTC("Vel:           ");
           R=porcentaje(3); //Llama la funcion para digitar el porcentaje de exitacion
           //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)
           R =(R > 100) ? 1000:R*10;
           LCD_PUTC("\f");
           //Muestra el Porcentaje de Velocidad en pantalla
           lcd_gotoxy(21,2);
           printf(lcd_putc,"Vel: %3.1f",R/10);
```

```
           velocidad=R;
           delay_ms(100);
           LCD_PUTC("\f");
           lcd_gotoxy(1,1);
           LCD_PUTC("VELOCIDAD MOTOR");
           lcd_gotoxy(5,2);
           LCD_PUTC("%");
           lcd_gotoxy(21,1);
           LCD_PUTC("PRESIONE'A' PARA TECL");
           lcd_gotoxy(21,2);
           LCD_PUTC("PRESIONE'B' PARA POT");
        }
/*=============================================================================*/
/*=======================    CONTROL DE VELOCIDAD POR EL POTENCIÓMETRO    =======================*/
        if(c=='B')
        {
         LCD_PUTC("\f");
           lcd_gotoxy(1,1);
           LCD_PUTC("PRESIONE '#'");
           lcd_gotoxy(1,2);
           LCD_PUTC("PARA SALIR");
         WHILE(1){
           c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
           //Lectura del Potenciometro
           set_adc_channel(0);   //Escoge el canal 0 de los puertos analogos
           delay_us(50);
           adc=read_adc();      //Almacena en duty el valor del voltaje del pot
           adc =(adc > 1000) ? 1000:adc;
           lcd_gotoxy(21,1);
           printf(lcd_putc,"%Lu\r\n",adc/10);
           lcd_gotoxy(25,1);
           LCD_PUTC("%");
```

```
            set_pwm1_duty(adc);
            delay_ms(50);
            if(c=='#'){
            LCD_PUTC("\f");
            lcd_gotoxy(1,1);
            LCD_PUTC("VELOCIDAD MOTOR");
            lcd_gotoxy(5,2);
            LCD_PUTC("%");
            lcd_gotoxy(21,1);
            LCD_PUTC("PRESIONE'A' PARA TECL");
            lcd_gotoxy(21,2);
            LCD_PUTC("PRESIONE'B' PARA POT");

            break;
            }
          }

         }
 /*=====================================================================*/

   }
}
```

Figura 12. Programación en CCS C Compiler.

Descripción paso a paso del código:

1. Se realizan las configuraciones iniciales del microcontrolador, estableciendo las fuentes de reloj, fusibles y otros parámetros. También se define la resolución ADC y se configura el reloj a 20 MHz.

2. Se definen macros para acceder a los puertos A, B, C y D, utilizando direcciones de memoria específicas para el PIC 18F4550.

3. Se utiliza la directiva `#USE FAST_IO` para configurar los puertos A, B, C y D en modo rápido de alto rendimiento. Esto es útil para acceder rápidamente a los puertos.

4. Se define la configuración de pines para la pantalla LCD, especificando pines para control de habilitación (ENABLE), selección de registro (RS), lectura/escritura (RW) y los pines de datos (DATA4 a DATA7).

5. Se definen pines para la matriz de teclado 4x4, especificando las filas (row0 a row3) y columnas (col0 a col3) del teclado.

6. Se incluye la librería "Teclado4x4.h", que proporciona funciones para leer el teclado.

7. Se incluyen las librerías estándar `stdlib.h` y `string.h` para utilizar funciones como `printf` y manipular cadenas de caracteres.

8. Se declaran variables como `adc` (para la lectura del potenciómetro), `velocidad` (para la velocidad del motor), `R` (variable auxiliar) y `c` (para almacenar caracteres del teclado).

9. Se definen tres funciones: `tecla` para esperar hasta que se presione una tecla, `tecla_time` para esperar por un tiempo determinado la presión de una tecla, y `porcentaje` para ingresar un valor de porcentaje a través del teclado.

10. En la función `main`, se realizan las configuraciones iniciales, como activar las resistencias pull-up en el puerto B, configurar pines como entrada o salida, establecer la frecuencia PWM, configurar el ADC, inicializar el LCD y el teclado, y mostrar mensajes iniciales en el LCD.

11. En el bucle principal `while(1)`, se lee el valor del potenciómetro y se muestra en el LCD como un porcentaje de velocidad.

12. Si el usuario presiona la tecla 'A', se le permite ingresar un valor de velocidad a través del teclado numérico. El valor ingresado se muestra en el LCD y se establece como la nueva velocidad del motor.

13. Si el usuario presiona la tecla 'B', el programa entra en un bucle que permite al usuario controlar la velocidad del motor utilizando el potenciómetro. Si el usuario presiona '#' durante este modo, el programa regresa al modo de configuración de velocidad.

Paso N.º 4: Compilar: **Según Practica # 0**

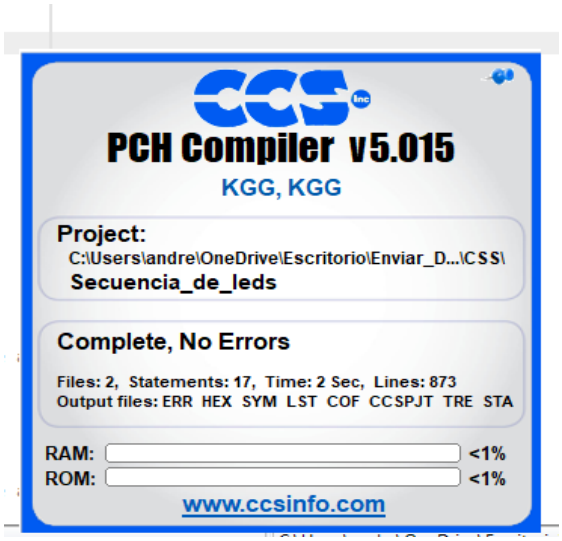


Figura 13. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Práctica # 0**

Figura 14. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

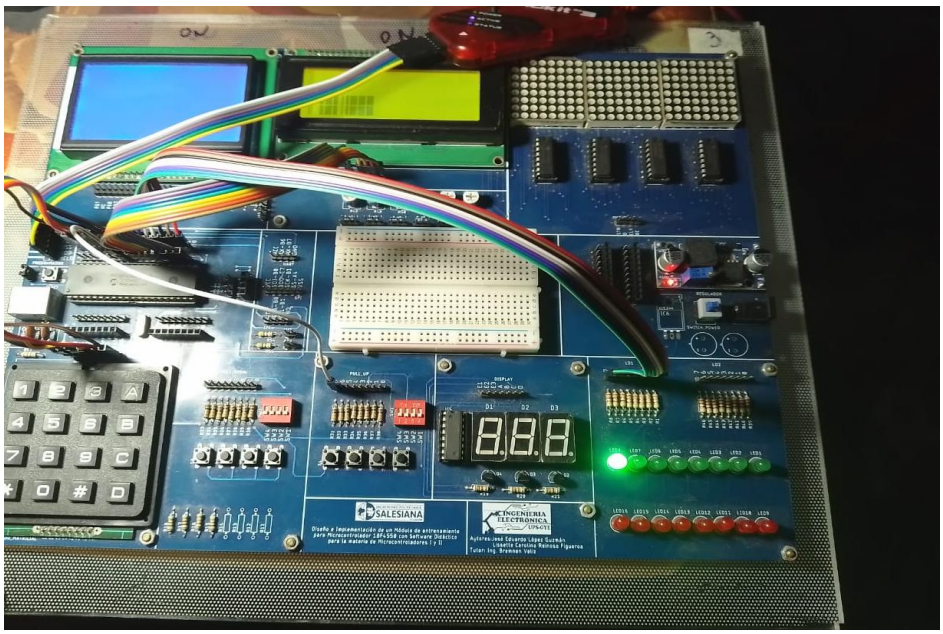


Figura 15. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 10**
NOMBRE PRÁCTICA:	Control PID de una resistencia calefactora.	

1. **Objetivo General.**

   Implementar un programa que realice un control PID.

2. **Objetivos Específicos.**

   Implementar un programa en el PIC que realice el control PID de un sistema que consiste en una resistencia calefactora y una termocupla como sensor, los datos se visualizan en el LCD.

3. **Marco Teórico.**

   Controlador PID.

   El controlador PID (Proporcional-Integral-Derivativo) es un método ampliamente utilizado para el control de sistemas dinámicos, y su implementación puede variar según las características específicas del sistema y los componentes utilizados. La implementación de un controlador PID en un microcontrolador PIC, como el PIC18F4550, requerirá conocimientos en control automático y programación de microcontroladores. (Nise, 2014)

   Resistencias Calefactoras:

   Las resistencias calefactoras se utilizan para generar calor en aplicaciones como sistemas de calefacción, secado de materiales, procesos de soldadura, entre otros. La teoría básica involucra la Ley de Joule, que establece que la potencia disipada (calor) en una resistencia es directamente proporcional al cuadrado de la corriente que la atraviesa y a su resistencia (P = I^2 * R). Controlar la temperatura implica ajustar la corriente eléctrica suministrada a la resistencia.

Relés de Estado Sólido (SSR - Solid State Relays):

Los relés de estado sólido son dispositivos electrónicos que permiten controlar la energía eléctrica a través de una carga (como una resistencia calefactora) utilizando componentes de estado sólido, como tiristores o transistores. Los SSR ofrecen ventajas sobre los relés electromagnéticos tradicionales, ya que son más rápidos, duraderos y no generan ruido mecánico. Pueden ser controlados por señales digitales o analógicas. (Rashid, 2018)

4. Descripción.

El sistema comienza midiendo la temperatura actual del proceso mediante una termocupla conectada a una tarjeta interfaz y esta a su vez es conectada a la entrada analógica A0 del PIC 18F4550. Esta temperatura es la variable controlada que se desea mantener cercana a un valor deseado.

El usuario tiene la capacidad de establecer la temperatura deseada (setpoint) utilizando el teclado conectado al microcontrolador. El valor deseado se muestra en la pantalla LCD.

El controlador PID (Proporcional, Integral y Derivativo) calcula la señal de control necesaria para mantener la temperatura medida lo más cerca posible del valor deseado. El control PID utiliza el error entre la temperatura medida y la temperatura deseada para ajustar la señal de control.

La señal de control generada por el controlador PID se utiliza para controlar la potencia suministrada a la resistencia calefactora. Para hacerlo, el PIC 18F4550 utiliza un módulo de PWM integrado para generar una señal PWM. Esta señal PWM controla la intensidad de corriente que pasa a través del MOSFET IRF540N.

El MOSFET IRF540N es un dispositivo semiconductor de potencia que actúa como un interruptor controlado por la señal PWM generada por el PIC. Cuando la señal PWM está en su nivel alto, el MOSFET permite que la corriente fluya hacia la resistencia calefactora, generando calor. Cuando la señal PWM está en su nivel bajo, el MOSFET corta la corriente, deteniendo la generación de calor.

El MOSFET IRF540N es capaz de manejar cargas de alta potencia, pero no es adecuado para conmutar directamente cargas de CA. Por lo tanto, se utiliza un

relé de estado sólido entre el MOSFET y la resistencia calefactora. El relé de estado sólido actúa como un interruptor controlado por el MOSFET, permitiendo o cortando

La alimentación de la resistencia calefactora en función de la señal de control del MOSFET.

La resistencia calefactora recibe la corriente controlada por el relé de estado sólido. La cantidad de calor generada por la resistencia depende de la corriente que fluye a través de ella. Al controlar la señal PWM, el sistema puede ajustar la cantidad de calor generado por la resistencia para mantener la temperatura medida lo más cerca posible del valor deseado.

El sistema sigue este ciclo de medir la temperatura actual, calcular la señal de control a través del controlador PID y ajustar la potencia suministrada a la resistencia calefactora. Esto se hace continuamente para mantener la temperatura cercana al valor deseado incluso en presencia de perturbaciones o cambios en las condiciones del proceso.

El sistema utiliza el controlador PID para ajustar la potencia suministrada a la resistencia calefactora mediante un relé de estado sólido y un MOSFET. Esto permite mantener una temperatura medida lo más cercana posible a un valor deseado establecido por el usuario a través del teclado y mostrado en la pantalla LCD. El control PID garantiza un control preciso y continuo de la temperatura del proceso.

5. Materiales.

- Módulo didáctico.
- Jumpers Conectores.
- Termocupla Tipo K.
- Tarjeta Interfaz para Termocupla AD8495
- Mosfet IRF540N.
- Relé de estado sólido.
- Resistencia Calefactora.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

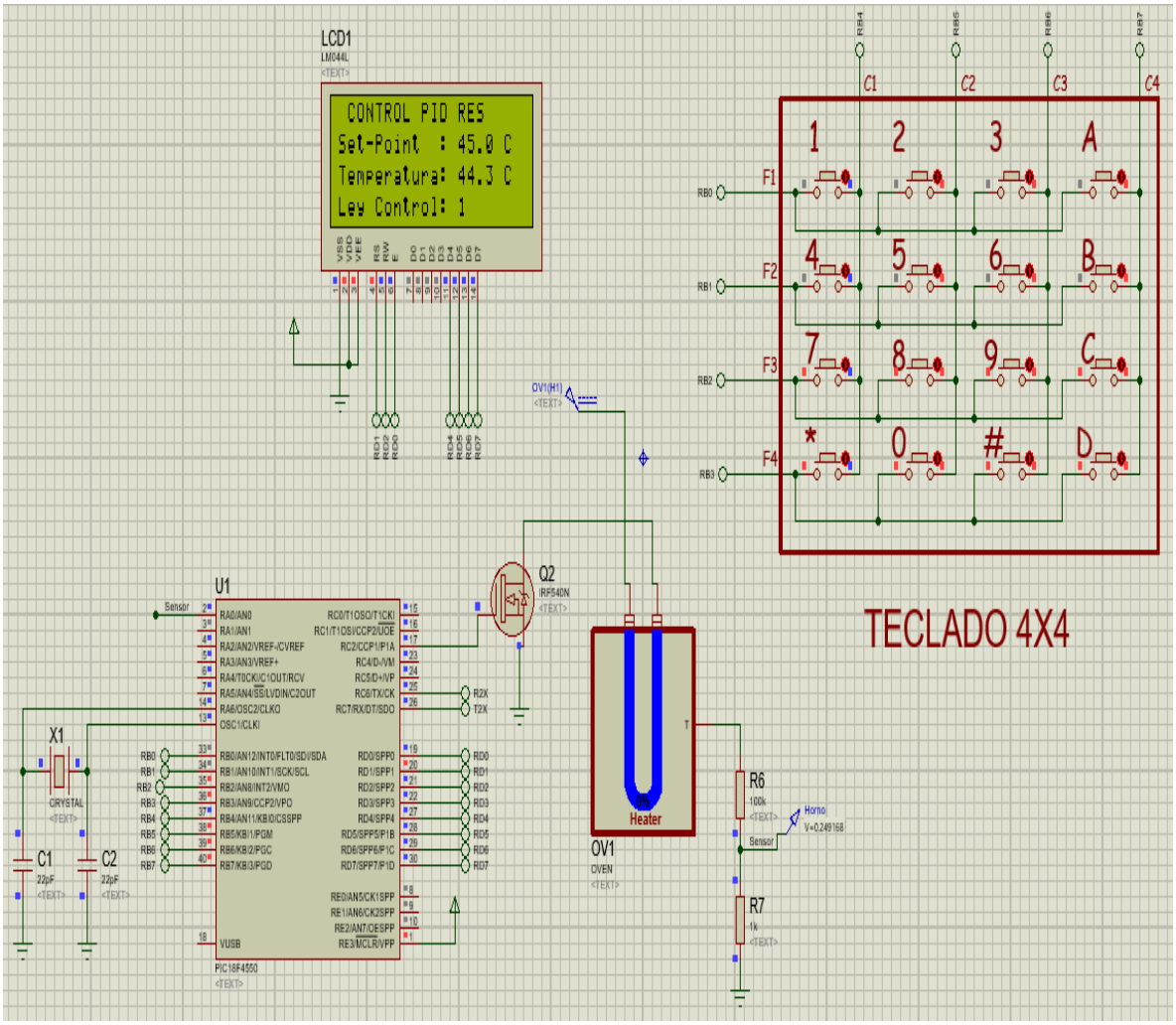


Figura 1. Simulación en Proteus.

Pines del PIC	Pines del Teclado/LCD/AD8495/Mosfet
B0	r1-TECLADO
B1	r2-TECLADO
B2	r3-TECLADO
B3	r4-TECLADO
B4	c1-TECLADO
B5	c2-TECLADO
B6	c3-TECLADO
B7	c4-TECLADO
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
A0	OUT-AD8495
C2	1-GATE MOSFET
Pines del AD8495	

V+	VDD-5V
GND	VSS-0V
+	AMARILLO-TERMOCUPLA
-	ROJO-TERMOCUPLA
Pines del Mosfet	
2-DRAIN MOSFET	NEGATIVO (-) RELÉ
3-SOURCE MOSFET	VSS-0V
Pines del Relé	
POSITIVO (+)	VDD-5V
L1	FASE - 110 Volts
T1	RESISTENCIA-Fase
Pines de la Resistencia	
RESISTENCIA-Neutro	NEUTRO - 110 Volts

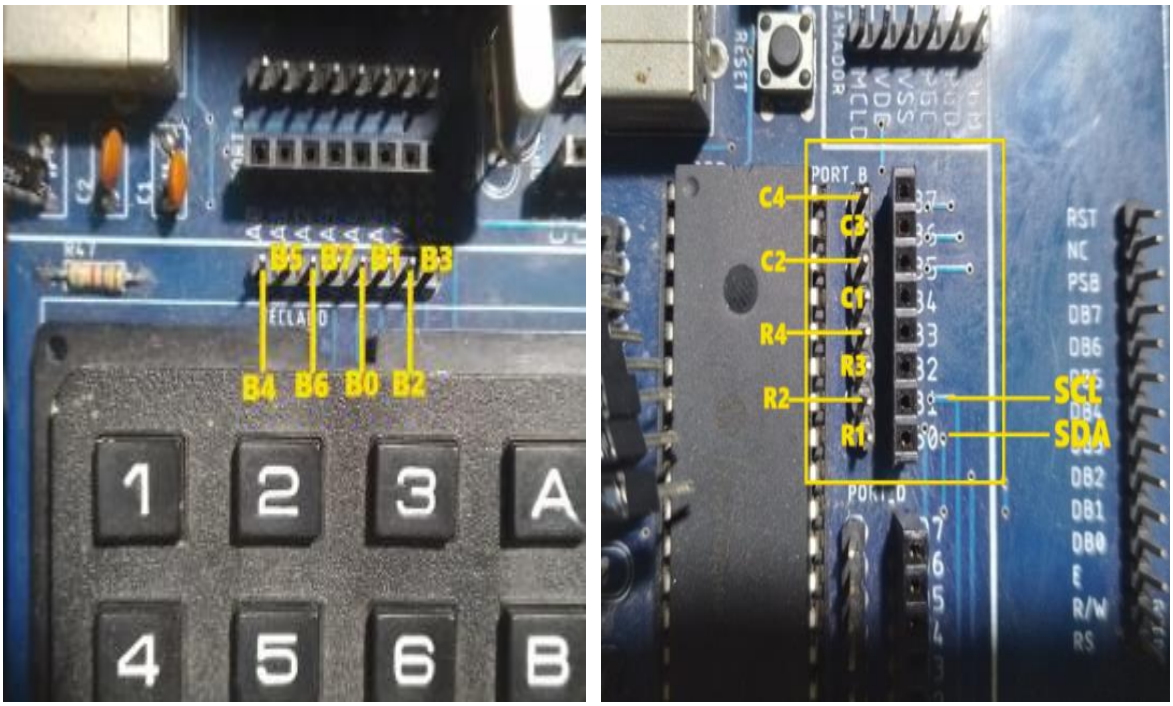


Figura 2. Conexión del Teclado al PIC.

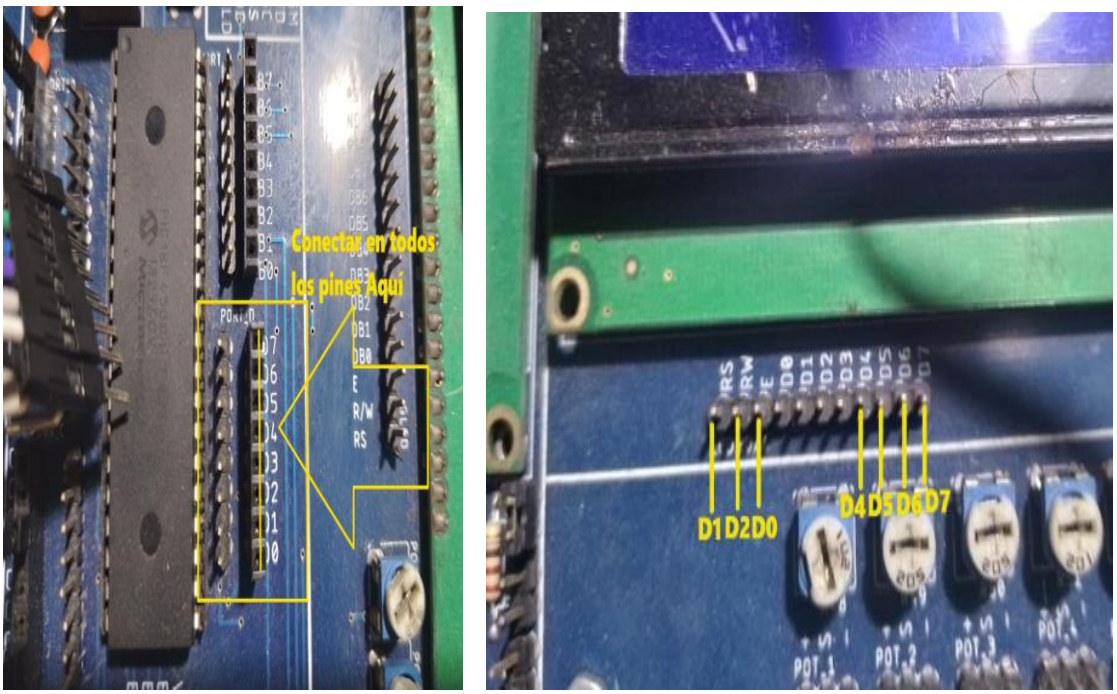


Figura 3. Conexiones del Puerto D a la pantalla LCD.

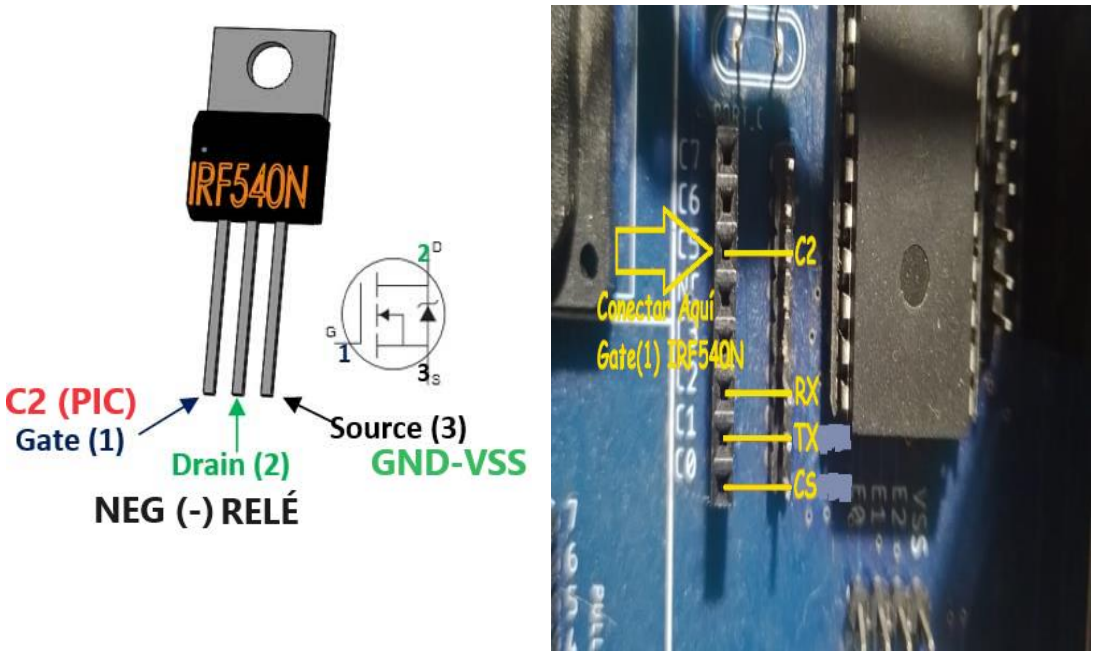


Figura 4. Conexiones del IRF con el PIC.

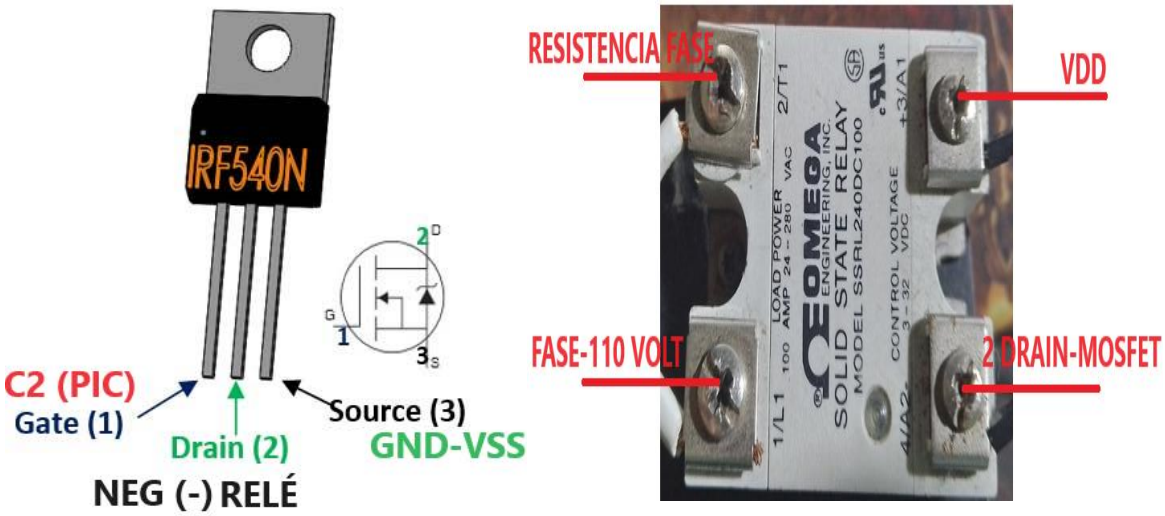


Figura 5. Conexiones del IRF con el Relé.

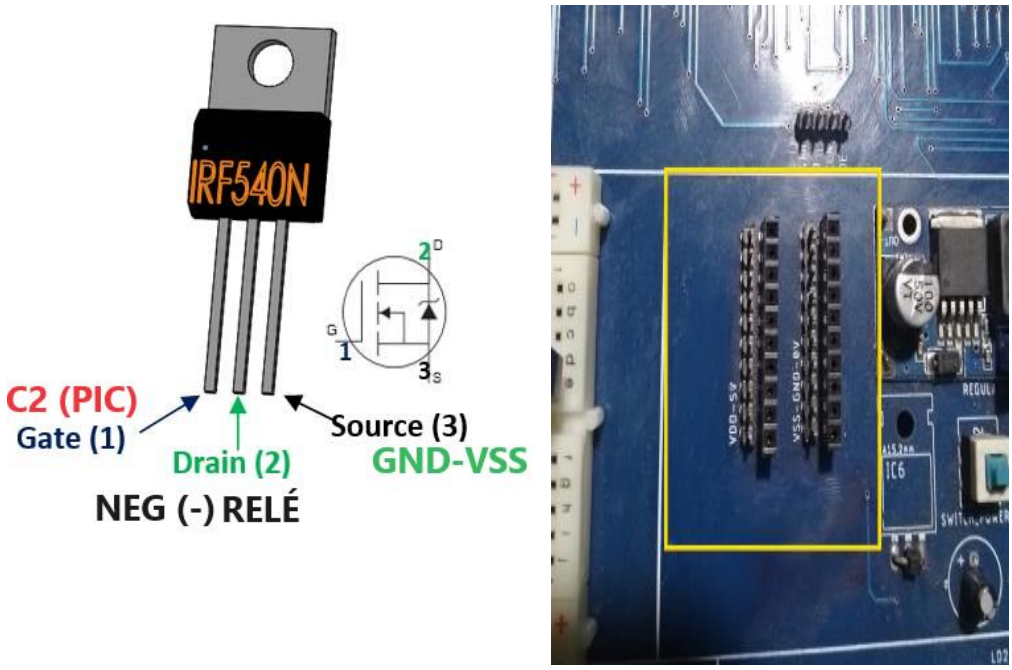


Figura 6. Conexiones del IRF con la alimentación.

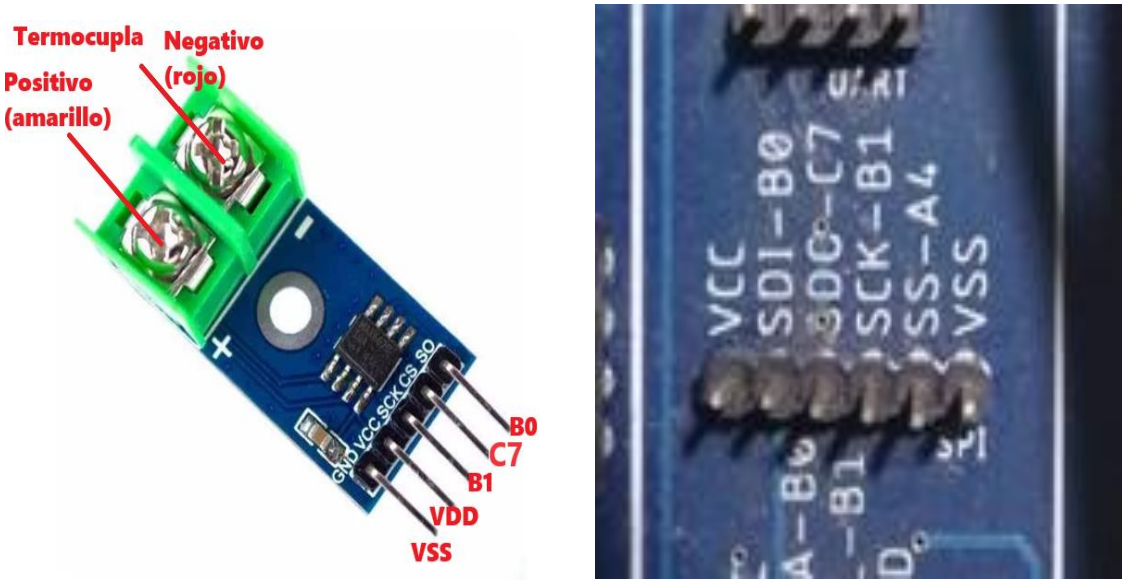


Figura 7. Conexiones del Bloque SPI al Módulo Max6675.

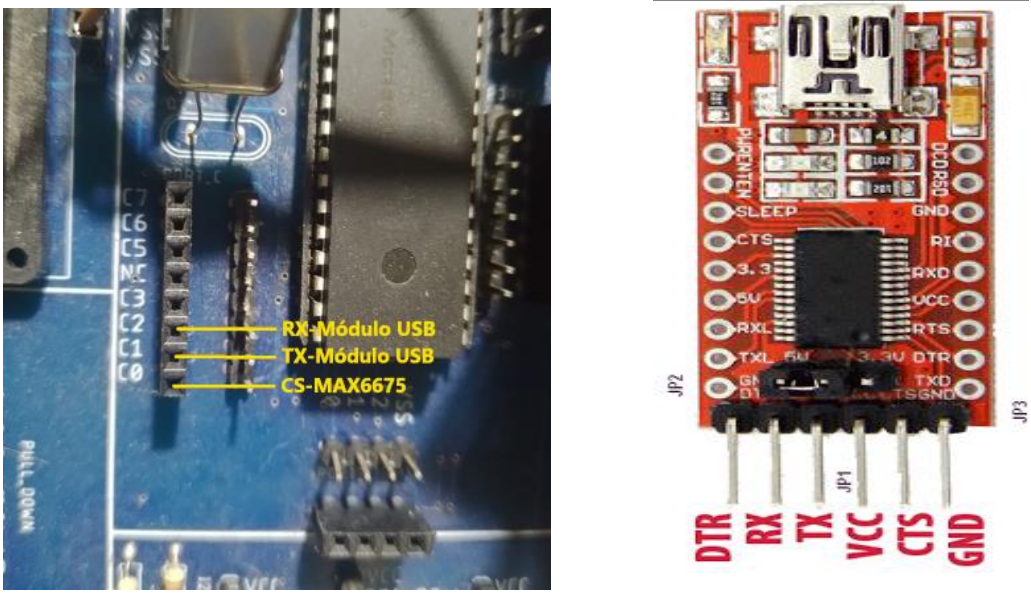


Figura 8. Conexiones del Bloque C al Módulo USB.

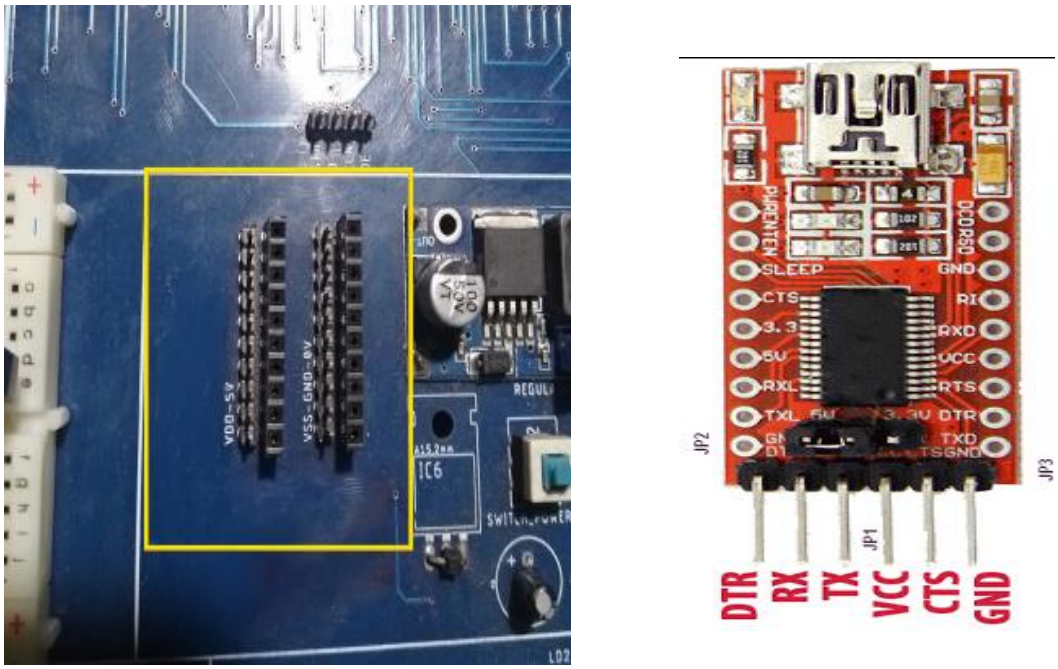


Figura 9. Conexiones de la alimentación al Módulo USB.

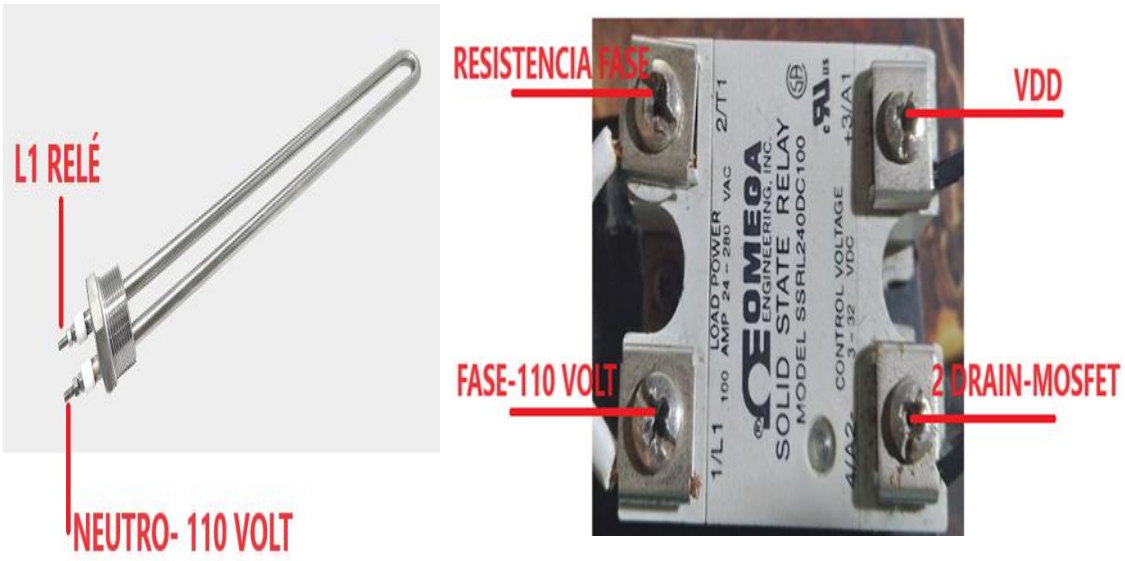


Figura 10. Conexiones del Puerto D a la pantalla LCD.

Paso N. º 2: Ejecutar el ide CCS C Compiler y crear un nuevo proyecto, File » New » Source File. **Según Practica # 0**

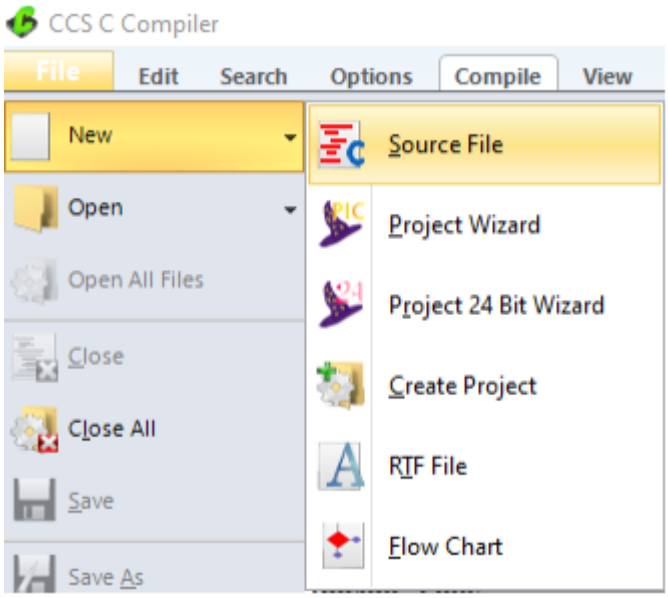


Figura 11. Crear un nuevo proyecto en el ide de CCS C Compiler

Paso N. º 3: Realizar la programación de acuerdo con la siguiente figura.

```
#INCLUDE <18F4550.h>// Declaramos al Pic 18F4550
#DEVICE ADC=10//Seteamos la conversión analógica con una resolución de 10 bits
#FUSES NOWDT//No Watch Dog Timer
#FUSES WDT128//Watch Dog Timer uses 1:128 Postscale
#FUSES NOBROWNOUT//No brownout reset
#FUSES NOLVP//No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES HS//Fusible para usar el oscilador externo de alta velocidad

#use delay(clock=20000000)//Seteamos el reloj externo a 20MHz
#include <lcd.c>// Incluimos la libreria de la pantalla LCD

//Configura direccion de memoria de los puertos A,B,C,D
#BYTE PORTA=0xF80
#BYTE PORTB=0xF81
#BYTE PORTC=0xF82
#BYTE PORTD= 0xF83
////////////////////////////////////////////////////
# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento
# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento
# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento
# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento
#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos

////////Definimos los pines de la pantalla LCD///////
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////Definimos los pines de la pantalla LCD///////
```

```
#define row0 PIN_B0                          //Filas del teclado colocar resistencia pullup
#define row1 PIN_B1                          //Filas del teclado colocar resistencia pullup
#define row2 PIN_B2                          //Filas del teclado colocar resistencia pullup
#define row3 PIN_B3                          //Filas del teclado colocar resistencia pullup
#define col0 PIN_B4                          //Columnas del teclado
#define col1 PIN_B5                          //Columnas del teclado
#define col2 PIN_B6                          //Columnas del teclado
#define col3 PIN_B7                          //Columnas del teclado
#include "Teclado4x4.h"                      //Librería de teclado
#include <stdlib.h>//libreria para usar printf
#include <string.h>//Libreria para convertir lo ingresado por el teclado a float

unsigned int16 adc,control=0;//Lectura de la entrada analógica y variable de control iniciada en 0
float R=450;//Referencia de 45 °C por defecto
float yM=0,e=0.0,e_1=0.0,e_2=0.0,u=0.0,u_1=0.0,T=0.1; // variables del pid
float kp,ti,td,q0,q1,q2; // variables del pid
float k=0.535,tao=8.43; // variables del pid
float TsMA,Wn,P1,P2; // variables del pid
char c;// variable para guardar el caracter que se ingresa por el teclado
int i=0; //Posicion del dato recibido a mostrar en el LCD
```

```
void main()
{
   LCD_INIT();   //Inicializo el LCD
   delay_ms(50);       //permite estabilizar al oscilador
   port_b_pullups (true);  //Utiliza las resistencias PULL UP internas del puerto B
   set_tris_a(0b00000011);          //Pongo el RA0 como entrada
   set_tris_c(0);
   set_tris_d(0);
   setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM a 1kHz
   setup_ccp1(ccp_pwm);                //Configurar modulo CCP1 en modo PWM
   set_pwm1_duty(0);                   //Dejo en cero la salida PWM
   setup_adc_ports(AN0);              //Configurar ADC (Lectura de temperatura)
   setup_adc(ADC_CLOCK_INTERNAL);      //Reloj interno para la conversion analoga digital)
   kbd_init();   //inicializa teclado

   //*************************************************************************//
   //************  DISEÑO POR ASIGNACIÓN DE 2 POLOS REALES   ***************//
   //*************************************************************************//

   TsMA=30;                     //Tiempo deseado en Lazo Cerrado
   Wn=3/(TsMA);                //Frecuencia natural del sistema

   //Ubicación de 2 Polos reales
   P1=2*Wn;
   P2=Wn*Wn;

   kp=(P1*tao-1)/k;         //Calculo de Kc
   ti=(k*kp)/(P2*tao);      //Calculo de ti

  // Calculo del controlador PID digital
   q0=kp*(1+T/(2*ti)+td/T);
   q1=-kp*(1-T/(2*ti)+(2*td)/T);
   q2=(kp*td)/T;
```

```
   for(;;)
{

  set_adc_channel(0);                  //Seleccionar Canal 0 para sensor de Temperatura
  adc=read_adc(); //Leer ADC
  delay_us(10); // Delay for stabilization
  voltSalida=adc*((float)vref/(float)1024);//Conversion de bits a voltaje ******DAC
        ym=((voltSalida+0.0075625)/0.00578125)*10.00;//Formula cáculada para el uso de la
                                                     //trajeta interfaz
        //Llama la funcion del controlador PID
        PID();
        //tiempo de muestreo
        delay_ms(73);

           lcd_gotoxy(2,1);
           lcd_putc("CONTROL PID RES");
           lcd_gotoxy(1,2);
           printf(lcd_putc,"Set-Point  : %3.1f C   ",R/10);
           lcd_gotoxy(21+i,1);
           printf(lcd_putc,"Temperatura: %3.1f C",yM/10);
           lcd_gotoxy(21,2);
           printf(lcd_putc,"Ley Control: %ld   ",control);

        c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
        if(c=='D')
        {
           LCD_PUTC("\f");
           lcd_gotoxy(1,2);
           LCD_PUTC("Referencia(0-100):");
           lcd_gotoxy(21,1);
           LCD_PUTC("y presione *");
           lcd_gotoxy(21,2);
           LCD_PUTC("SP:                    ");
           R=escalon(3); //Llama la funcion para digitar el escalon de exitacion
           //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)
           R =(R > 100) ? 1000:R*10;
```

```
            //Muestra el SETPOINT en pantalla
            lcd_gotoxy(21,2);
            printf(lcd_putc,"SP: %3.1f          ",R/10);
            delay_ms(2000);
            LCD_PUTC("\f");
          }
 }

}
```

Figura 12. Programación en CCS C Compiler.

Descripción paso a paso del código:

Configuración de Fusibles y Periféricos: El código comienza configurando los fusibles del microcontrolador para definir aspectos como el oscilador, el watchdog timer, la protección del código, etc. Luego, se incluyen las bibliotecas necesarias, como la librería para el LCD.

Definición de Pines y Puertos: Se definen los pines utilizados para conectar el microcontrolador con el teclado, la pantalla LCD y otros periféricos. Esto incluye la asignación de pines para la entrada analógica, los puertos A, B, C y D, y la configuración de puertos de entrada y salida.

Inicialización del Control PID: Se inicializan todas las variables necesarias para el control PID. Esto incluye valores como la referencia deseada (R), las constantes del controlador (kp, ti, td), las variables de error (e, e_1, e_2), y las variables de control (u, u_1).

Funciones de Lectura de Teclado: Se definen funciones para leer las teclas presionadas en el teclado. Esto se utiliza para configurar la referencia (R) y realizar otras interacciones con el programa.

Función de Control PID (PID): Se implementa la función de control PID, que calcula la señal de control (u) en función del error actual (e) y los valores de las constantes PID (kp, ti, td). La señal de control se ajusta para evitar saturación y se aplica al controlador PWM que controla la resistencia calefactora.

Función para Establecer el Setpoint (escalon): Se define una función que permite al usuario configurar el valor de referencia (setpoint) deseado para la temperatura. Esto se hace ingresando el valor por el teclado.

Función Principal (main): En la función principal, el programa entra en un bucle

infinito. En cada iteración del bucle, se realiza lo siguiente:

a. Se lee la temperatura actual del sensor conectado al puerto analógico.

b. Se calcula la señal de control utilizando la función de control PID.

c. Se muestra en la pantalla LCD la temperatura actual, el valor de referencia (setpoint) y la señal de control.

d. Se verifica si se ha presionado una tecla en el teclado. Si se presiona la tecla "D", el usuario puede ingresar un nuevo valor de referencia (setpoint) desde el teclado.

Control de la Resistencia Calefactora: La señal de control calculada se utiliza para controlar la resistencia calefactora mediante la modulación por ancho de pulso (PWM). Esto permite variar la potencia suministrada a la resistencia para mantener la temperatura cerca del valor deseado.

Paso N.º 4: Compilar: **Según Practica # 0**

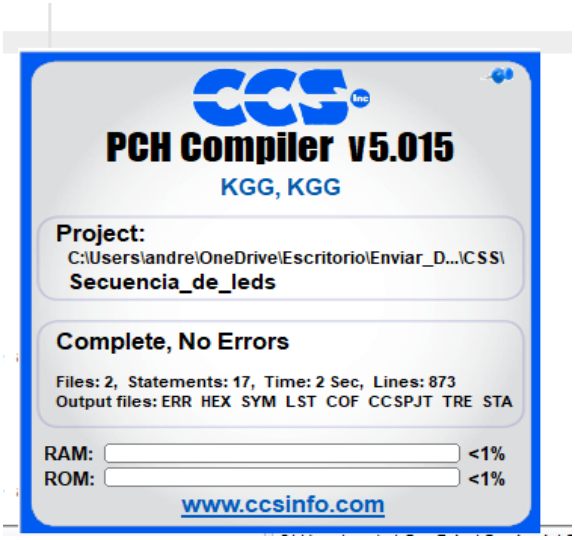


Figura 13. Compilación Realizada.

Paso N. º 5: Cargar en el PIC el archivo .hex creado al momento de compilar, usando el software PicKit3 y el dispositivo Pickit3. **Según Practice # 0**

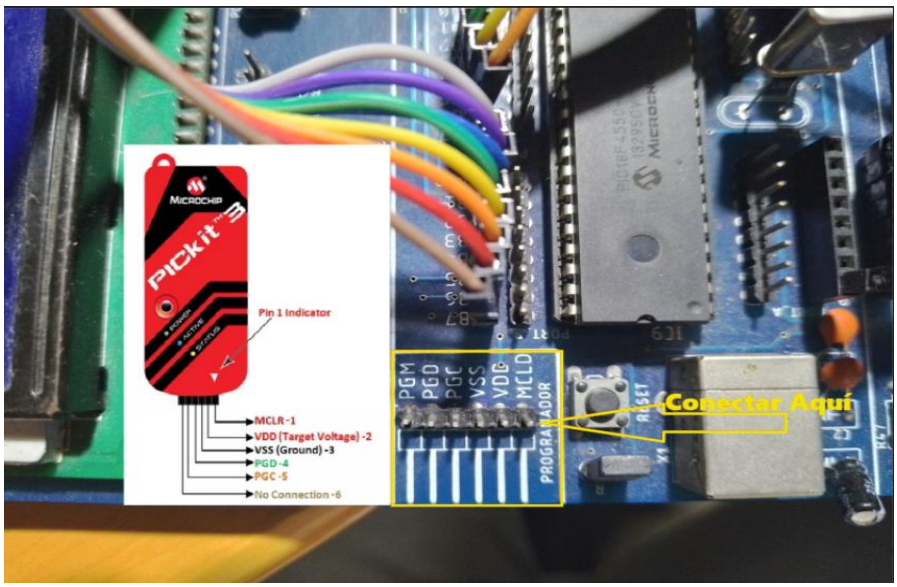


Figura 14. Cargando el Programa al PIC.

Paso N. º 6: Probamos el funcionamiento en el módulo.

Resultado.

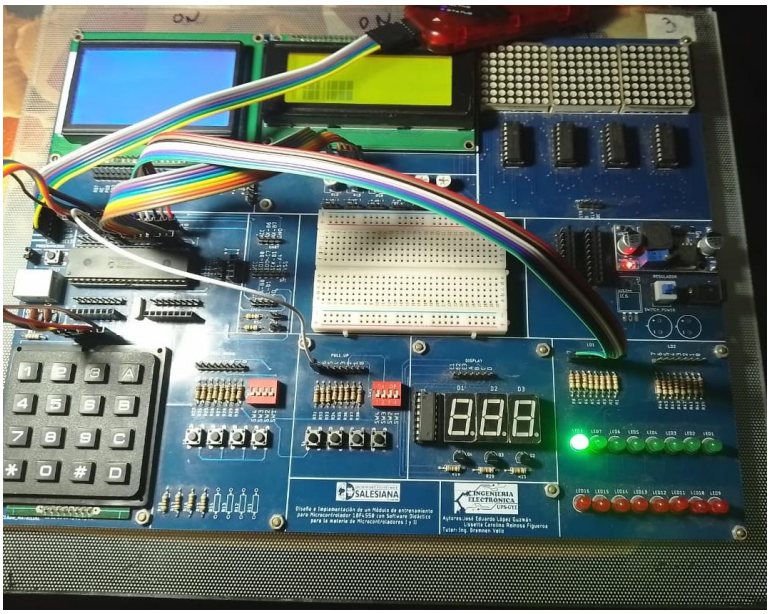


Figura 15. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 11**
NOMBRE PRÁCTICA:	SCADA en Labview para el control PID de una resistencia calefactora.	

1. **Objetivo.**

   Implementar una interfaz en Labview para la práctica 11(Módulo PID).

2. **Objetivos Específicos.**

2. Implementar un programa en Labview en el cual se lean los datos de temperatura que llegan del PIC por medio de comunicación serial.

3. **Marco Teórico.**

   Comunicación Serial (UART):

   La comunicación serial se utiliza para transmitir datos desde el PIC18F4550 a otro dispositivo, en este caso, un software de LabVIEW que se ejecuta en una computadora. El código configura el UART para transmitir datos a 9600 baudios.

   Interfaz con Pantalla LCD: El código utiliza una pantalla LCD alfanumérica para mostrar la temperatura medida en grados Celsius. Se configuran los pines de comunicación con el LCD.

   Formateo y Visualización de Datos:

   El código realiza cálculos para convertir la temperatura medida en una forma legible y la muestra en el LCD. Se utilizan variables para descomponer la temperatura en centenas, decenas y unidades antes de mostrarla en el LCD.

   Bucle Infinito:

   El programa se ejecuta en un bucle infinito, lo que significa que continuará leyendo, formateando y mostrando la temperatura de manera continua. Esto es útil para aplicaciones de monitoreo continuo.

Comunicación con LabVIEW:

La comunicación serial se utiliza para transmitir datos de temperatura al software LabVIEW que se ejecuta en una computadora. LabVIEW puede procesar y visualizar estos datos de temperatura. (Rashid, 2018)

4. Descripción.

El PIC18F4550 está configurado para leer una señal de un sensor de temperatura de tipo K (termocupla) a través del chip MAX6675. La termocupla mide la temperatura y convierte esta medición en una señal digital que se comunica con el PIC a través de la interfaz SPI.

La temperatura medida se convierte en grados Celsius y se muestra en un display LCD alfanumérico. La temperatura se muestra en tres dígitos (centenas, decenas y unidades) junto con el mensaje "TERMOCUPLA K". El valor mostrado en el LCD se actualiza continuamente.

Además de mostrarse en el LCD, el valor de temperatura se envía a través de la comunicación serie (UART) a una velocidad de 9600 baudios. Esto permite que los datos de temperatura se transmitan a una interfaz en Labview.

El código se ejecuta en un bucle infinito, lo que significa que el PIC continuará leyendo, mostrando y transmitiendo la temperatura de manera continua sin detenerse.

Este código permite al PIC18F4550 leer la temperatura de una termocupla tipo K, mostrarla en un LCD y transmitirla a través de comunicación serie. Esto facilita la monitorización y el registro continuo de la temperatura medida.

5. Materiales.

- Módulo didáctico PID.
- Módulo didáctico.
- Jumpers conectores.
- Termocupla tipo K.
- Módulo Max6675.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler
- Software LabView

6. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

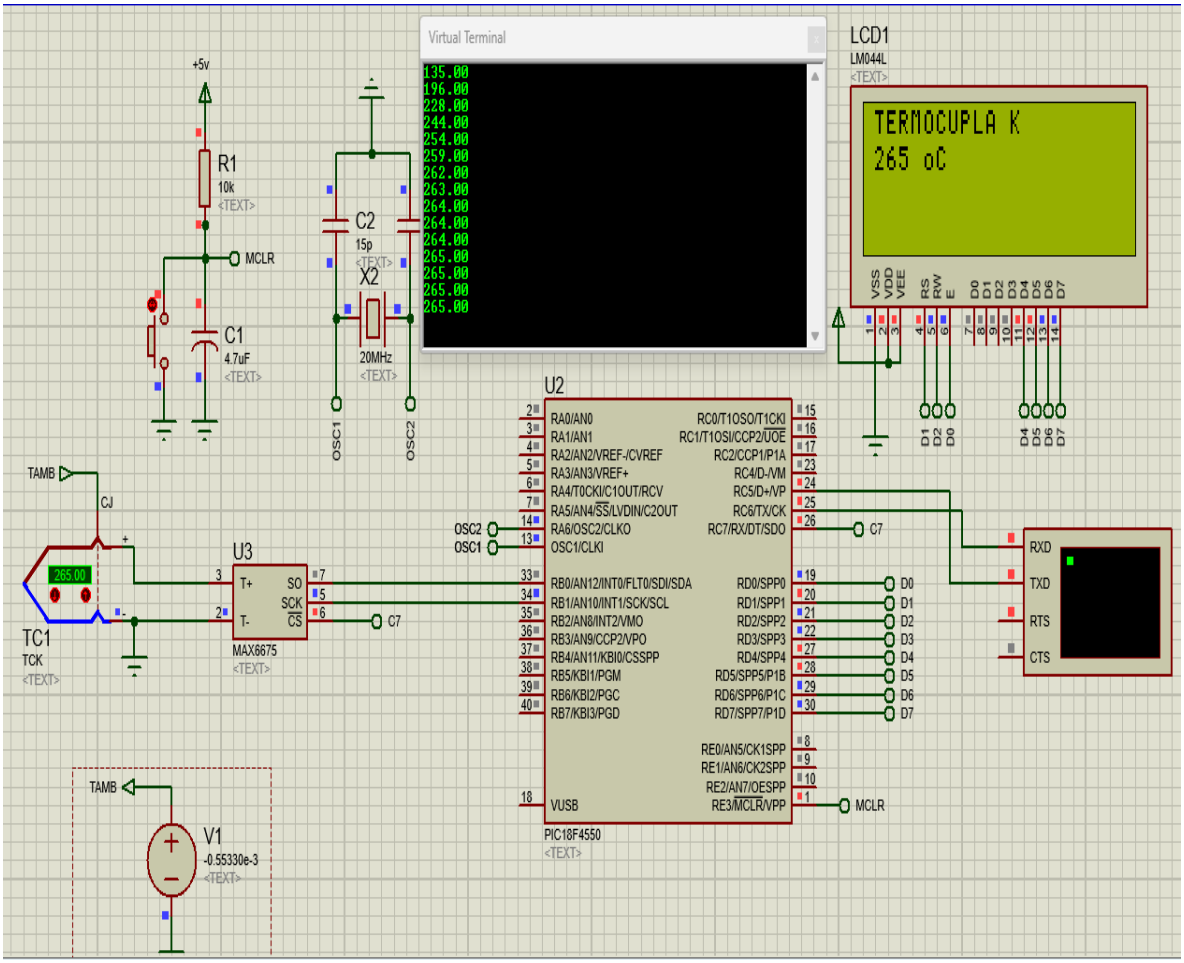


Figura 1. Simulación en Proteus.

Pines del PIC	Pines de la Pantalla/Max6675
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
B0	SO-MAX6675
B1	SCK-MAX6675
C7	CS-MAX6675
C6	TX-MÓDULO USB
C5	RX-MÓDULO USB
Pines del Max6675	
VCC-MAX6675	VDD-5V
GND-MAX6675	VSS-0V

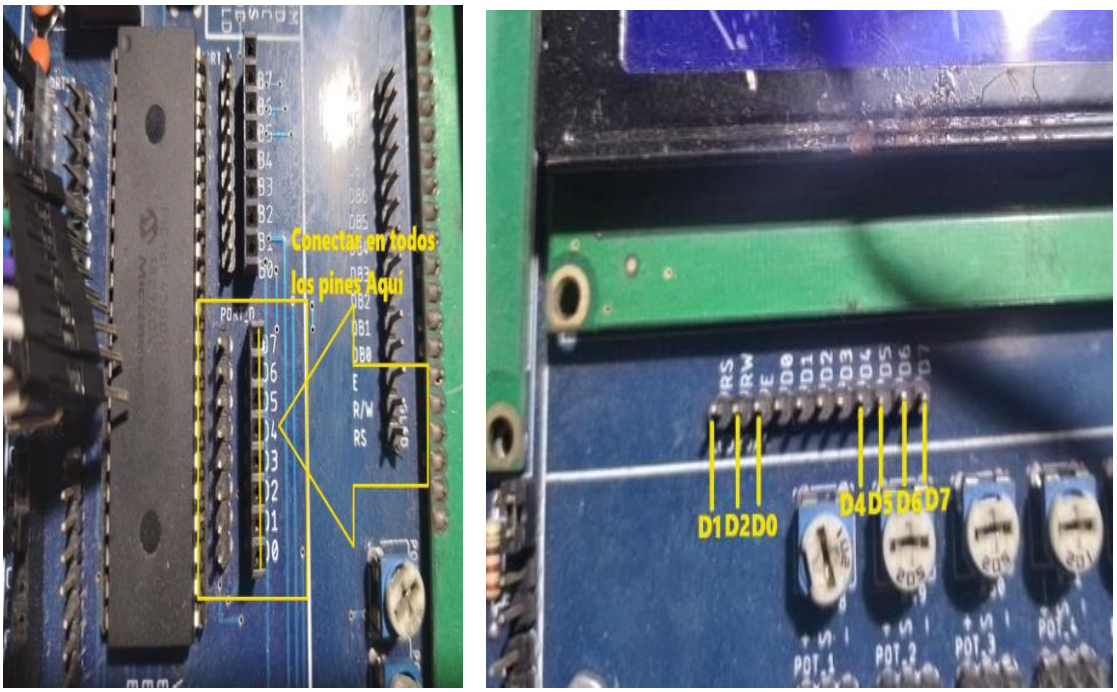


Figura 2. Conexiones del Puerto D a la pantalla LCD.

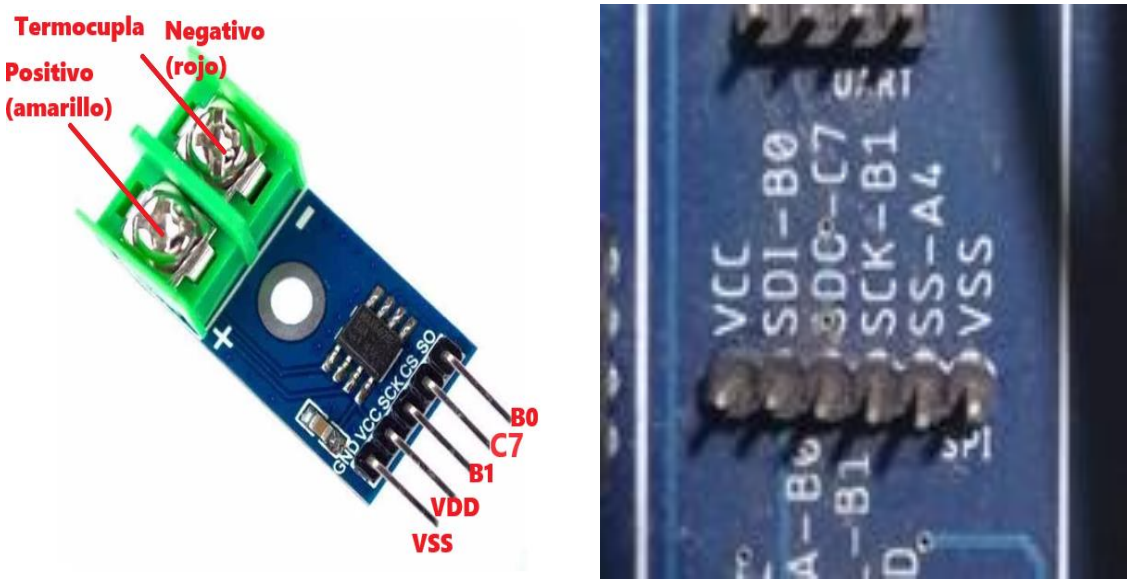


Figura 3. Conexiones del Bloque SPI al Módulo Max6675.

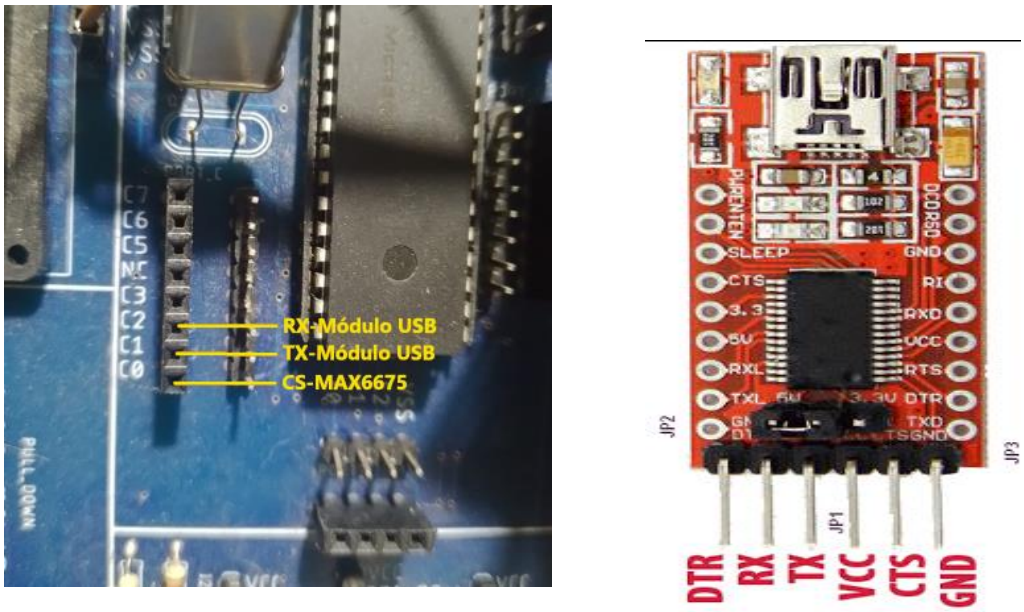


Figura 4. Conexiones del Bloque C al Módulo USB.

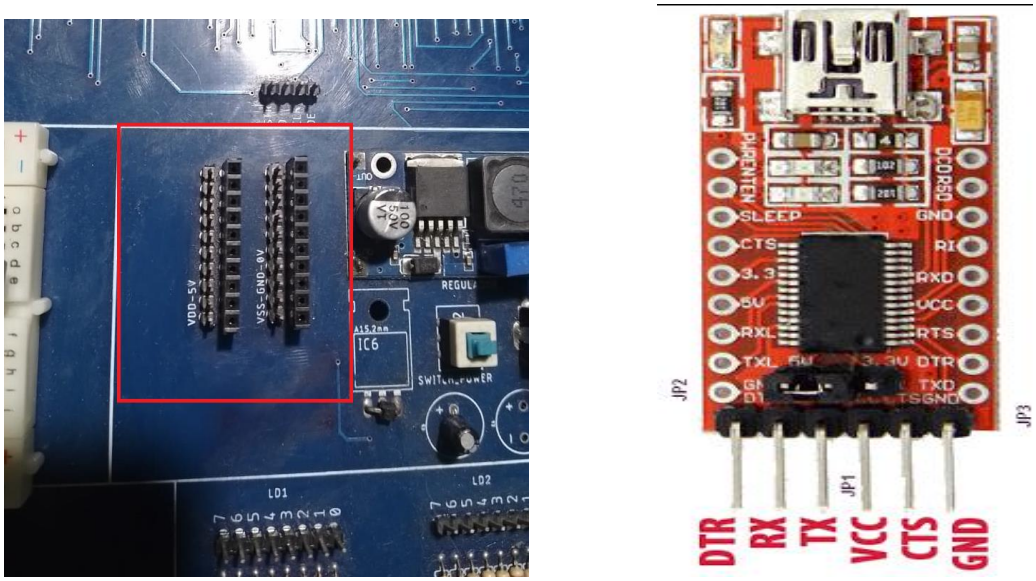


Figura 5. Conexiones de la alimentación al Módulo USB.

Paso N. º 2: En el entorno de programación de Labview realizar la siguiente programación gráfica.

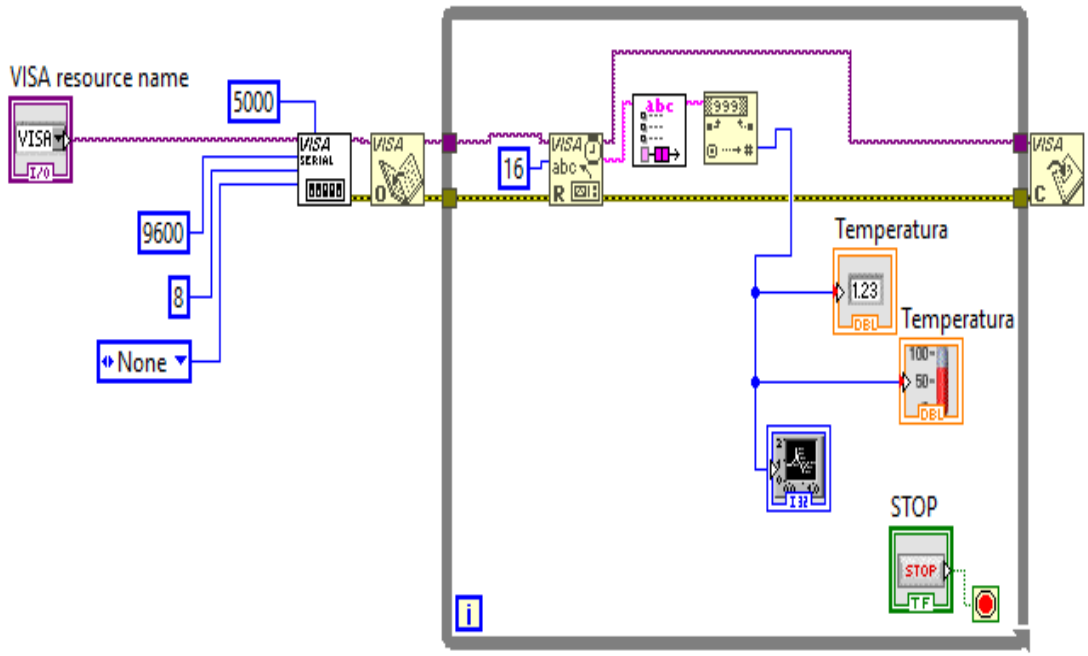


Figura 6. Programación de la interfaz en Labview.

Paso N. º 3: Realizar la programación en CCS Compiler de acuerdo con la siguiente figura.

```
#include <18f4550.h>// Incluimos el fichero del Microcontrolador PIC18F4550
#fuses INTRC_IO //Fusible para usar el oscilador interno
#FUSES NOWDT//No Watch Dog Timer
#FUSES WDT128//Watch Dog Timer uses 1:128 Postscale
#FUSES HS//Fusible para usar el oscilador externo de alta velocidad
#FUSES NOPROTECT //Fusible para desactivar la protección del código
#FUSES CPUDIV1 // Fusible para activar el reloj divisor 1
#FUSES PLL1 //Fusible para multiplicar el oscilador interno por un factor de 4

#use delay(clock=20000000)   // Definimos la Frecuencia del oscilador de PIC(20MHz)
#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c5, bits=8)
#include "MAX6675_Lib.c"     // Incluimos la librería MAX6675
#include <lcd.c> // incluimos la libreria del LCD
```

```
//Declaramos los pines a usar por la pantalla LCD
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////////////////////////////////////////////

int16 Tm;   // Definimos las variable de temperatura del sensor como flotante

int16 Unidad, Decena, Centena, Centena1; // Definimos las variables como enteros de 16-Bit
 char msg[32];
void main() {                            // Inicia la función principal

   set_tris_d(0x00);                     // Seteamos el puerto D como salida

   lcd_init();

   do{                                   // Inicia al bucle do ... While infinito

      Tm = read_termocupla_k();                  // Lectura de la termocupla tipo K

      Tm = Tm - 35;

      Centena = Tm/100;                          // Captura el dígito de las centenas

      Centena1 = Tm - Centena*100;               // Captura el resto de la división anterior

      Decena = Centena1/10;                      // Captura el dígito de las decenas

      Unidad = Centena1 - Decena*10;             // Captura el dígito de las unidades

    LCD_PUTC("TERMOCUPLA K");
     lcd_gotoxy(1,2);
     printf(lcd_putc,"%ld",Centena); // Visualiza el dígito de las centenas en la pantalla, posición (1,2)
     lcd_gotoxy(2,2);
     printf(lcd_putc,"%ld",Decena);             // Visualiza el dígito de las decenas en la pantalla, posición (2,2)
     lcd_gotoxy(3,2);
     printf(lcd_putc,"%ld",Unidad);             // Visualiza el dígito de las unidades en la pantalla, posición (3,2)

    lcd_gotoxy(5,2);

    LCD_PUTC("o");             // Imprime texto en la pantalla

    lcd_gotoxy(6,2);

    LCD_PUTC("C");             // Imprime texto en la pantalla /

    delay_ms(100);                          // Tiempo de espera de 100ms

      sprintf(msg,"%3.2f\r\n",toFloat_TC(read_termocupla_k()));
     printf("%s",msg);

   }while(1);                               // Termina el bucle do ... While infinito
}
```

Figura 7. Programación en CCS Compiler.

Descripción paso a paso del código:

Inclusión de Librerías y Configuración de Fusibles:

Se incluye la librería <18f4550.h> que contiene las definiciones y configuraciones específicas para el microcontrolador PIC18F4550.

Se configuran los fusibles que determinan el funcionamiento del microcontrolador, como el tipo de oscilador, la protección del código y otros parámetros importantes.

Configuración de Oscilador y Comunicación Serie:

Se define la frecuencia del oscilador interno del PIC18F4550 a 20MHz mediante la directiva #use delay.

Se utiliza #use rs232 para configurar la comunicación serie (UART) a una velocidad de 9600 baudios, sin paridad, 8 bits de datos, en los pines C6 (transmisión) y C5 (recepción).

Inclusión de Librería para Sensor de Temperatura MAX6675:

Se incluye la librería "MAX6675_Lib.c", que proporciona las funciones necesarias para comunicarse con el sensor de temperatura tipo K MAX6675.

Inclusión de Librería para Control de Pantalla LCD:

Se incluye la librería <lcd.c>, que permite controlar una pantalla LCD alfanumérica. Se definen los pines utilizados para conectar la pantalla LCD al PIC.

Declaración de Variables:

Se declaran variables para almacenar la temperatura medida (Tm), así como variables para descomponer la temperatura en centenas, decenas y unidades (Unidad, Decena, Centena, Centena1) para mostrarla en el LCD.

Se define un arreglo de caracteres msg para formatear y mostrar la temperatura en la comunicación serie.

Configuración de Puertos:

Se configura el puerto D del PIC como salida para controlar un display o LEDs.

Bucle Principal:

Se entra en un bucle infinito (do...while) que realiza las siguientes acciones de forma continua:

Lee la temperatura del sensor de tipo K MAX6675 mediante read_termocupla_k().

Realiza cálculos para convertir la temperatura leída en grados Celsius.

Descompone la temperatura en centenas, decenas y unidades.

Muestra la temperatura en el LCD junto con un mensaje ("TERMOCUPLA K").

Envía la temperatura formateada a través de la comunicación serie (UART).

Espera 100 ms antes de repetir el ciclo.

Paso N.º 3: Conectar el Pic a la computadora por medio de la interfaz USB y dar play a la interfaz de Labview

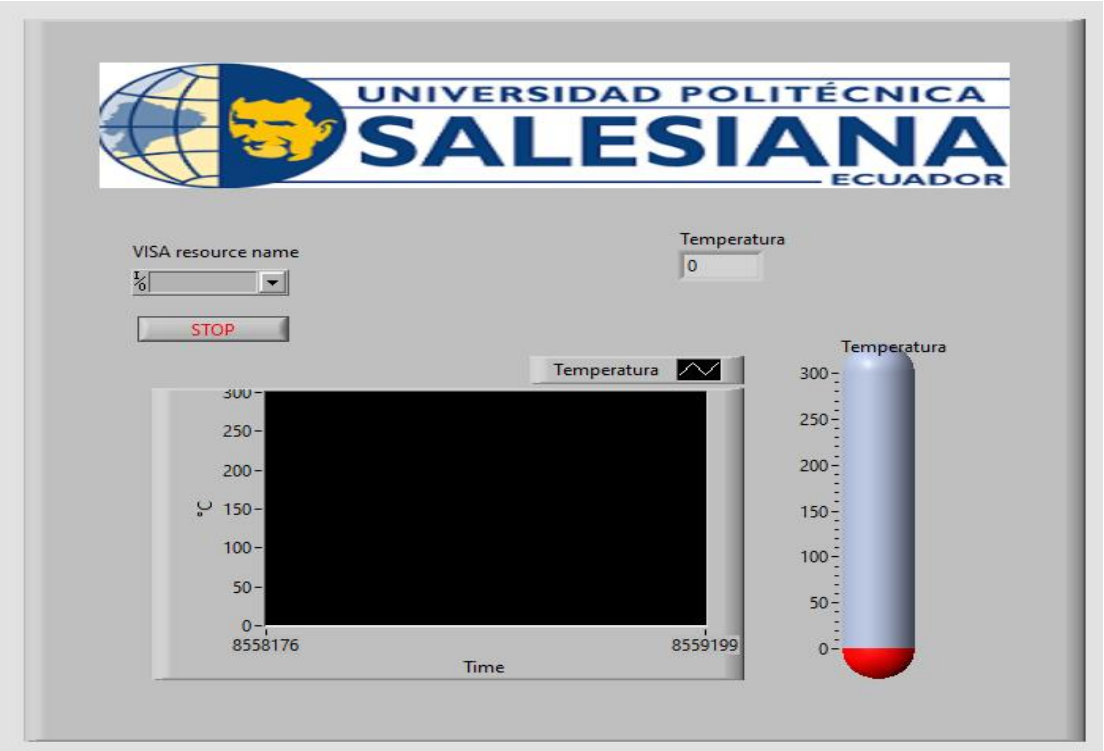


Figura 8. Interfaz de Labview.

Paso N. º 4: Probamos el funcionamiento en el módulo.

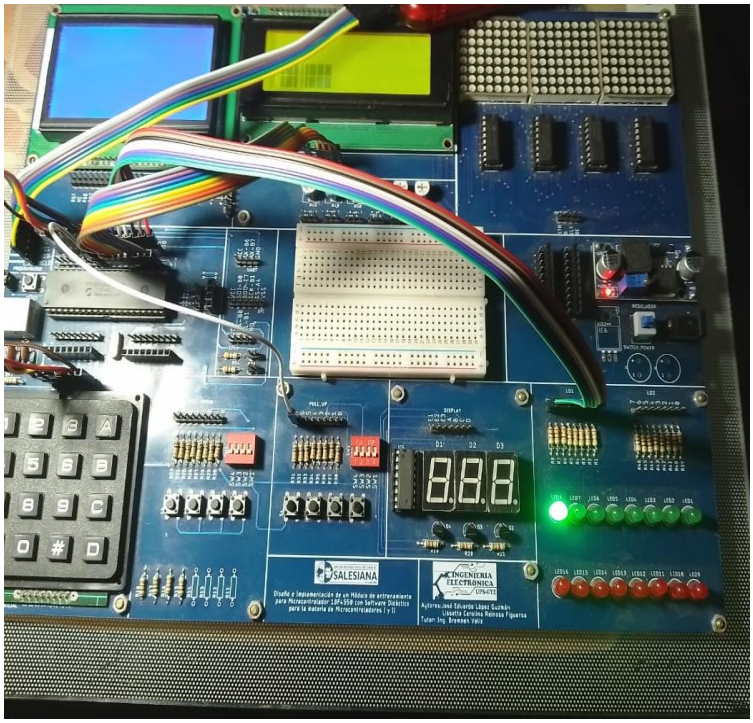


Figura 9. Prueba de funcionamiento.

UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR	MANUAL DE PROCEDIMIENTOS DE PRÁCTICAS	
MATERIA: _____	**ALUMNO:** _____	**PRÁCTICA # 12**
NOMBRE PRÁCTICA:	Interfaz gráfica en Matlab para el control PID de una resistencia calefactora.	

7. Objetivo.

Implementar una interfaz en Matlab para la práctica 11(Módulo PID).

8. Objetivos Específicos.

3. Implementar un programa en Matlab en el cual se lean los datos de temperatura que llegan del PIC por medio de comunicación serial.

9. Marco Teórico.

Interfaz Gráfica de Usuario (GUI):

El código utiliza la función figure para crear una interfaz gráfica que incluye un gráfico para mostrar los datos de temperatura en tiempo real y un botón "Detener" para controlar la adquisición de datos. MATLAB ofrece herramientas para crear interfaces de usuario personalizadas que facilitan la visualización y el control de datos en tiempo real. (MATLAB, 2023)

Comunicación Serial:

El código configura un objeto de puerto serial (SerialP) para establecer una conexión con un dispositivo externo a través del puerto COM3. La comunicación serial se utiliza para recibir datos de temperatura transmitidos por el dispositivo. Los parámetros de configuración, como la velocidad de baudios, el número de bits de datos y la paridad, se establecen de acuerdo con la configuración del dispositivo externo. (MATLAB, 2023)

Bucle de Adquisición de Datos:

El bucle principal (while(~parar)) se ejecuta continuamente hasta que se presiona el botón "Detener" en la GUI. En cada iteración, se utiliza fread para leer un flujo de bytes del puerto serial. Estos bytes se procesan para extraer la temperatura, que se muestra en el gráfico y en un cuadro de texto en la GUI. La pausa de 0.1 segundos (pause(dt)) controla la velocidad de actualización del gráfico. (MATLAB, 2023)

Actualización del Gráfico:

El gráfico en el GUI se actualiza en tiempo real con el tiempo en el eje X y la temperatura en el eje Y. Se controla el rango de visualización del gráfico para permitir un seguimiento continuo de los datos. Cuando se alcanzan ciertos límites, se ajusta automáticamente el rango visualizado en el gráfico. (MATLAB, 2023)

Finalización de la Interfaz:

Cuando se presiona el botón "Detener" en la GUI, se establece la variable parar en true, lo que provoca la salida del bucle de adquisición de datos. Luego, se cierra y libera el puerto serial. (MATLAB, 2023)

10. Descripción.

El código crea una interfaz de usuario en MATLAB que permite la adquisición y visualización en tiempo real de datos de temperatura transmitidos por un dispositivo PIC a través de comunicación serial. El proceso sería el siguiente:

Ejecutas el código en MATLAB.

Se abre la interfaz gráfica de usuario, que muestra un gráfico vacío y un cuadro de texto para la temperatura.

Conectas el módulo PID que transmite datos de temperatura a través del puerto serial.

Haces clic en el botón "Detener" cuando desees detener la adquisición de datos.

La interfaz muestra en tiempo real los datos de temperatura en el gráfico y en el cuadro de texto.

Puedes ajustar los límites del gráfico en tiempo real si es necesario.

Cuando hayas terminado de monitorear, cierras la interfaz gráfica.

El código está diseñado para funcionar como una herramienta de monitoreo en tiempo real y puede ser útil para proyectos que requieren seguimiento continuo de la temperatura a medida que se generan datos en el dispositivo PIC.

11. Materiales.

- Módulo didáctico PID.
- Módulo didáctico.
- Jumpers conectores.
- Termocupla tipo K.
- Módulo Max6675.
- Programador Pickit3.
- Software PROTEUS
- Software CCS Compiler

12. Desarrollo.

Paso N.º 1: Realizar las conexiones en el módulo de acuerdo con el esquema que se muestra en la figura.

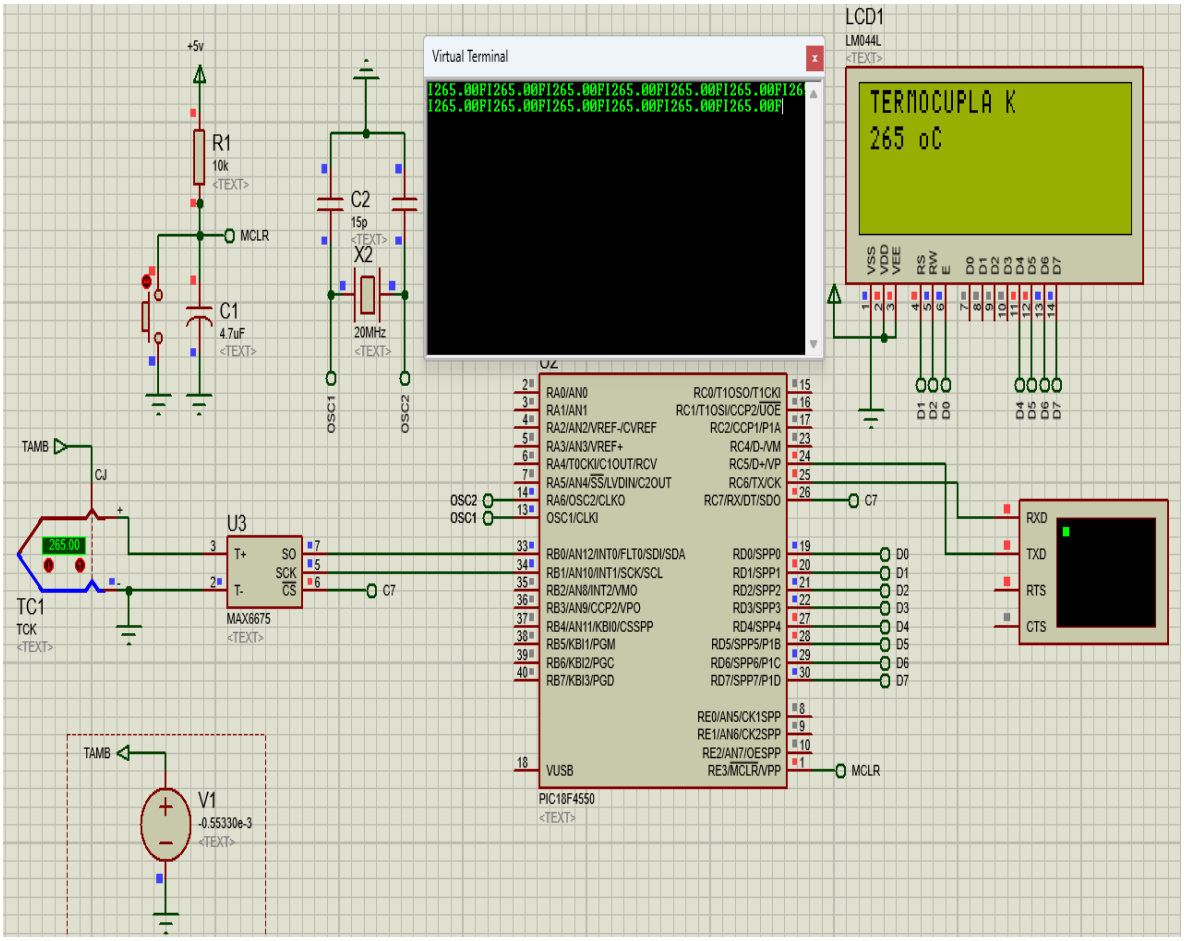


Figura 1. Simulación en Proteus.

Pines del PIC	Pines de la Pantalla/Max6675
D0	E-PANTALLA
D1	RS-PANTALLA
D2	RW-PANTALLA
D4	D4-PANTALLA
D5	D5-PANTALLA
D6	D6-PANTALLA
D7	D7-PANTALLA
B0	SO-MAX6675
B1	SCK-MAX6675
C7	CS-MAX6675
C6	TX-MÓDULO USB
C5	RX-MÓDULO USB
Pines del Max6675	
VCC-MAX6675	VDD-5V
GND-MAX6675	VSS-0V

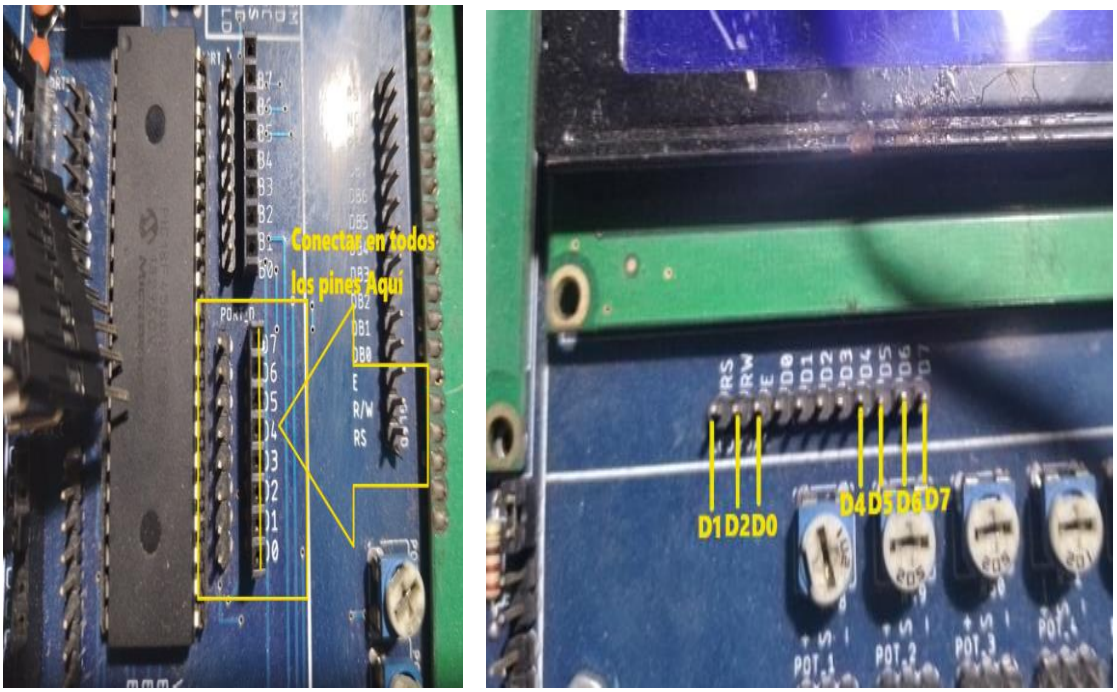


Figura 2. Conexiones del Puerto D a la pantalla LCD.

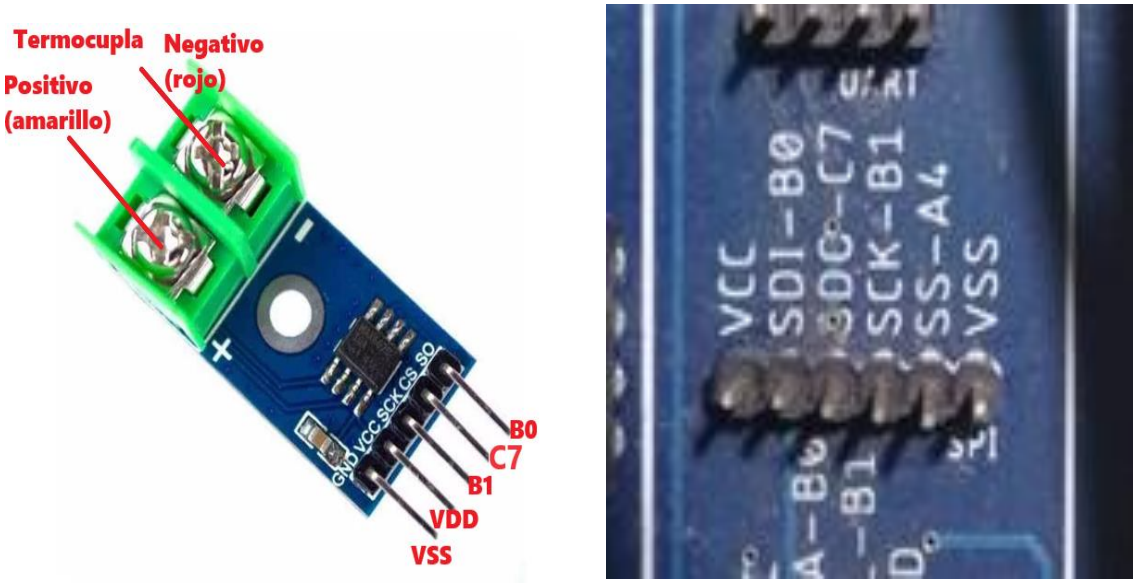


Figura 3. Conexiones del Bloque SPI al Módulo Max6675.

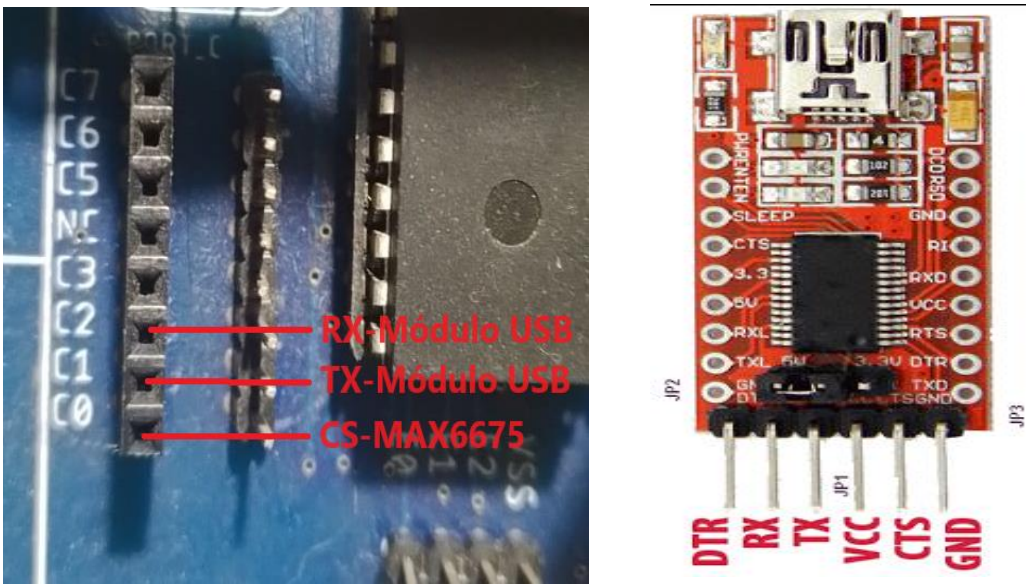


Figura 4. Conexiones del Bloque C al Módulo USB.

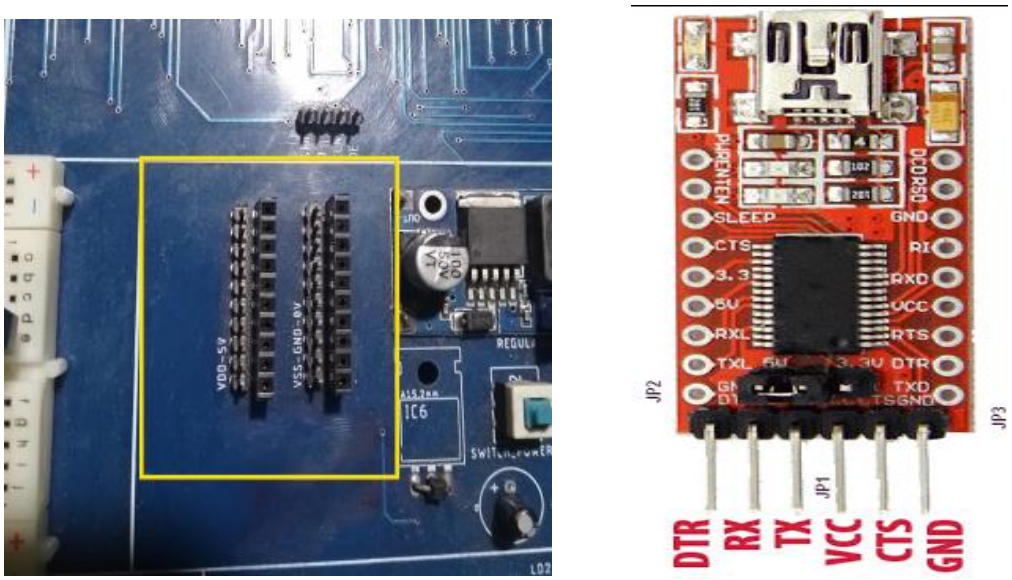


Figura 5. Conexiones de la alimentación al Módulo USB.

Paso N. º 2: En el entorno de programación de Matlab realizar la siguiente programación.

```
%% Monitoreo en Tiempo Real
function varargout=monitoreo(varargin)
```

```
parar=false;
fclose('all')



global tiempo salida escalon y
fig(1)=figure('name','Monitor','menubar','none','position',[200 200 800 700],'color',[0.9 0.6 0.3])
%Texto del Titulo
Texto(1)=uicontrol('parent',fig(1),'style','text','string','Interfaz PIC','position',[200 600 400 50],'BackgroundColor',[0.9 0.6 0.3],'fontsize',25);
movegui(fig(1),'center');
axe(1)=axes('parent',fig(1),'units','pixels','position',[60 80 600 550],'xlim',[0 40],'ylim',[0 200],'xgrid','on','ygrid','on')
set(get(axe(1),'XLabel'),'String','Tiempo (Seg)')
set(get(axe(1),'YLabel'),'String','Temperatura (°C)')
lin(1)=line('parent',axe(1),'xdata',[],'ydata',[],'Color','r','LineWidth',2.5);
bot(1)=uicontrol('parent',fig(1),'style','pushbutton','string','Detener','position',[680 50 100 50],'callback',@stop,'fontsize',11)
txbx(1)=uicontrol('parent',fig(1),'style','tex','string','Temperatura','position',[680 100 100 50],'fontsize',11);
txbx(2)=uicontrol('parent',fig(1),'style','tex','string','Temp °C','position',[680 150 100 50],'fontsize',11)
```

```
%% Funcion Pare
    function varargout=stop(hObject,evendata)
        parar=true;
        fclose(SerialP);
        delete(SerialP);
        clear SerialP;


    end
```

```
     %% funcion Graficar
    % function varargout=grafique(hObject,evendata)
         tiempo=[0];
         salida=[0];
         escalon=[0];
         deg1="0";

         dt=1;
         limx=[0 40];
         limy=[0 200];
         set(axe(1),'xlim',limx,'ylim',limy);
```

```
     %% Configura el Puerto Serial

     SerialP=serial('COM3');
     set(SerialP,'Baudrate',9600); % se configura la velocidad a 9600 Baudios
     set(SerialP,'StopBits',1); % se configura bit de parada a uno
     set(SerialP,'DataBits',8); % se configura que el dato es de 8 bits, debe estar entre 5 y 8
     set(SerialP,'Parity','none'); % se configura sin paridad
     fopen(SerialP);
```

```
%% Grafico
k=5;nit = 10000;
   while(~parar)
       % Lectura del Dato por Puerto Serial
       variable= (fread(SerialP,20,'uchar'));
       ini=find(variable==73); %Busca el retorno de carro (Primer dato)
       ini=ini(1)+1;
       fin=find(variable==70); %Busca operador grados (ultimo dato)
       fin= fin(find(fin>ini))-1;
       fin=fin(1);
       tempC=char(variable(ini:fin))';
       temp=str2num(tempC);

       set(txbx(1),'string',tempC);
       %Actualiza las variables del grafico

        tiempo=[tiempo tiempo(end)+dt];
        salida=[salida temp];
        escalon=[escalon str2num(deg1)];
        set(lin(1),'xdata',tiempo,'ydata',salida);
        pause(dt); %% espera 0.1 seg para cada interación
        if tiempo(end)>=limx % actualizo grafica cuando llega a su limite en tiempo real
        limx=[0 limx(2)+40];
        set(axe(1),'xlim',limx);
        end

        if salida(end)>=limy % actualizo grafica cuando llega a su limite en tiempo real
        limy=[0 limy(2)+30];
        set(axe(1),'ylim',limy);
        end

        k=k+1;
        if(k==nit)
            parar=true;
        end
    end
    parar=false;

end
```

Figura 6. Programación de la interfaz en Matlab.

Descripción paso a paso del código:

Inicialización y Configuración de la Interfaz Gráfica:

El código comienza creando una interfaz gráfica de usuario (GUI) utilizando la función figure de MATLAB. Esta GUI tiene un título ("Interfaz PIC") y un botón de "Detener" para detener la adquisición de datos.

Se crea un gráfico (axes) para visualizar la temperatura medida y se establecen etiquetas para los ejes X e Y.

Se configuran los parámetros de la GUI, como el rango de tiempo y temperatura en el gráfico, utilizando las funciones xlim y ylim.

Configuración del Puerto Serial:

Se configura un objeto de puerto serie (SerialP) para comunicarse con un dispositivo conectado al puerto COM3 a una velocidad de baudios de 9600, 1 bit de parada, 8 bits de datos y sin paridad.

Se abre el puerto serial con la función fopen para iniciar la comunicación.

Bucle de Adquisición de Datos y Actualización del Gráfico:

El código entra en un bucle infinito (while(~parar)) que se ejecutará hasta que se presione el botón "Detener" en la GUI.

Dentro del bucle, se lee un flujo de bytes (presumiblemente datos de temperatura) del puerto serial utilizando fread. Los datos se procesan para extraer la temperatura.

La temperatura se muestra en un cuadro de texto en la GUI y se agrega a un conjunto de datos para su posterior visualización en el gráfico.

El gráfico se actualiza continuamente con el tiempo en el eje X y la temperatura en el eje Y.

Se realiza una pausa de 0.1 segundos (pause(dt)) para controlar la velocidad de actualización del gráfico.

El bucle finaliza si se alcanza una condición específica (posiblemente relacionada con k y nit).

Finalización de la Interfaz:

Cuando se presiona el botón "Detener" en la GUI, se establece la variable parar en true, lo que provoca la salida del bucle de adquisición de datos.

Se cierra y libera el puerto serial utilizando fclose y delete.

```
#include <18f4550.h> // Incluimos el fichero del Microcontrolador PIC18F4550
#FUSES INTRC_IO //Fusible para usar el oscilador interno
#FUSES NOWDT//No Watch Dog Timer
#FUSES WDT128//Watch Dog Timer uses 1:128 Postscale
#FUSES HS//Fusible para usar el oscilador externo de alta velocidad
#FUSES NOPROTECT //Fusible para desactivar la protección del código
#FUSES CPUDIV1 // Fusible para activar el reloj divisor 1
#FUSES PLL1 //Fusible para multiplicar el oscilador interno por un factor de 4

#use delay(clock=20000000)   // Definimos la Frecuencia del oscilador de PIC(20MHz)
#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c5, bits=8)
#include "MAX6675_Lib.c"     // Incluimos la librería MAX6675
#include <lcd.c> // incluimos la libreria del LCD

//Declaramos los pines a usar por la pantalla LCD
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////////////////////////////////////////////

int16 Tm;  // Definimos las variable de temperatura del sensor como flotante

int16 Unidad, Decena, Centena, Centena1; // Definimos las variables como enteros de 16-Bit
 char msg[32];
void main() {                            // Inicia la función principal

   set_tris_d(0x00);                     // Seteamos el puerto D como salida

   lcd_init();

   do{                                   // Inicia al bucle do ... While infinito

      Tm = read_termocupla_k();                 // Lectura de la termocupla tipo K
      //Tm = do_everything();

      Tm = Tm - 35;

      Centena = Tm/100;                         // Captura el dígito de las centenas

      Centena1 = Tm - Centena*100;              // Captura el resto de la división anterior

      Decena = Centena1/10;                     // Captura el dígito de las decenas

      Unidad = Centena1 - Decena*10;            // Captura el dígito de las unidades

     lcd_gotoxy(1,1);
```

```
    LCD_PUTC("TERMOCUPLA K");
     lcd_gotoxy(1,2);
     printf(lcd_putc,"%ld",Centena); // Visualiza el dígito de las centenas en la pantalla, posición (1,2)
     lcd_gotoxy(2,2);
     printf(lcd_putc,"%ld",Decena);              // Visualiza el dígito de las decenas en la pantalla, posición (2,2)
     lcd_gotoxy(3,2);
     printf(lcd_putc,"%ld",Unidad);              // Visualiza el dígito de las unidades en la pantalla, posición (3,2)

    lcd_gotoxy(5,2);

    LCD_PUTC("o");             // Imprime texto en la pantalla

    lcd_gotoxy(6,2);

    LCD_PUTC("C");             // Imprime texto en la pantalla /

    delay_ms(100);                          // Tiempo de espera de 100ms

     sprintf(msg,"I%3.2fFI%3.2fFI%3.2fF",toFloat_TC(read_termocupla_k()),toFloat_TC(read_termocupla_k()),toFloat_TC(read_termocupla_k()));
     printf("%s",msg);

   }while(1);                               // Termina el bucle do ... While infinito
}
```

Figura 7. Programación en CCS Compiler.

Paso N. º 3: Conectar el Pic a la computadora por medio de la interfaz USB y dar play a la interfaz de Matlab.

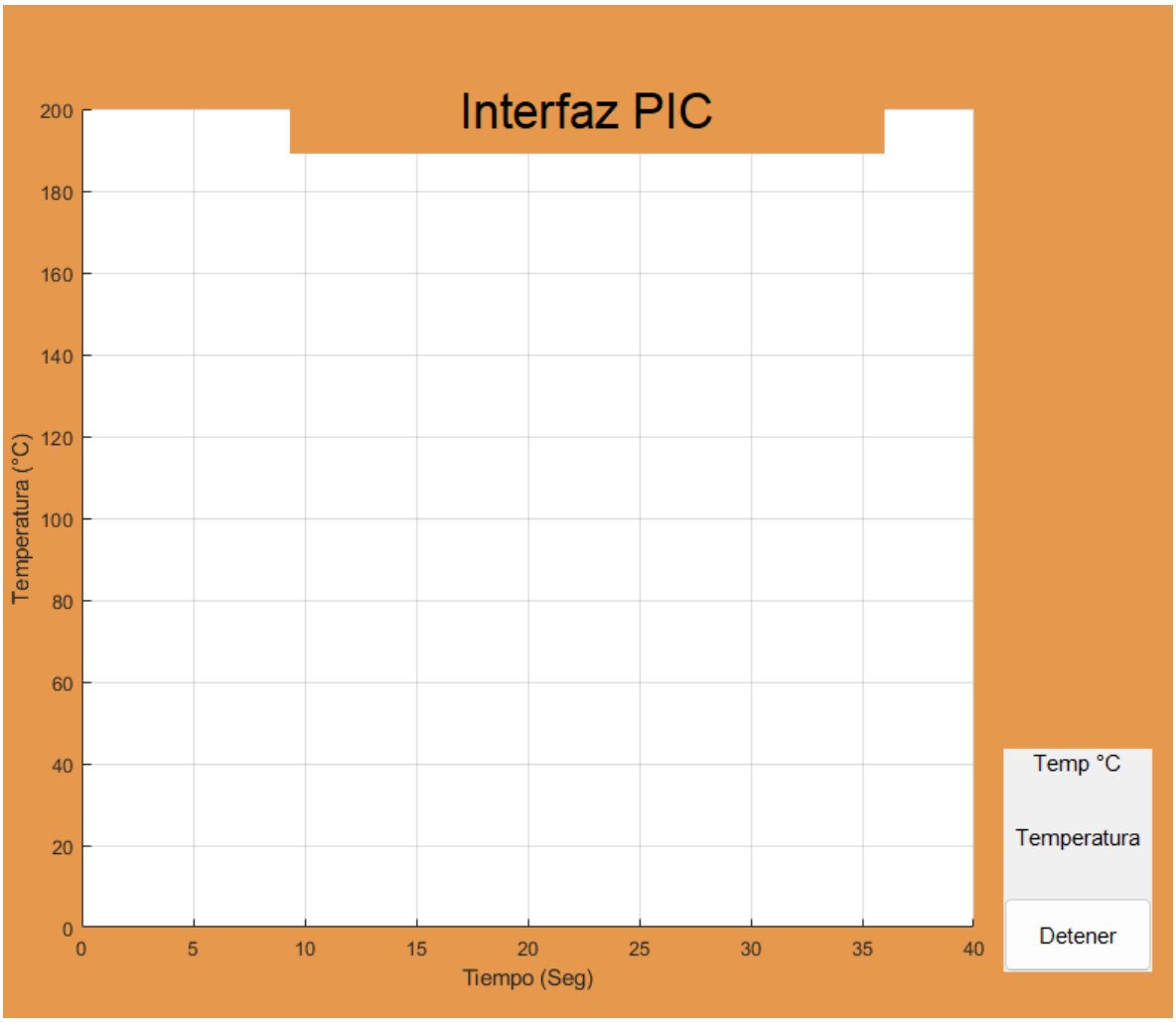


Figura 8. Interfaz en Matlab

Paso N. º 4: Probamos el funcionamiento en el módulo.

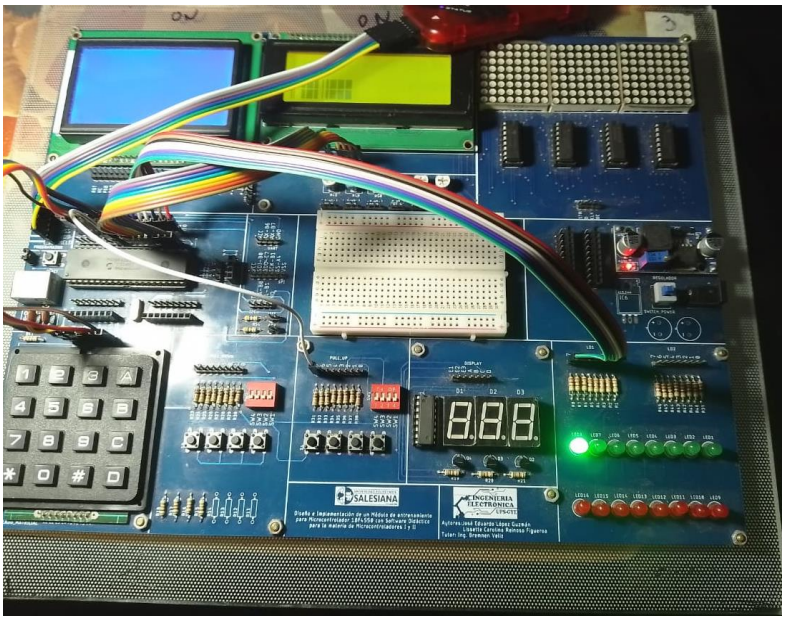


Figura 9. Prueba de funcionamiento.

ANEXO C. CÓDIGOS.

PRÁCTICA # 1: INICIO DE UNA SECUENCIA DE ENCENDIDO DE LEDS MEDIANTE PULSTADORES UTILIZANDO BUBLES FOR.

```
#include <18f4550.h> //Libreria para usar el PIC18F4550

#fuses INTRC_IO //Fusible para usar el oscilador interno

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(internal=8M) // Reloj Interno

#BYTE PORTB=0xF81  //Se declara el puerto B

#byte PORTD=0x0f83 // Se declara el puerto D


void main ()

{

  SET_TRIS_B(0B1111111);          //Configura el puerto B como entradas

  SET_TRIS_D(0B00000000);         //Configura el Puerto D Como salidas

  WHILE(TRUE)                     //Haga por siempre ....

  {
```

```
  delay_ms(2000);
  IF(!BIT_TEST(PORTB,0))        // Pregunta por el estado del pin RB0
  {
    PORTD=0B00000001;            //Prendo unicamente el led de RD0
     DELAY_MS(200);              //Retardo de 500 milisegundos
     WHILE (!BIT_TEST(PORTD,7))    //Haga mientras el LED RB7 se encuentre
apagado
     {
       PORTD=PORTD<<1;           //Rote hacia la izquierda una unidad
       DELAY_MS(200);            //Retardo de 500 milisegundos
     }
     DELAY_MS(200);              //Retardo de 500 milisegundos
     WHILE (!BIT_TEST(PORTD,0))    //Haga mientras el LED RB0 se encuentre
apagado
     {
       PORTD=PORTD>>1;           //Rote hacia la derecha una unidad
       DELAY_MS(200);            //Retardo de 500 milisegundos
     }
     }
  }
}
```

PRÁCTICA # 2: MANEJO DE ADC-DAC CON POTENCIOMETRO.

```
#include <18F4550.h> //Libreria para usar el PIC18F4550

#device adc=10 //seteamos el adc de 10 bits

#fuses INTRC_IO //Fusible para usar el oscilador interno

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(clock=20Mhz) //reloj externo

#include <lcd.c> // incluimos la libreria del LCD

#BYTE PORT_A = 0xf80 //Declaramos el puerto A

#BYTE PORT_D= 0xF83 //Declaramos el puerto B

# USE FAST_IO(A)//usamos el puerto A en fast i/o

# USE FAST_IO(D)//usamos el puerto D en fast i/o

//Declaramos los pines a usar por la pantalla LCD

#define LCD_ENABLE_PIN PIN_D0

#define LCD_RS_PIN PIN_D1

#define LCD_RW_PIN PIN_D2
```

```
#define LCD_DATA4 PIN_D4

#define LCD_DATA5 PIN_D5

#define LCD_DATA6 PIN_D6

#define LCD_DATA7 PIN_D7

////////////////////////////////////////////////

#define vref 5.00     /* Definimos el voltaje de referencia a medir = 5V*/

long bits;     //Variable almacena los bits

float volt;     //Almacena el voltaje

void main()

{

  set_tris_a(0b00000001);          //Pongo el RA0 como entrada

  set_tris_d(0);                   //Pongo el PuertoD como Salida

  setup_adc_ports(all_analog);     //Pongo todo el puerto A analogo

  setup_adc(adc_clock_internal);   //Selecciono reloj interno para conversion

    lcd_init();                    //Inicializo el LCD

  lcd_putc("\f");                  //Borro el LCD

  lcd_gotoxy(1,1);                 //Ubiquese en la posicion 1,1

  lcd_putc("El Voltaje es ");

  while(1)

  {

     set_adc_channel(0);           //Selecciono el canal 0 (RA0)

     delay_ms(1);                  //llamo retardo de 1 ms
```

```
      bits=read_adc();              //Guarde el dato del ADC en volts ****ADC
      volt=bits*((float)vref/(float)1023);//Conversion de bits a voltaje ******DAC
      lcd_gotoxy(2,2);              //Ubiquese en la posicion 2,2
      printf(lcd_putc," %f   Volt ",volt);  //Muestra el valor numerico de la conversión
      delay_ms(100);
   }
}
```

PRÁCTICA # 3: IMPLEMENTACION DE INTERRUPCIONES INTERNAS Y EXTERNAS USANDO PULSADORES.

- **INTERRUPCION EXTERNA**

```
#include <18F4550.h> //Libreria para usar el PIC18F4550

#device adc=10 //seteamos el adc de 10 bits

#fuses INTRC_IO //Fusible para usar el oscilador interno

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(clock=20Mhz) //reloj externo

#include <lcd.c> // incluimos la libreria del LCD

#BYTE PORT_A = 0xf80 //Declaramos el puerto A

#BYTE PORT_D= 0xF83 //Declaramos el puerto B

# USE FAST_IO(A)//usamos el puerto A en fast i/o

# USE FAST_IO(D)//usamos el puerto D en fast i/o

//Declaramos los pines a usar por la pantalla LCD

#define LCD_ENABLE_PIN PIN_D0

#define LCD_RS_PIN PIN_D1
```

```
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////////////////////////////////////////////
#define vref 5.00     /* Definimos el voltaje de referencia a medir = 5V*/
long bits;     //Variable almacena los bits
float volt;     //Almacena el voltaje
void main()
{
  set_tris_a(0b00000001);          //Pongo el RA0 como entrada
  set_tris_d(0);                   //Pongo el PuertoD como Salida
  setup_adc_ports(all_analog);     //Pongo todo el puerto A analogo
  setup_adc(adc_clock_internal);   //Selecciono reloj interno para conversion
    lcd_init();                    //Inicializo el LCD
  lcd_putc("\f");                  //Borro el LCD
  lcd_gotoxy(1,1);                 //Ubiquese en la posicion 1,1
  lcd_putc("El Voltaje es ");
  while(1)
  {
     set_adc_channel(0);           //Selecciono el canal 0 (RA0)
```

```
      delay_ms(1);                //llamo retardo de 1 ms
      bits=read_adc();            //Guarde el dato del ADC en volts ****ADC
      volt=bits*((float)vref/(float)1023);//Conversion de bits a voltaje ******DAC
      lcd_gotoxy(2,2);            //Ubiquese en la posicion 2,2
      printf(lcd_putc," %f   Volt ",volt);  //Muestra el valor numerico de la conversión
      delay_ms(100);
   }
}
```

- **INTERRUPCION INTERNA**

```
#include <18F4550.h>  //Declaramos el PIC18F4550

#device ADC=10  //Seteamos el adc en 10 bits

#fuses INTRC_IO //Fusible para usar el oscilador interno

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses PLL1 //Fusible para multiplicar el oscilador interno por un factor de 4

#fuses CPUDIV1 // Fusible para activar el reloj divisor 1

#use delay(clock=20000000)//Declaramos el reloj externo de 20 Mhz

#byte porta = 0xf80 // Identificador para el puerto A.

#byte portb = 0xf81 // Identificador para el puerto B.

#byte portc = 0xf82 // Identificador para el puerto C.

#byte portd = 0xf83 // Identificador para el puerto D.

#byte porte = 0xf84 // Identificador para el puerto E.

#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c7, bits=8)//Declaramos la
comunicación serial

#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) // Declaramos la
comunicación I2C para el LCD

#include <i2c_Flex_LCD.c> //incluimos la libreria del LCD I2C

#include <stdio.h>//libreria para usar printf

#include <math.h> // libreria para claculos matemáticos
```

```
void temperatura();
char dato;
float temp;
#int_rda // entra a interrupción, ejecuta lo que mande el monitor serial
void rda_isr()
{
  dato=getc();
  switch(dato)
  {
    case '1' : output_high(pin_c2);
               break;
    case '2' : output_low(pin_c2);
               break;
    case '9' : temperatura();
               break;
  }
}
void main()
{
enable_interrupts(int_rda);
lcd_init(0x4E,16,2);
```

```
lcd_backlight_led(ON); //Enciende la luz de Fondo
lcd_clear();  //Limpia el LCD
   while(1)
   {
   }
}
void temperatura(){
      setup_adc_ports(AN0);
      setup_adc(adc_clock_div_64);
      set_adc_channel(0);          //Selecciono el canal 0 (RA0)
      delay_us(50);                //llamo retardo de 50 ms
      temp=(read_adc())/2;         //Convertimos la lectura del adc a temperatura
      lcd_gotoxy(1, 2);
      printf(lcd_putc,"%0.2f °C",temp);
      delay_ms(1000);

}
```

PRÁCTICA # 4: USO DE TIMMERS PARA GENERAR RETARDO DE ENCENDIDO DE LEDS.

```
#include <18f4550.h>// Declaramos al Pic 18F4550

#DEVICE ADC=10//seteamos el adc de 10 bits

#USE DELAY(crystal=20000000)//reloj externo 20 Mhz

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#byte porta = 0xf80 // Identificador para el puerto A.

#byte portb = 0xf81 // Identificador para el puerto B.

#byte portc = 0xf82 // Identificador para el puerto C.

#byte portd = 0xf83 // Identificador para el puerto D.

#byte porte = 0xf84 // Identificador para el puerto E.

int16 Ts_cont=0;

// ******************************** //

// ******* Interrupción Timer 1 **** //

// ******************************** //

#int_timer1  //TIMER1
```

```
void sampling_time(void){
  Ts_cont--;          //Se decrementa hasta llegar a cero
  set_timer1(15536);//Carga de nuevo el timer1
  if (Ts_cont<=0) // Si llega a cero, se cumplió el periodo de muestreo
  {
    output_toggle(pin_c2);
    Ts_cont=100;  // Inicializa el contador para el próximo periodo
  }
}
// ********************************* //
// ****** Programa Principal ******* //
// ********************************* //
void main() {
  set_tris_d(0b0); //Configura el pin D0 como salida (LED)
  // Configura el Timer 1
  setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8 );
  Ts_cont=100;        //Carga Contrador para Calcular el Periodo de Muestreo
  set_timer1(15536); // Inicializa el Timer 1 para calcular 8 Segundos de Ts
  // Habilitar las interrupciones del PIC
  enable_interrupts(int_timer1);
  enable_interrupts(GLOBAL);
  while(1){ }
```

}

PRÁCTICA # 5: LECTURA DE TEMPERATURA CON UNA TERMOCUPLA USANDO COMUNICACIÓN SPI PARA OPTIMIZACION DE PINES.

```
#include <18f4550.h>          // Incluimos el fichero del Microcontrolador PIC18F4550

#fuses HS,NOWDT,NOPROTECT     // Definimos las palabras de configuración del PIC

                              // HS(cristal 20MHz), No Watch Dog Timer, Sin proteccion de memoria de
programa

#use delay(clock=20000000)    // Definimos la Frecuencia del oscilador de PIC(20MHz)

#include "MAX6675_Lib.c"      // Incluimos la librería MAX6675

#include <lcd.c> // incluimos la libreria del LCD

//Declaramos los pines a usar por la pantalla LCD

#define LCD_ENABLE_PIN PIN_D0

#define LCD_RS_PIN PIN_D1

#define LCD_RW_PIN PIN_D2

#define LCD_DATA4 PIN_D4

#define LCD_DATA5 PIN_D5

#define LCD_DATA6 PIN_D6

#define LCD_DATA7 PIN_D7

////////////////////////////////////////////////

int16 Tm;                     // Definimos las variable de temperatura del sensor RTD PT100
como flotante

int16 Unidad, Decena, Centena, Centena1; // Definimos las variables como enteros de 16-Bit
```

```
void main() {                        // Inicia la función principal


  set_tris_d(0x00);                  // Seteamos el puerto D como salida


  lcd_init();
  do{                                // Inicia al bucle do ... While infinito
    Tm = read_termocupla_k();        // Lectura de la termocupla tipo K
    m = Tm - 35;
    Centena = Tm/100;                // Captura el dígito de las centenas
    Centena1 = Tm - Centena*100;     // Captura el resto de la división anterior
    Decena = Centena1/10;            // Captura el dígito de las decenas
    Unidad = Centena1 - Decena*10;   // Captura el dígito de las unidades
   lcd_gotoxy(1,1);
   LCD_PUTC("TERMOCUPLA K");
    lcd_gotoxy(1,2);
    printf(lcd_putc,"%ld",Centena); // Visualiza el dígito de las centenas en la pantalla, posición (1,2)
    lcd_gotoxy(2,2);
    printf(lcd_putc,"%ld",Decena);   // Visualiza el dígito de las decenas en la pantalla, posición (2,2)
    lcd_gotoxy(3,2);
    printf(lcd_putc,"%ld",Unidad);   // Visualiza el dígito de las unidades en la pantalla, posición (3,2)
   lcd_gotoxy(5,2);
```

```
    LCD_PUTC("o");              // Imprime texto en la pantalla
    lcd_gotoxy(6,2);


    LCD_PUTC("C");              // Imprime texto en la pantalla /
    delay_ms(100);                      // Tiempo de espera de 100ms
   }while(1);                           // Termina el bucle do ... While infinito
}
```

PRÁCTICA # 6: CONEXIÓN DE LCD USANDO COMUNICACIÓN I2C PARA LA OPTIMIZACION DE PINES.

```
#include <18f4550.h>// Declaramos al Pic 18F4550

#USE DELAY(crystal=200000000)//Seteamos el reloj externo a 20MHz

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD

#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C

#define LCD_COLS 16

void main() {

lcd_init(0x4E,16,2);

lcd_backlight_led(ON); //Enciende la luz de Fondo

lcd_clear();  //Limpia el LCD

   char mensaje[] = "Hola, Dennys Leon !";

   int mensaje_len = 21; // tamaño del mensaje

   char mensaje2[] = "Universidad Salesiana";
```

```
    int end_index = mensaje_len - LCD_COLS;


while(true) {
 for (int i = 0; i <= end_index; i++)
       {
           lcd_gotoxy(1, 1);
           for (int j = i; j < i + LCD_COLS; j++)
           {
               lcd_putc(mensaje[j]);
           }
           delay_ms(100);
            lcd_gotoxy(1, 2);
           for (int k = i; k < i + LCD_COLS; k++)
           {
               lcd_putc(mensaje2[k]);
           }
           delay_ms(100);
       }
}
}
```

PRÁCTICA # 7: CONTROL DE ACCESO (CERRADURA ELECTRONICA) USANDO LCD Y TECLADO MATRICIAL.

```
#include <18F4550.h>// Declaramos al Pic 18F4550

#fuses INTRC_IO //Fusible para usar el oscilador interno

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#fuses PUT // Fusible para activar el Power-Up Timer

#use delay(clock=20Mhz)//Seteamos el reloj externo a 20MHz

#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD

#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C

#include <stdlib.h>//libreria para usar printf

#BYTE PORT_A = 0xf80//Declaramos el puerto A

#BYTE PORT_B = 0xf81//Declaramos el puerto B

#define row0 PIN_B0                    //Filas del teclado colocar resistencia pullup

#define row1 PIN_B1                    //Filas del teclado colocar resistencia pullup

#define row2 PIN_B2                    //Filas del teclado colocar resistencia pullup

#define row3 PIN_B3                    //Filas del teclado colocar resistencia pullup
```

```
#define col0 PIN_B4                    //Columnas del teclado

#define col1 PIN_B5                    //Columnas del teclado

#define col2 PIN_B6                    //Columnas del teclado

#define col3 PIN_B7                    //Columnas del teclado

#include "Teclado4x4.h"                 //Librería de teclado

//Variables Globales

CHAR K;

void main(){

 WRITE_EEPROM(0,7); //Guarda en la eemprom posicion cero la nueva contraseña

 WRITE_EEPROM(1,2);

 WRITE_EEPROM(2,3);

 WRITE_EEPROM(3,3);

char k;   //declaracion de variable

int i;   //declaracion de variable

char data[4], clave[4] ;   //declaracion de arreglo de variable

lcd_init(0x4E,16,2);

lcd_backlight_led(ON); //Enciende la luz de Fondo

lcd_clear();  //Limpia el LCD

kbd_init();   //inicializa teclado

port_b_pullups(true); //inicializa pull ups puerto b

while (true){      //mientras sea verdad

i=0;       //i=0
```

```
lcd_gotoxy(1,1);
printf(lcd_putc,"\f pulsar tecla 1\n"); //muestra mensaje cursor posicion 1,2
//lazo de captura de datos----------
while(i<=3){      //mientras i<=2 hacer
k=kbd_getc();       //captura dato del teclado
if(k!=0){         //pregunta si k es distinto (k!=0) a cero
 switch (k)
  {
     case 48:
        k=0;
        break;
     case 49:
        k=1;
        break;
     case 50:
        k=2;
        break;
     case 51:
        k=3;
        break;
     case 52:
        k=4;
```

```
        break;


      case 53:
        k=5;
        break;
      case 54:
        k=6;
        break;
      case 55:
        k=7;
        break;
      case 56:
        k=8;
        break;
      case 57:
        k=9;
        break;
      default:
        k=0;
        break;
   }
data[i]=k;        //guardo en espacio de data segun i el valor de k
```

```
i++;            //incremento i
if(i<4){
lcd_clear();  //Limpia el LCD
 lcd_gotoxy(1, 1);
printf(lcd_putc," pulsar tecla %u",i+1);
}      //imprime valor numerico}
lcd_gotoxy(1, 2);
printf(lcd_putc," # pulsado %u",k);
}
}
//------------------------------
for (i=0 ; i<=3 ; i++){   //incremento i de 0 hasta 2
clave[i]=read_eeprom(i);
delay_ms(50);
}   //guarda la clave de la eeprom en la variable clave
                        //en cada posicion de i, por eso hace lazo hasta i=2
if ((data[0]==clave[0])&&(data[1]==clave[1])&&(data[2]==clave[2])&&(data[3]==clave[3])){ //si
los datos son iguales
printf(lcd_putc,"\f puerta abierta");      //muestra mensaje puerta abierta
output_high(pin_a1);
delay_ms(1000);
output_low(pin_a1);
}
```

```
else      //si los datos no son iguales
printf(lcd_putc,"\f acceso denegado");      //muestra mensaje puerta cerrada
delay_ms(1000);
}
}
```

PRÁCTICA # 8: COMUNICACIÓN SERIAL USP ENTRE DOS PIC.

- **PIC 1 ENVÍA**

```
#include <18F4550.h> //Libreria para usar el PIC18F4550
#fuses HS //Fusible para usar el oscilador externo de alta velocidad
#fuses NOWDT //Fusible para descativar el watchdog timer
#fuses NOPROTECT //Fusible para desactivar la protección del código
#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje
#use delay(clock=20000000)//reloj externo
#use rs232(baud=9600, xmit=PIN_c6,rcv=PIN_c7,parity=N, bits=8)//Declaramos
                                        //la comunicación serial
#define BUTTON_A  PIN_D0 // Definimos el botón para el comando de 'A' alto
#define BUTTON_B  PIN_D1 // Definimos el botón para el comando de 'B' bajo


void main()
{
   set_tris_d(0xFF); // Seteamos el puerto D como entradas
   while (1)
   {
      if (!input(BUTTON_A)) // Si el botón de alto es presionado
      {
```

```
            delay_ms(100); // un delay
            printf("A\r\n"); // envia 'A' por serial
        }
        if (!input(BUTTON_B)) // Si el botón de bajo es presionado
        {
            delay_ms(100); // un delay
            printf("B\r\n"); // envia 'B' por serial
        }
        delay_ms(20); // retardo para estabilidad
    }
}
```

- **PIC 2 RECIBE**

```
#include <18F4550.h>//Libreria para usar el PIC18F4550

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#use delay(clock=20000000) //reloj externo

#use rs232(baud=9600, xmit=PIN_c6,rcv=PIN_c7,parity=N, bits=8)//Declaramos la comunicación serial

#use i2c(Master,Fast=100000, sda=PIN_B0, scl=PIN_B1,force_sw) //Declaramos la comunicación I2C para el LCD

#include <i2c_Flex_LCD.c>//Incluimos la libreria del LCD I2C

#include <stdio.h>//para usar printf

#include <math.h> // libreria para claculos matemáticos

void main()

{

lcd_init(0x4E,16,2);

lcd_backlight_led(ON); //Enciende la luz de Fondo

lcd_clear();  //Limpia el LCD

   set_tris_c(0b11111110); // Setea RC2 como salida, el resto de pines como entradas

   lcd_gotoxy(1,1);
```

```
lcd_putc("Control De LED");
lcd_gotoxy(1,2);
lcd_putc("Comunicacion Serial");
while (1)
{
   if (kbhit()) // Chequea si hay un dato disponible en el buffer UART
   {
      char datoRecibido = getc(); // Reibe un dato desde el UART
      putc(datoRecibido);        // Eco del dato recibido

      if (datoRecibido == '\r') // Chequea por un final de linea
      {
         putc('\n'); // agrega un caracter de nueva linea
      }
      if (datoRecibido == 'A') // Si dato recibido es 'A' alto
      {
       LCD_PUTC("\f");   //Borrar el contenido del LCD
       lcd_gotoxy(1,1);
       lcd_putc("LED");
       lcd_gotoxy(1,2);
       printf(lcd_putc,"Encendido");
         output_high(PIN_C2); // Setea RC2 alto
```

```
        }
        else if (datoRecibido == 'B') // Si dato recibido es 'B' bajo
        {
           LCD_PUTC("\f");   //Borrar el contenido del LCD
             lcd_gotoxy(1,1);
             lcd_putc("LED");
            lcd_gotoxy(1,2);
             printf(lcd_putc,"Apagado");
           output_low(PIN_C2); // Setea RC2 bajo
        }
      }
   }
}
```

PRÁCTICA # 9: CONTROL DE VELOCIDAD DE UN MOTOR MEDIANTE UN POTENCIOMETRO Y TECLADO MATRICIAL.

```
#include <18F4550.h>// Declaramos al Pic 18F4550

#fuses HS //Fusible para usar el oscilador externo de alta velocidad

#fuses NOWDT //Fusible para descativar el watchdog timer

#fuses NOPROTECT //Fusible para desactivar la protección del código

#fuses NOLVP //Fusible para desctivar la programación por bajo voltaje

#fuses NODEBUG // Fusible para no usar la función debug

#fuses NOBROWNOUT // Fusible para desactivar el reset de Brown-Out

#DEVICE ADC=10 //Seteamos la conversión analógica con una resolución de 10 bits

#USE DELAY(clock=20000000)//Seteamos el reloj externo a 20MHz


#BYTE PORT_A = 0xf80//Declaramos el puerto A

#BYTE PORT_B = 0xf81//Declaramos el puerto B

#BYTE PORT_C = 0xf82//Declaramos el puerto C

#BYTE port_D = 0xf83//Declaramos el puerto D


# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento

# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento

# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento
```

```
# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento

#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos

#INCLUDE <LCD.C> // Incluimos la libreria de la pantalla LCD

/////////Definimos los pines de la pantalla LCD///////
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
/////////Definimos los pines de la pantalla LCD///////

#define row0 PIN_B0                    //Filas del teclado colocar resistencia pullup
#define row1 PIN_B1                    //Filas del teclado colocar resistencia pullup
#define row2 PIN_B2                    //Filas del teclado colocar resistencia pullup
#define row3 PIN_B3                    //Filas del teclado colocar resistencia pullup
#define col0 PIN_B4                   //Columnas del teclado
#define col1 PIN_B5                   //Columnas del teclado
```

```
#define col2 PIN_B6                    //Columnas del teclado

#define col3 PIN_B7                    //Columnas del teclado

#include "Teclado4x4.h"                 //Librería de teclado


#include <stdlib.h>//libreria para usar printf

#include <string.h>//Libreria para convertir lo ingresado por el teclado a float


int16 adc,velocidad=0; //declaramos las variables para la lectura del potenciómetro y la velocidad del motor

float R=0; // Variable auxiliar para la velocidad

char c; // variable para guardar el caracter que se ingresa por el teclado




/*================================================================================*/

/*=======================       FUNCION TECLA       =======================*/

/*================================================================================*/

//Funcion encargada de esperar a que se presione una tecla

char tecla(void)

{

  char c;

  do{ //espera hasta que se presione una tecla

    c=kbd_getc(); //Captura valor del teclado
```

```
    }
  while(c=='\0');
  return(c);
}

/*=====================================================================
======*/
/*=======================    FUNCION TECLA CON TIMER
=======================*/
/*=====================================================================
======*/
// Pregunta por una Tecla por un tiempo, si no hay actividad, deja de preguntar
// y deja que el PIC continue con su trabajo

char tecla_time(void) {
  char c='\0';
  unsigned int16 timeout;
  timeout=0;
  c=kbd_getc(); //Captura valor del teclado
  while(c=='\0' && (++timeout< (KEYHIT_DELAY*150)))
  {
    delay_us(10);
    c=kbd_getc(); //Captura valor del teclado
  }
```

```
  return(c);
}



/*=====================================================================
======*/
/*============     FUNCION PARA DIGITAR LA VELOCIDAD
====================*/
/*=====================================================================
======*/
long porcentaje(int nd)
{
  //Esta funcion captura el escalon desde el teclado, si el proceso está tomando
  //datos el escalon sirve para exitar el sistema, por otro lado si el sistema
  //está controlando, el escalo sirve para establecer el setpoint del sistema

  long val;
  int i;
  char str[6]; //Variable tipo String

  str[0]='0';
  for(i=0;i<nd;i++)
  {
```

```
    c=tecla();  //Lee el valor del teclado y espera hasta que alguna tecla se pulse
    if(c!='#'){
      //Muestra el digito presionado en el LCD
      lcd_gotoxy(25+i,4);
      lcd_putc(c);
      //Almacena el dato presionado en la variable String
      str[i]=c;
    }
    else{i=nd;} //Si se presiona * sale del For
  }
  val = atol(str); //Convierte el String en un valor numerico
}

/*=====================================================================
======*/
/*=======================    FUNCION DEL PRINCIPAL
=======================*/
/*=====================================================================
======*/
void main()
{
  port_b_pullups(true); //inicializa pull ups puerto b
  set_tris_a(0b00000011);         //Pongo el RA0 como entrada
  set_tris_c(0);
```

```
  set_tris_d(0);
  setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM
a 1kHz
  setup_ccp1(ccp_pwm);               //Configurar modulo CCP1 en modo PWM
  set_pwm1_duty(0);                  //Dejo en cero la salida PWM
  setup_adc_ports(AN0);              //Configurar ADC (Lectura de temperatura)
  setup_adc(adc_clock_internal);      //Reloj interno para la conversion analoga digital)
  set_adc_channel(0);                //Seleccionar Canal 0 para sensor de Temperatura
  LCD_INIT();                        //Inicializo el LCD
  kbd_init();   //inicializa teclado
  LCD_PUTC("\f");                    //Limpio el LCD
    lcd_gotoxy(1,1);
    LCD_PUTC("VELOCIDAD MOTOR");
    lcd_gotoxy(5,2);
    LCD_PUTC("%");
    lcd_gotoxy(21,1);
    LCD_PUTC("PRESIONE'A' PARA TECL");
    lcd_gotoxy(21,2);
    LCD_PUTC("PRESIONE'B' PARA POT");
  while(1)
  {
    lcd_gotoxy(1,2);
```

```
    printf(lcd_putc,"%Lu\r\n",velocidad/10);//Se visualiza en la pantalla la velocidad en
porcentaje de 0 a 100%

    set_pwm1_duty(velocidad);//seteamos la velocidad del motor

     //Espera por 10 mili segundos

     delay_ms(10);


      c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado

/*=======================    INGRESAR VELOCIDAD POR EL TECLADO
======================*/

      if(c=='A')

      {

        LCD_PUTC("\f");

        lcd_gotoxy(1,1);

        LCD_PUTC("DIGITE % VELOCIDAD");

        lcd_gotoxy(1,2);

        LCD_PUTC("Velocidad(0-100%)");

        lcd_gotoxy(21,1);

        LCD_PUTC("y presione '#'");

        lcd_gotoxy(21,2);

        LCD_PUTC("Vel:          ");

        R=porcentaje(3); //Llama la funcion para digitar el porcentaje de exitacion

        //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)

        R =(R > 100) ? 1000:R*10;
```

```
          LCD_PUTC("\f");
          //Muestra el Porcentaje de Velocidad en pantalla
          lcd_gotoxy(21,2);
          printf(lcd_putc,"Vel: %3.1f",R/10);
          velocidad=R;
          delay_ms(100);
          LCD_PUTC("\f");
          lcd_gotoxy(1,1);
          LCD_PUTC("VELOCIDAD MOTOR");
          lcd_gotoxy(5,2);
          LCD_PUTC("%");
          lcd_gotoxy(21,1);
          LCD_PUTC("PRESIONE'A' PARA TECL");
          lcd_gotoxy(21,2);
          LCD_PUTC("PRESIONE'B' PARA POT");
        }
/*=====================================================================
======*/
/*=======================    CONTROL DE VELOCIDAD POR EL POTENCIÓMETRO
=======================*/
        if(c=='B')
        {
         LCD_PUTC("\f");
```

```
  lcd_gotoxy(1,1);
  LCD_PUTC("PRESIONE '#'");
  lcd_gotoxy(1,2);
  LCD_PUTC("PARA SALIR");
 WHILE(1){
  c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
  //Lectura del Potenciometro
  set_adc_channel(0);   //Escoge el canal 0 de los puertos analogos
  delay_us(50);
  adc=read_adc();      //Almacena en duty el valor del voltaje del pot
  adc =(adc > 1000) ? 1000:adc;
  lcd_gotoxy(21,1);
  printf(lcd_putc,"%Lu\r\n",adc/10);
  lcd_gotoxy(25,1);
  LCD_PUTC("%");
  set_pwm1_duty(adc);
  delay_ms(50);
  if(c=='#'){
  LCD_PUTC("\f");
  lcd_gotoxy(1,1);
  LCD_PUTC("VELOCIDAD MOTOR");
  lcd_gotoxy(5,2);
```

```
            LCD_PUTC("%");
            lcd_gotoxy(21,1);
            LCD_PUTC("PRESIONE'A' PARA TECL");
            lcd_gotoxy(21,2);
            LCD_PUTC("PRESIONE'B' PARA POT");
            break;
            }
          }
        }

/*=====================================================================
======*/


  }
}
```

PRÁCTICA # 10: CONTROL PID DE UNA RESISTENCIA CALEFACTORA.

```
#INCLUDE <18F4550.h>// Declaramos al Pic 18F4550

#fuses INTRC_IO, NOWDT, WDT128, HS, NOPROTECT, CPUDIV1, PLL1// Seteamos los fusibles


#use delay(clock=20000000)//Seteamos el reloj externo a 20MHz


#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c5, bits=8)//Seteamos la comunicación serial

#include "MAX6675_Lib.c"//Incluimos la libreria de la Interfaz para la termocupla

#include <lcd.c>// Incluimos la libreria de la pantalla LCD


//Configura direccion de memoria de los puertos A,B,C,D

#BYTE PORTA=0xF80

#BYTE PORTB=0xF81

#BYTE PORTC=0xF82

#BYTE PORTD= 0xF83

//////////////////////////////////////////////////////

# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento

# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento

# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento
```

```
# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento

#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos


////////Definimos los pines de la pantalla LCD///////

#define LCD_ENABLE_PIN PIN_D0

#define LCD_RS_PIN PIN_D1

#define LCD_RW_PIN PIN_D2

#define LCD_DATA4 PIN_D4

#define LCD_DATA5 PIN_D5

#define LCD_DATA6 PIN_D6

#define LCD_DATA7 PIN_D7

////////Definimos los pines de la pantalla LCD///////


#define row0 PIN_B0                  //Filas del teclado colocar resistencia pullup

#define row1 PIN_B1                  //Filas del teclado colocar resistencia pullup

#define row2 PIN_B2                  //Filas del teclado colocar resistencia pullup

#define row3 PIN_B3                  //Filas del teclado colocar resistencia pullup

#define col0 PIN_B4                 //Columnas del teclado

#define col1 PIN_B5                 //Columnas del teclado

#define col2 PIN_B6                 //Columnas del teclado

#define col3 PIN_B7                 //Columnas del teclado

#include "Teclado4x4.h"              //Librería de teclado
```

```
#include <stdlib.h>//libreria para usar printf

#include <string.h>//Libreria para convertir lo ingresado por el teclado a float


unsigned int16 adc,control=0;//Lectura de la entrada analógica y variable de control iniciada en 0

float R=750;//Referencia de 45 °C por defecto

float Tm,yM=0,e=0.0,e_1=0.0,e_2=0.0,u=0.0,u_1=0.0,T=0.1; // variables del pid

float kp,ti,td,q0,q1,q2; // variables del pid

float k=0.535,tao=10.0,theta=1.0;// variables del pid

float TsMA,Wn,P1,P2; // variables del pid

char c;// variable para guardar el caracter que se ingresa por el teclado

int i=0; //Posicion del dato recibido a mostrar en el LCD

char msg[32];//La variable Caracter que se va a enviar por comunicación serial

int16 Sensor;   // Definimos las variable de temperatura del sensor

/*================================================================
======*/

/*=======================       FUNCION TECLA       =======================*/

/*================================================================
======*/

//Funcion encargada de esperar a que se presione una tecla

char tecla(void)

{

  char c;

  do{ //espera hasta que se presione una tecla
```

```
    c=kbd_getc(); //Captura valor del teclado
   }
  while(c=='\0');
  return(c);
}

/*=====================================================================
======*/
/*=======================    FUNCION TECLA CON TIMER
=======================*/
/*=====================================================================
======*/
// Pregunta por una Tecla por un tiempo, si no hay actividad, deja de preguntar
// y deja que el PIC continue con su trabajo

char tecla_time(void) {
  char c='\0';
  unsigned int16 timeout;
  timeout=0;
  c=kbd_getc(); //Captura valor del teclado
  while(c=='\0' && (++timeout< (KEYHIT_DELAY*150)))
  {
    delay_us(10);
    c=kbd_getc(); //Captura valor del teclado
```

```
  }
  return(c);
}

/*=====================================================================
======*/
/*=======================    FUNCION DEL CONTROL PID
=======================*/
/*=====================================================================
======*/
void PID(void)
{

   e=1*(R-yM);
   // Controle PID
     u = u_1 + q0*e + q1*e_1 + q2*e_2; //Ley del controlador PID discreto

   if (u >= 5000.0)       //Saturo la accion de control 'uT' en un tope maximo y minimo
    u = 5000.0;

   if (u <= 0.0)
    u = 0.0;
```

```
    control=u/5;


    //Retorno a los valores reales
    e_2=e_1;
    e_1=e;
    u_1=u;


    //La accion calculada la transformo en PWM
    set_pwm1_duty(control);


}




/*=====================================================================
======*/
/*============   FUNCION PARA DIGITAR ESCALÓN/SETPOINT
====================*/
/*=====================================================================
======*/
long escalon(int nd)
{
  //Esta funcion captura el escalon desde el teclado, si el proceso está tomando
  //datos el escalon sirve para exitar el sistema, por otro lado si el sistema
  //está controlando, el escalo sirve para establecer el setpoint del sistema
```

```
  long val;
  int i;
  char str[5]; //Variable tipo String

  str[0]='0';
  for(i=0;i<nd;i++)
  {

    c=tecla();  //Lee el valor del teclado y espera hasta que alguna tecla se pulse
    if(c!='*'){
      //Muestra el digito presionado en el LCD
      lcd_gotoxy(26+i,4);
      lcd_putc(c);
      //Almacena el dato presionado en la variable String
      str[i]=c;
    }
    else{i=nd;} //Si se presiona * sale del For
  }
  val = atol(str); //Convierte el String en un valor numerico
  return(val);
}
```

```
void main()
{
  LCD_INIT();   //Inicializo el LCD
  delay_ms(50);      //permite estabilizar al oscilador
  port_b_pullups (true);  //Utiliza las resistencias PULL UP internas del puerto B
  set_tris_a(0b00000011);          //Pongo el RA0 como entrada
  set_tris_c(0);
  set_tris_d(0);
  setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM
a 1kHz
  setup_ccp1(ccp_pwm);               //Configurar modulo CCP1 en modo PWM
  set_pwm1_duty(0);                  //Dejo en cero la salida PWM
  //setup_adc_ports(AN0);              //Configurar ADC (Lectura de temperatura)
  //setup_adc(ADC_CLOCK_INTERNAL);      //Reloj interno para la conversion analoga
digital)
  kbd_init();   //inicializa teclado

  //*************************************************************************//
  //*************  DISEÑO POR ASIGNACIÓN DE 2 POLOS REALES   ****************//
  //*************************************************************************//

  TsMA=30;                  //Tiempo deseado en Lazo Cerrado
```

```
  Wn=5.8335/(TsMA);          //Frecuencia natural del sistema


  //Ubicación de 2 Polos reales
  P1=2*Wn;
  P2=Wn*Wn;


  kp=(P1*tao-1)/k;        //Calculo de Kc
  ti=(k*kp)/(P2*tao);     //Calculo de ti



 // Calculo del controlador PID digital
  q0=kp*(1+T/(2*ti)+td/T);
  q1=-kp*(1-T/(2*ti)+(2*td)/T);
  q2=(kp*td)/T;


   for(;;)
{
/*
  set_adc_channel(0);               //Seleccionar Canal 0 para sensor de Temperatura
  adc=read_adc(); //Leer ADC
  delay_us(10); // Delay for stabilization
  voltSalida=adc*((float)vref/(float)1024);//Conversion de bits a voltaje ******DAC
```

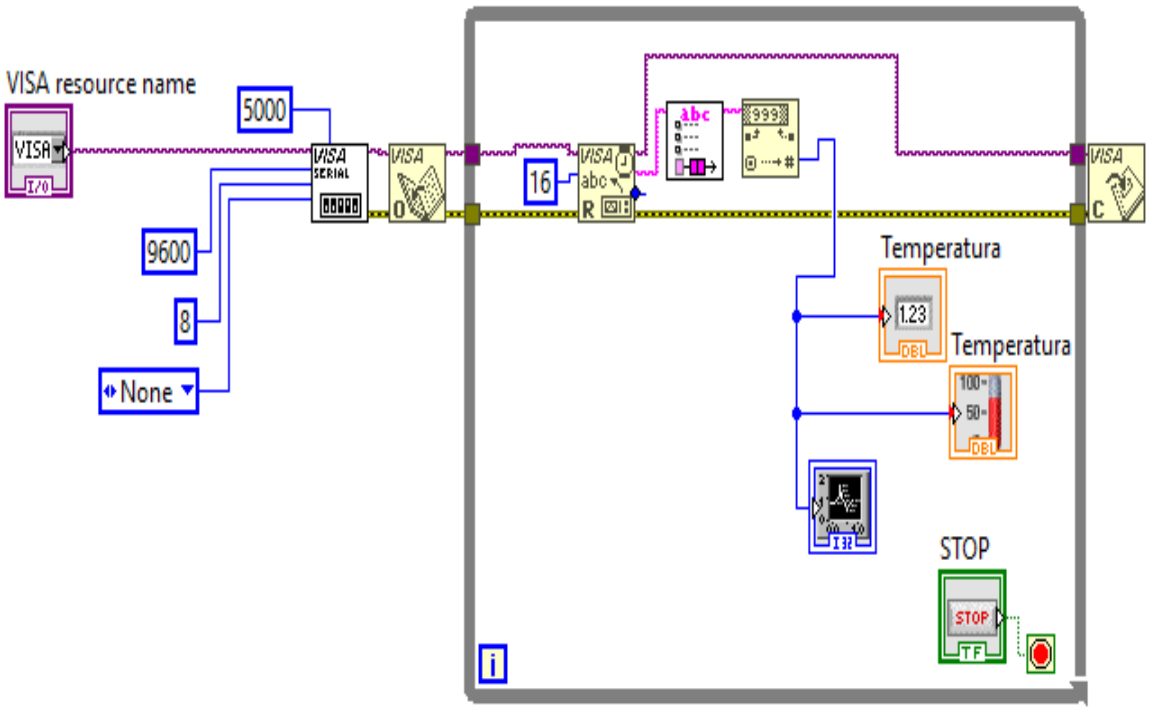
```
  ym=((voltSalida+0.0075625)/0.00578125)*10.00;//Formula cáculada para el uso de la
                                        //trajeta interfaz
  */
       Sensor=  read_termocupla_k();
       Tm = toFloat_TC(Sensor);
       delay_us(10); // Delay for stabilization
       ym = (Tm)*10;
       //Llama la funcion del controlador PID
       PID();
       //tiempo de muestreo
       delay_ms(73);

         lcd_gotoxy(2,1);
         lcd_putc("CONTROL PID RES");
         lcd_gotoxy(1,2);
         printf(lcd_putc,"Set-Point  : %3.1f C   ",R/10);
         lcd_gotoxy(21+i,1);
         printf(lcd_putc,"Temperatura: %3.1f C",yM/10);
         sprintf(msg,"%ld\r\n",Sensor-27); //Para Labview
         printf("%s",msg); //Para Labview

//sprintf(msg,"I%3.2fFI%3.2fFI%3.2fF",toFloat_TC(Sensor),toFloat_TC(Sensor),toFloat_TC(S
ensor)); //Para Matlab
```

```
   //printf("%s",msg); // Para Matlab
   lcd_gotoxy(21,2);
   printf(lcd_putc,"Ley Control: %ld   ",control);

c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
if(c=='D')
{
   LCD_PUTC("\f");
   lcd_gotoxy(1,2);
   LCD_PUTC("Referencia(0-100):");
   lcd_gotoxy(21,1);
   LCD_PUTC("y presione *");
   lcd_gotoxy(21,2);
   LCD_PUTC("SP:                    ");
   R=escalon(3); //Llama la funcion para digitar el escalon de exitacion
   //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)
   R =(R > 100) ? 1000:R*10;

   //Muestra el SETPOINT en pantalla
   lcd_gotoxy(21,2);
   printf(lcd_putc,"SP: %3.1f        ",R/10);
   delay_ms(2000);
```

```
            LCD_PUTC('\f');
        }
    }


}
```

PRÁCTICA # 11: SCADA EN LABVIEW PARA EL CONTROL DE PID DE UNA RESISTENCIA CALEFACTORA.

Labview:



CCS:

#INCLUDE <18F4550.h>// Declaramos al Pic 18F4550

#fuses INTRC_IO, NOWDT, WDT128, HS, NOPROTECT, CPUDIV1, PLL1// Seteamos los fusibles

#use delay(clock=20000000)//Seteamos el reloj externo a 20MHz

#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c5, bits=8)//Seteamos la comunicación serial

#include "MAX6675_Lib.c"//Incluimos la libreria de la Interfaz para la termocupla

#include <lcd.c>// Incluimos la libreria de la pantalla LCD

```
//Configura direccion de memoria de los puertos A,B,C,D

#BYTE PORTA=0xF80

#BYTE PORTB=0xF81

#BYTE PORTC=0xF82

#BYTE PORTD= 0xF83

//////////////////////////////////////////////////////

# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento

# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento

# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento

# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento

#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos


/////////Definimos los pines de la pantalla LCD///////

#define LCD_ENABLE_PIN PIN_D0

#define LCD_RS_PIN PIN_D1

#define LCD_RW_PIN PIN_D2

#define LCD_DATA4 PIN_D4

#define LCD_DATA5 PIN_D5

#define LCD_DATA6 PIN_D6
```

```
#define LCD_DATA7 PIN_D7

////////Definimos los pines de la pantalla LCD///////


#define row0 PIN_B0                    //Filas del teclado colocar resistencia pullup

#define row1 PIN_B1                    //Filas del teclado colocar resistencia pullup

#define row2 PIN_B2                    //Filas del teclado colocar resistencia pullup

#define row3 PIN_B3                    //Filas del teclado colocar resistencia pullup

#define col0 PIN_B4                   //Columnas del teclado

#define col1 PIN_B5                   //Columnas del teclado

#define col2 PIN_B6                   //Columnas del teclado

#define col3 PIN_B7                   //Columnas del teclado

#include "Teclado4x4.h"                //Librería de teclado

#include <stdlib.h>//libreria para usar printf

#include <string.h>//Libreria para convertir lo ingresado por el teclado a float


unsigned int16 adc,control=0;//Lectura de la entrada analógica y variable de control iniciada
en 0

float R=750;//Referencia de 45 °C por defecto

float Tm,yM=0,e=0.0,e_1=0.0,e_2=0.0,u=0.0,u_1=0.0,T=0.1; // variables del pid

float kp,ti,td,q0,q1,q2; // variables del pid

float k=0.535,tao=10.0,theta=1.0;// variables del pid

float TsMA,Wn,P1,P2; // variables del pid

char c;// variable para guardar el caracter que se ingresa por el teclado
```

```
int i=0; //Posicion del dato recibido a mostrar en el LCD

char msg[32];//La variable Caracter que se va a enviar por comunicación serial

int16 Sensor;   // Definimos las variable de temperatura del sensor

/*=====================================================================
======*/

/*======================       FUNCION TECLA        ======================*/

/*=====================================================================
======*/

//Funcion encargada de esperar a que se presione una tecla

char tecla(void)

{

  char c;

  do{ //espera hasta que se presione una tecla

     c=kbd_getc(); //Captura valor del teclado

    }

  while(c=='\0');

  return(c);

}


/*=====================================================================
======*/

/*======================    FUNCION TECLA CON TIMER
======================*/
```

```
/*=====================================================================
======*/

// Pregunta por una Tecla por un tiempo, si no hay actividad, deja de preguntar

// y deja que el PIC continue con su trabajo


char tecla_time(void) {

  char c='\0';

  unsigned int16 timeout;

  timeout=0;

  c=kbd_getc(); //Captura valor del teclado

  while(c=='\0' && (++timeout< (KEYHIT_DELAY*150)))

  {

    delay_us(10);

    c=kbd_getc(); //Captura valor del teclado

  }

  return(c);

}




/*=====================================================================
======*/

/*======================    FUNCION DEL CONTROL PID
======================*/
```

```
/*================================================================
======*/
void PID(void)
{

   e=1*(R-yM);
   // Controle PID
     u = u_1 + q0*e + q1*e_1 + q2*e_2; //Ley del controlador PID discreto

   if (u >= 5000.0)        //Saturo la accion de control 'uT' en un tope maximo y minimo
    u = 5000.0;

   if (u <= 0.0)
    u = 0.0;

    control=u/5;

    //Retorno a los valores reales
    e_2=e_1;
    e_1=e;
    u_1=u;

    //La accion calculada la transformo en PWM
```

```
    set_pwm1_duty(control);


}



/*=====================================================================
======*/

/*============   FUNCION PARA DIGITAR ESCALÓN/SETPOINT
====================*/

/*=====================================================================
======*/

long escalon(int nd)

{

  //Esta funcion captura el escalon desde el teclado, si el proceso está tomando

  //datos el escalon sirve para exitar el sistema, por otro lado si el sistema

  //está controlando, el escalo sirve para establecer el setpoint del sistema


  long val;

  int i;

  char str[5]; //Variable tipo String


  str[0]='0';

  for(i=0;i<nd;i++)

  {
```

```
      c=tecla();  //Lee el valor del teclado y espera hasta que alguna tecla se pulse
      if(c!='*'){
         //Muestra el digito presionado en el LCD
         lcd_gotoxy(26+i,4);
         lcd_putc(c);
         //Almacena el dato presionado en la variable String
         str[i]=c;
      }
      else{i=nd;} //Si se presiona * sale del For
   }
   val = atol(str); //Convierte el String en un valor numerico
   return(val);
}

void main()
{
   LCD_INIT();   //Inicializo el LCD
   delay_ms(50);      //permite estabilizar al oscilador
   port_b_pullups (true);  //Utiliza las resistencias PULL UP internas del puerto B
   set_tris_a(0b00000011);          //Pongo el RA0 como entrada
   set_tris_c(0);
```

```
  set_tris_d(0);

  setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM
a 1kHz

  setup_ccp1(ccp_pwm);              //Configurar modulo CCP1 en modo PWM

  set_pwm1_duty(0);                 //Dejo en cero la salida PWM

  //setup_adc_ports(AN0);           //Configurar ADC (Lectura de temperatura)

  //setup_adc(ADC_CLOCK_INTERNAL);      //Reloj interno para la conversion analoga
digital)

  kbd_init();   //inicializa teclado


  //*************************************************************************//

  //*************  DISEÑO POR ASIGNACIÓN DE 2 POLOS REALES   ****************//

  //*************************************************************************//


  TsMA=30;                  //Tiempo deseado en Lazo Cerrado

  Wn=5.8335/(TsMA);         //Frecuencia natural del sistema


  //Ubicación de 2 Polos reales

  P1=2*Wn;

  P2=Wn*Wn;


  kp=(P1*tao-1)/k;        //Calculo de Kc

  ti=(k*kp)/(P2*tao);     //Calculo de ti
```

```
 // Calculo del controlador PID digital
  q0=kp*(1+T/(2*ti)+td/T);
  q1=-kp*(1-T/(2*ti)+(2*td)/T);
  q2=(kp*td)/T;

   for(;;)
{
/*
  set_adc_channel(0);              //Seleccionar Canal 0 para sensor de Temperatura
  adc=read_adc(); //Leer ADC
  delay_us(10); // Delay for stabilization
  voltSalida=adc*((float)vref/(float)1024);//Conversion de bits a voltaje ******DAC
  ym=((voltSalida+0.0075625)/0.00578125)*10.00;//Formula cáculada para el uso de la
                                        //trajeta interfaz
  */
       Sensor=  read_termocupla_k();
       Tm = toFloat_TC(Sensor);
       delay_us(10); // Delay for stabilization
       ym = (Tm)*10;
       //Llama la funcion del controlador PID
```

```
PID();
//tiempo de muestreo
delay_ms(73);

  lcd_gotoxy(2,1);
  lcd_putc("CONTROL PID RES");
  lcd_gotoxy(1,2);
  printf(lcd_putc,"Set-Point  : %3.1f C   ",R/10);
  lcd_gotoxy(21+i,1);
  printf(lcd_putc,"Temperatura: %3.1f C",yM/10);
  sprintf(msg,"%ld\r\n",Sensor-27); //Para Labview
  printf("%s",msg); //Para Labview

//sprintf(msg,"I%3.2fFI%3.2fFI%3.2fF",toFloat_TC(Sensor),toFloat_TC(Sensor),toFloat_TC(Sensor)); //Para Matlab
  //printf("%s",msg); // Para Matlab
  lcd_gotoxy(21,2);
  printf(lcd_putc,"Ley Control: %ld   ",control);

c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
if(c=='D')
{
  LCD_PUTC("\f");
```

```
        lcd_gotoxy(1,2);
        LCD_PUTC("Referencia(0-100):");
        lcd_gotoxy(21,1);
        LCD_PUTC("y presione *");
        lcd_gotoxy(21,2);
        LCD_PUTC("SP:                    ");
        R=escalon(3); //Llama la funcion para digitar el escalon de exitacion
        //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)
        R =(R > 100) ? 1000:R*10;

        //Muestra el SETPOINT en pantalla
        lcd_gotoxy(21,2);
        printf(lcd_putc,"SP: %3.1f        ",R/10);
        delay_ms(2000);
        LCD_PUTC("\f");
      }
 }

}
```

PRÁCTICA # 12: INTERFAZ GRAFICA EN MATLAB PARA EL CONTROL PID DE UNA RESISTENCIA CALEFACTORA.

Matlab:

```
%% Monitoreo en Tiempo Real
function varargout=monitoreo(varargin)
parar=false;
fclose('all')

global tiempo salida escalon y
fig(1)=figure('name','Monitor','menubar','none','position',[200 200 800
700],'color',[0.9 0.6 0.3])
%Texto del Titulo
Texto(1)=uicontrol('parent',fig(1),'style','text','string','Interfaz
PIC','position',[200 600 400 50],'BackgroundColor',[0.9 0.6 0.3],'fontsize',25);
movegui(fig(1),'center');
axe(1)=axes('parent',fig(1),'units','pixels','position',[60 80 600 550],'xlim',[0
40],'ylim',[0 200],'xgrid','on','ygrid','on')
set(get(axe(1),'XLabel'),'String','Tiempo (Seg)')
set(get(axe(1),'YLabel'),'String','Temperatura (°C)')
lin(1)=line('parent',axe(1),'xdata',[],'ydata',[],'Color','r','LineWidth',2.5);
bot(1)=uicontrol('parent',fig(1),'style','pushbutton','string','Detener','position
',[680 50 100 50],'callback',@stop,'fontsize',11)
txbx(1)=uicontrol('parent',fig(1),'style','tex','string','Temperatura','position',
[680 100 100 50],'fontsize',11);
txbx(2)=uicontrol('parent',fig(1),'style','tex','string','Temp °C','position',[680
150 100 50],'fontsize',11)

%% Funcion Pare
    function varargout=stop(hObject,evendata)
        parar=true;
        fclose(SerialP);
        delete(SerialP);
        clear SerialP;

    end
    %% funcion Graficar
   % function varargout=grafique(hObject,evendata)
        tiempo=[0];
        salida=[0];
        escalon=[0];
        deg1="0";

        dt=1;
        limx=[0 40];
        limy=[0 200];
        set(axe(1),'xlim',limx,'ylim',limy);


    %% Configura el Puerto Serial

    SerialP=serial('COM3');
    set(SerialP,'Baudrate',9600); % se configura la velocidad a 9600 Baudios
    set(SerialP,'StopBits',1); % se configura bit de parada a uno
```

```
    set(SerialP,'DataBits',8); % se configura que el dato es de 8 bits, debe estar
entre 5 y 8
    set(SerialP,'Parity','none'); % se configura sin paridad
    fopen(SerialP);

     %% Grafico
     k=5;nit = 10000;
        while(~parar)
            % Lectura del Dato por Puerto Serial
            variable= (fread(SerialP,20,'uchar'));
            ini=find(variable==73); %Busca el retorno de carro (Primer dato)
            ini=ini(1)+1;
            fin=find(variable==70); %Busca operador grados (ultimo dato)
            fin= fin(find(fin>ini))-1;
            fin=fin(1);
            tempC=char(variable(ini:fin))';
            temp=str2num(tempC);

            set(txbx(1),'string',tempC);
            %Actualiza las variables del grafico

             tiempo=[tiempo tiempo(end)+dt];
             salida=[salida temp];
             escalon=[escalon str2num(deg1)];
             set(lin(1),'xdata',tiempo,'ydata',salida);
             pause(dt); %% espera 0.1 seg para cada interación
             if tiempo(end)>=limx % actualizo grafica cuando llega a su limite en
tiempo real
             limx=[0 limx(2)+40];
             set(axe(1),'xlim',limx);
             end

             if salida(end)>=limy % actualizo grafica cuando llega a su limite en
tiempo real
             limy=[0 limy(2)+30];
             set(axe(1),'ylim',limy);
             end

             k=k+1;
             if(k==nit)
                 parar=true;
             end
        end
        parar=false;

end
```

CCS:

#INCLUDE <18F4550.h>// Declaramos al Pic 18F4550

#fuses INTRC_IO, NOWDT, WDT128, HS, NOPROTECT, CPUDIV1, PLL1// Seteamos los fusibles

```
#use delay(clock=20000000)//Seteamos el reloj externo a 20MHz


#use rs232(baud=9600, parity=N, xmit=pin_c6, rcv=pin_c5, bits=8)//Seteamos la comunicación serial

#include "MAX6675_Lib.c"//Incluimos la libreria de la Interfaz para la termocupla

#include <lcd.c>// Incluimos la libreria de la pantalla LCD


//Configura direccion de memoria de los puertos A,B,C,D

#BYTE PORTA=0xF80

#BYTE PORTB=0xF81

#BYTE PORTC=0xF82

#BYTE PORTD= 0xF83

//////////////////////////////////////////////////////

# USE FAST_IO(A)//Directiva para configurar el puerto A en modo rápido(fast) alto rendimiento

# USE FAST_IO(B)//Directiva para configurar el puerto B en modo rápido(fast) alto rendimiento

# USE FAST_IO(C)//Directiva para configurar el puerto C en modo rápido(fast) alto rendimiento

# USE FAST_IO(D)//Directiva para configurar el puerto D en modo rápido(fast) alto rendimiento

#define  KEYHIT_DELAY   1    //Tiempo de espera del teclado en milisegundos
```

```
////////Definimos los pines de la pantalla LCD///////
#define LCD_ENABLE_PIN PIN_D0
#define LCD_RS_PIN PIN_D1
#define LCD_RW_PIN PIN_D2
#define LCD_DATA4 PIN_D4
#define LCD_DATA5 PIN_D5
#define LCD_DATA6 PIN_D6
#define LCD_DATA7 PIN_D7
////////Definimos los pines de la pantalla LCD///////

#define row0 PIN_B0                    //Filas del teclado colocar resistencia pullup
#define row1 PIN_B1                    //Filas del teclado colocar resistencia pullup
#define row2 PIN_B2                    //Filas del teclado colocar resistencia pullup
#define row3 PIN_B3                    //Filas del teclado colocar resistencia pullup
#define col0 PIN_B4                    //Columnas del teclado
#define col1 PIN_B5                    //Columnas del teclado
#define col2 PIN_B6                    //Columnas del teclado
#define col3 PIN_B7                    //Columnas del teclado
#include "Teclado4x4.h"                //Librería de teclado
#include <stdlib.h>//libreria para usar printf
#include <string.h>//Libreria para convertir lo ingresado por el teclado a float
```

```
unsigned int16 adc,control=0;//Lectura de la entrada analógica y variable de control iniciada en 0

float R=750;//Referencia de 45 °C por defecto

float Tm,yM=0,e=0.0,e_1=0.0,e_2=0.0,u=0.0,u_1=0.0,T=0.1; // variables del pid

float kp,ti,td,q0,q1,q2; // variables del pid

float k=0.535,tao=10.0,theta=1.0;// variables del pid

float TsMA,Wn,P1,P2; // variables del pid

char c;// variable para guardar el caracter que se ingresa por el teclado

int i=0; //Posicion del dato recibido a mostrar en el LCD

char msg[32];//La variable Caracter que se va a enviar por comunicación serial

int16 Sensor;   // Definimos las variable de temperatura del sensor

/*================================================================================*/

/*=======================       FUNCION TECLA       =======================*/

/*================================================================================*/

//Funcion encargada de esperar a que se presione una tecla

char tecla(void)

{

  char c;

  do{ //espera hasta que se presione una tecla

     c=kbd_getc(); //Captura valor del teclado

    }

  while(c=='\0');
```

```
  return(c);
}

/*====================================================================
======*/
/*=======================    FUNCION TECLA CON TIMER
=======================*/
/*====================================================================
======*/
// Pregunta por una Tecla por un tiempo, si no hay actividad, deja de preguntar
// y deja que el PIC continue con su trabajo

char tecla_time(void) {
  char c='\0';
  unsigned int16 timeout;
  timeout=0;
  c=kbd_getc(); //Captura valor del teclado
  while(c=='\0' && (++timeout< (KEYHIT_DELAY*150)))
  {
    delay_us(10);
    c=kbd_getc(); //Captura valor del teclado
  }
  return(c);
}
```

```
/*=====================================================================
======*/

/*=======================    FUNCION DEL CONTROL PID
=======================*/

/*=====================================================================
======*/

void PID(void)

{


   e=1*(R-yM);

   // Controle PID

     u = u_1 + q0*e + q1*e_1 + q2*e_2; //Ley del controlador PID discreto


   if (u >= 5000.0)        //Saturo la accion de control 'uT' en un tope maximo y minimo

    u = 5000.0;


   if (u <= 0.0)

    u = 0.0;


    control=u/5;


    //Retorno a los valores reales
```

```
    e_2=e_1;
    e_1=e;
    u_1=u;

    //La accion calculada la transformo en PWM
    set_pwm1_duty(control);

}

/*=====================================================================
======*/
/*============   FUNCION PARA DIGITAR ESCALÓN/SETPOINT
====================*/
/*=====================================================================
======*/
long escalon(int nd)
{
  //Esta funcion captura el escalon desde el teclado, si el proceso está tomando
  //datos el escalon sirve para exitar el sistema, por otro lado si el sistema
  //está controlando, el escalo sirve para establecer el setpoint del sistema

  long val;
  int i;
```

```
  char str[5]; //Variable tipo String


  str[0]='0';
  for(i=0;i<nd;i++)
  {


    c=tecla();  //Lee el valor del teclado y espera hasta que alguna tecla se pulse
    if(c!='*'){
      //Muestra el digito presionado en el LCD
      lcd_gotoxy(26+i,4);
      lcd_putc(c);
      //Almacena el dato presionado en la variable String
      str[i]=c;
    }
    else{i=nd;} //Si se presiona * sale del For
  }
  val = atol(str); //Convierte el String en un valor numerico
  return(val);
}


void main()
{
```

```
   LCD_INIT();   //Inicializo el LCD
   delay_ms(50);      //permite estabilizar al oscilador
   port_b_pullups (true);  //Utiliza las resistencias PULL UP internas del puerto B
   set_tris_a(0b00000011);          //Pongo el RA0 como entrada
   set_tris_c(0);
   set_tris_d(0);
   setup_timer_2(t2_div_by_4,249,1);   //Configuracion de Timer 2 para establecer frec. PWM
a 1kHz
   setup_ccp1(ccp_pwm);               //Configurar modulo CCP1 en modo PWM
   set_pwm1_duty(0);                  //Dejo en cero la salida PWM
   //setup_adc_ports(AN0);             //Configurar ADC (Lectura de temperatura)
   //setup_adc(ADC_CLOCK_INTERNAL);      //Reloj interno para la conversion analoga
digital)
   kbd_init();   //inicializa teclado

   //*************************************************************************//
   //*************  DISEÑO POR ASIGNACIÓN DE 2 POLOS REALES   ****************//
   //*************************************************************************//

   TsMA=30;                  //Tiempo deseado en Lazo Cerrado
   Wn=5.8335/(TsMA);         //Frecuencia natural del sistema

   //Ubicación de 2 Polos reales
```

```
  P1=2*Wn;
  P2=Wn*Wn;

  kp=(P1*tao-1)/k;        //Calculo de Kc
  ti=(k*kp)/(P2*tao);     //Calculo de ti


 // Calculo del controlador PID digital
  q0=kp*(1+T/(2*ti)+td/T);
  q1=-kp*(1-T/(2*ti)+(2*td)/T);
  q2=(kp*td)/T;

   for(;;)
{
/*
  set_adc_channel(0);               //Seleccionar Canal 0 para sensor de Temperatura
  adc=read_adc(); //Leer ADC
  delay_us(10); // Delay for stabilization
  voltSalida=adc*((float)vref/(float)1024);//Conversion de bits a voltaje ******DAC
  ym=((voltSalida+0.0075625)/0.00578125)*10.00;//Formula cáculada para el uso de la
                                        //trajeta interfaz
  */
```

```
        Sensor=  read_termocupla_k();
        Tm = toFloat_TC(Sensor);
        delay_us(10); // Delay for stabilization
        ym = (Tm)*10;
        //Llama la funcion del controlador PID
        PID();
        //tiempo de muestreo
        delay_ms(73);

          lcd_gotoxy(2,1);
          lcd_putc("CONTROL PID RES");
          lcd_gotoxy(1,2);
          printf(lcd_putc,"Set-Point  : %3.1f C   ",R/10);
          lcd_gotoxy(21+i,1);
          printf(lcd_putc,"Temperatura: %3.1f C",yM/10);
          sprintf(msg,"%ld\r\n",Sensor-27); //Para Labview
          printf("%s",msg); //Para Labview

//sprintf(msg,"I%3.2fFI%3.2fFI%3.2fF",toFloat_TC(Sensor),toFloat_TC(Sensor),toFloat_TC(S
ensor)); //Para Matlab
          //printf("%s",msg); // Para Matlab
          lcd_gotoxy(21,2);
          printf(lcd_putc,"Ley Control: %ld   ",control);
```

```
      c=tecla_time();   //Lee el valor del teclado pero solo espera un tiempo determinado
      if(c=='D')
      {
        LCD_PUTC("\f");
        lcd_gotoxy(1,2);
        LCD_PUTC("Referencia(0-100):");
        lcd_gotoxy(21,1);
        LCD_PUTC("y presione *");
        lcd_gotoxy(21,2);
        LCD_PUTC("SP:                    ");
        R=escalon(3); //Llama la funcion para digitar el escalon de exitacion
        //Valida si R esta entre 0 y 100 (Esto es otra forma de usar el if - else)
        R =(R > 100) ? 1000:R*10;

        //Muestra el SETPOINT en pantalla
        lcd_gotoxy(21,2);
        printf(lcd_putc,"SP: %3.1f        ",R/10);
        delay_ms(2000);
        LCD_PUTC("\f");
      }
}
```

```
    }
```

LIBRERÍAS

I2C_FLEX_LCD

```
//---------------------------------------------------------------------------

// Title:         i2c_Flex_LCD

// Description:   Driver for common LCD with 1/2/3 or 4 rows by 1...20 columns

//                using PCF8574T interface board with I2C protocol.

// Date:          Nov-2013

// Ver.Rev.:      1.1

// Author:        Hugo Silva (sergio-hugo@bol.com.br) #Based on the routines of

//                "20X4_LCD_I2C_DRIVER.h" from Pumrin S. and "lcd4_i2c.c" from XP8100

//---------------------------------------------------------------------------

//

// lcd_init() Must be called before any other function.

//

// lcd_putc(c) Will display c on the next position of the LCD.

//

//     \f Clear LCD display

//     \n Set write position on next lcd line

//     \b LCD backspace

//     lcd_gotoxy(x,y) Set write position on LCD (upper left is 1,1)

//
```

```
// lcd_backlight_led(ON)/lcd_backlight_led(OFF) = Turn ON/OFF LCD Backlight LED
//
//------------------------------------------------------------------------
// LCD pins D0-D3 are not used.
//------------------------------------------------------------------------
//
// Comment   : Control of a compatible LCD (1...4 rows by 1...4 columns) from
//             a bus I2C with an EXPANDER of I/O with connection I2C.
//             The tests of these routines have been programmed using the IC
//             Phillips PCF8574T. I've used 4 bits mode programming.
//             The 8 bits mode programming is possible if you use 2 x PCF8574T.
//             RW Pin is not being used.
//
// As defined in the following structure the pin connection is as follows:
//
//  PCF8574P     LCD
//  ========     ======
//     P0        RS
//     P1        RW    (Not used!)
//     P2        Enable
//     P3        Led Backlight
//     P4        D4
```

```
//     P5       D5
//     P6       D6
//     P7       D7
//
//  The SCL and SDA pins should be pull-up resistor as shown below:
//
//            +5v
//             |
//             <
//             > 4.7K
//             <
//To PIC        |        To i2c slave
//pin xx ------------------ SDA pin
//(SDA)
//            +5v
//             |
//             <
//             > 4.7K
//             <
//To PIC        |        To i2c slave
//pin xx ------------------ SCL pin
//(SCL)
```

```
//
//To PIC                    To i2c slave
//Vss pin ----------------- Vss or ground pin
//              |
//             -----
//              ---  Ground
//               -
//
// THIS DOCUMENT IS PROVIDED TO THE USER "AS IS"
//-------------------------------------------------------------------------
byte LCD_ADDR ;//            0x4E        //I2C slave address for LCD module
byte lcd_total_rows ;//      2           //Number of rows: 1,2,3 or 4
byte lcd_total_columns ;//   16          //Number of columns: 1...20

#define RS                  0b00000001  //P0 - PCF8574T Pin connected to RS
#define RW                  0b00000010  //P1 - PCF8574T Pin connected to RW
#define ENABLE              0b00000100  //P2 - PCF8574T Pin connected to EN
#define LCD_BACKLIGHT       0b00001000  //P3 - PCF8574T Pin connected to
BACKLIGHT LED

#define addr_row_one        0x00        //LCD RAM address for row 1
#define addr_row_two        0x40        //LCD RAM address for row 2
#define addr_row_three      0x14        //LCD RAM address for row 3
```

```
#define addr_row_four        0x54       //LCD RAM address for row 4


#define ON                   1

#define OFF                  0

#define NOT                  ~

#define data_shifted         data<<4

int8 new_row_request=1, BACKLIGHT_LED=LCD_BACKLIGHT;


void lcd_backlight_led(byte bl)

{

    If (bl) BACKLIGHT_LED=LCD_BACKLIGHT; else BACKLIGHT_LED=OFF;

}


void i2c_send_nibble(byte data, byte type)

{

  switch (type)

  {

    case 0 :

    i2c_write(data_shifted | BACKLIGHT_LED);

    delay_cycles(1);

    i2c_write(data_shifted | ENABLE | BACKLIGHT_LED );

    delay_us(2);
```

```
      i2c_write(data_shifted & NOT ENABLE | BACKLIGHT_LED);
      break;

      case 1 :
      i2c_write(data_shifted | RS | BACKLIGHT_LED);
      delay_cycles(1);
      i2c_write(data_shifted | RS | ENABLE | BACKLIGHT_LED );
      delay_us(2);
      i2c_write(data_shifted | RS | BACKLIGHT_LED);
      break;
   }
}

void lcd_send_byte(byte data, byte type)
   {
       i2c_start();
       i2c_write(LCD_ADDR);
       i2c_send_nibble(data >> 4 , type);
       i2c_send_nibble(data & 0xf , type);
       i2c_stop();
   }
```

```
void lcd_clear()
{
        lcd_send_byte(0x01,0);
        delay_ms(2);
        new_row_request=1;
}

void lcd_init(byte ADDR, byte col, byte row)
{
   byte i;
   byte CONST lcd_type=2;  // 0=5x7, 1=5x10, 2=2 lines
   byte CONST LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6}; // These bytes need to be sent to the LCD to start it up.

   LCD_ADDR =ADDR;//            0x4E        //I2C slave address for LCD module
   lcd_total_rows =row;//       2           //Number of rows: 1,2,3 or 4
   lcd_total_columns= col ;
   disable_interrupts(GLOBAL);
   delay_ms(50); //LCD power up delay

   i2c_start();
   i2c_write(LCD_ADDR);
      i2c_send_nibble(0x00,0);
```

```
    delay_ms(15);


  for (i=1;i<=3;++i)
  {
    i2c_send_nibble(0x03,0);
    delay_ms(5);
  }
    i2c_send_nibble(0x02,0);
    delay_ms(5);
  i2c_stop();


  for (i=0;i<=3;++i) {
  lcd_send_byte(LCD_INIT_STRING[i],0);
  delay_ms(5);
  }
  lcd_clear();  //Clear Display
  enable_interrupts(GLOBAL);
}


void lcd_gotoxy( byte x, byte y)
{
byte row,column,row_addr,lcd_address;
```

```
static char data;


  if (y>lcd_total_rows) row=lcd_total_rows; else row=y;


  switch(row)
  {
    case 1:  row_addr=addr_row_one;     break;
    case 2:  row_addr=addr_row_two;     break;
    case 3:  row_addr=addr_row_three;   break;
    case 4:  row_addr=addr_row_four;    break;
    default: row_addr=addr_row_one;     break;
  }


  if (x>lcd_total_columns) column=lcd_total_columns; else column=x;
  lcd_address=(row_addr+(column-1));
  lcd_send_byte(0x80|lcd_address,0);
}


//Display the character on LCD screen.
void LCD_PUTC(char in_data)
{
 switch(in_data)
```

```
  {
    case '\f': lcd_clear();                    break;

    case '\n':
    new_row_request++;
    if (new_row_request>lcd_total_rows) new_row_request=1;
    lcd_gotoxy(1, new_row_request);
    break;

    case '\b': lcd_send_byte(0x10,0);          break;

    default: lcd_send_byte(in_data,1);         break;

  }
}
```

Teclado4x4.h

```
// Asignacion de los caracteres a las teclas
char const KEYS[4][4] =
{{'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'*','0','#','D'}};
 /*
{{'1','4','7','*'},
 {'2','5','8','0'},
 {'3','6','9','#'},
 {'A','B','C','D'}};*/

#define KBD_DEBOUNCE_FACTOR 3 // Set this number to apx n/333 where
// n is the number of times you expect
// to call kbd_getc each second
void kbd_init()
{
 set_tris_b(0x0F);   // B7-B4 columnas, B3-B0 filas
  port_b_pullups(true);
}
```

```
short int ALL_ROWS (void)
{
if(input (row0) & input (row1) & input (row2) & input (row3))
   return (0);
else
   return (1);
}

char kbd_getc()
{
static byte kbd_call_count;
static short int kbd_down;
static char last_key;
static byte col;
byte kchar;
byte row;
kchar='\0';
if(++kbd_call_count>KBD_DEBOUNCE_FACTOR)
  {
   switch (col)
     {
```

```
case 0:
  output_low(col0);
  output_high(col1);
  output_high(col2);
  output_high(col3);
  break;

case 1:
  output_high(col0);
  output_low(col1);
  output_high(col2);
  output_high(col3);
  break;

case 2:
  output_high(col0);
  output_high(col1);
  output_low(col2);
  output_high(col3);
  break;

case 3:
```

```
    output_high(col0);
    output_high(col1);
    output_high(col2);
    output_low(col3);
    break;
   }
if(kbd_down)
  {
   if(!ALL_ROWS())
     {
      kbd_down=false;
      kchar=last_key;
      last_key='\0';
     }
  }
else
  {
   if(ALL_ROWS())
     {
      if(!input (row0))
         row=0;
      else if(!input (row1))
```

```
          row=1;
        else if(!input (row2))
          row=2;
        else if(!input (row3))
          row=3;
        last_key =KEYS[row][col];
        kbd_down = true;
       }
     else
      {
       ++col;
       if(col==4)
          col=0;
      }
    }
  kbd_call_count=0;
 }
return(kchar);
}

//=========================
```

MAX6675_Lib.c

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//WEB       :  WWW.MIKROINGENIERIA.COM                                   //
//                                                                        //
//TÍTULO    :  PROYECTOS CON MICROCONTROLADORES                          /
//                                                                        //
//LIBRERIA  :  LIBRERÍA MAX6675                                          //
//                                                                        //
//AUTOR     :  MGTR. ING. BRYAM HUAMANCHUMO BACA                         //
//                                                                        //
//COMPILADOR :  CCS C COMPILER                                           //
//                                                                        //
////////////////////////////////////////////////////////////////////////////

#define SO   PIN_B0

#define CLK  PIN_B1

#define CS   PIN_C7
```

```
int16 thermo_error;                 // Indicador de error global para indicar si el termopar está
conectado o no

void init_max6675(void){

  output_low(CLK);

  output_low(SO);

  output_high(CS);

}

int16 read_max6675(void){           // Tarda 200ms para que el MAX6675 realice una
conversión

  int8 i;

  int16 data;

  output_low(CS);                   // Detiene cualquier proceso de conversión
```

```
  delay_us(1);

  for (i=0;i<16;i++){

    output_high(CLK);

    delay_us(1);

    output_low(CLK);

    shift_left(&data,2,input(SO));   // Lectura de  2 bytes de datos por el pin SO

  }

  thermo_error=bit_test(data,2);      // Bit de estado del termopar

  output_high(CS);

  return(data);

}
```

```
int16 sortout(int16 raw){

    return(0x0FFF & (raw>>4));         // Devuelve los bits relativos a la temperatura

}

float toFloat_TC(int16 tmp){

   return((float)tmp-35.00);              // Ajusta los datos a formato de punto flotante

}

int16 read_termocupla_k(void){

   init_max6675();

   delay_ms(100);

   return(sortout(read_max6675())/2);

}
```